Scalable Hardware-Algorithms for Binary Prefix Sums ⋆

R. Lin[1], K. Nakano[2], S. Olariu[3], M. C. Pinotti[4], J. L. Schwing[5], and A. Y. Zomaya[6]

[1] Department of Computer Science, SUNY Geneseo, Geneseo, NY 14454, USA
[2] Department of Electrical and Computer Engineering, Nagoya Institute of Technology, Showa-ku, Nagoya 466-8555, Japan
[3] Department of Computer Science, Old Dominion University, Norfolk, VA 23529, USA
[4] I.E.I., C.N.R, Pisa, ITALY
[5] Department of Computer Science, Central Washington University, Ellensburg, WA 98926, USA
[6] Parallel Computing Research Lab, Dept. of Electrical and Electronic Eng, University of Western Australia, Perth, AUSTRALIA

Abstract. The main contribution of this work is to propose a number of broadcast-efficient VLSI architectures for computing the sum and the prefix sums of a w^k-bit, $k \geq 2$, binary sequence using, 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 eminently practical. Using our design, the sum of a w^k-bit binary sequence can be obtained in the time of $2k-2$ broadcasts, using $2w^{k-2} + O(w^{k-3})$ blocks, while the corresponding prefix sums can be computed in $3k-4$ broadcasts using $(k+2)w^{k-2} + O(kw^{k-3})$ blocks.

1 Introduction

Recent advances in VLSI have made it possible to implement algorithm-structured chips as building blocks for high-performance computing systems. Since computing binary prefix sums (BPS) is a fundamental computing problem [1, 4], it makes sense to endow general-purpose computer systems with a special-purpose parallel BPS device, invoked whenever its services are needed. Recently, Blelloch [1] argued convincingly that *scan* operations – that boil down to parallel prefix computation – should be considered primitive parallel operations and, whenever possible, implemented in hardware. In fact, scans have been implemented in microcode in the Thinking Machines CM-5 [10].

Given an n-bit binary 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 j-th *prefix sum*. The BPS problem is to compute all the prefix sums $p_1, p_2, \cdots, p_n$ of A. In this article, we address the problem of designing efficient and scalable hardware-algorithms for the BPS problem. We adopt, as the central theme of this effort, the recognition of the fact that the delay incurred by a signal propagating along a medium is, at best, linear in the distance traversed. Thus, our main design criterion is to keep buses as short as possible. In this context, the main contribution of this work is to show that we can use short buses in conjunction with the shift switching

⋆ Work supported by NSF grants CCR-9522092 and MIP-9630870, by NASA grant NAS1-19858, by ONR grants N00014-95-1-0779 and N00014-97-1-0562 and by ARC grant 04/15/412/194.

technique introduced recently by Lin and Olariu [5] to design a scalable hardware-algorithm for BPS. As a byproduct, we also obtain a scalable hardware-algorithm for a parallel counter, that is, for computing the sum of a binary sequence (BS).

A number of algorithms for solving the BS and BPS problems on the reconfigurable mesh have been presented in the literature. For example, Olariu *et al.*[8] showed that an instance of size m of the BPS problem can be solved in $O(\frac{\log n}{\log m})$ time on a reconfigurable mesh with $2n \times (m+1)$, $(1 \leq m \leq n)$, processors. Later, Nakano [6, 7] presented $O(\frac{\log n}{\sqrt{m \log m}})$ time algorithms for the BP and BPS problems on a reconfigurable mesh with $n \times m$ processors. These algorithms exploit the intrinsic power of reconfiguration to achieve fast processing speed. Unfortunately, it has been argued recently that the reconfigurable mesh is unrealistic as the broadcasts involved occur on buses of arbitrary number of switches.

Our main contribution is to develop scalable hardware-algorithms for BS and BPS computation under the realistic assumption that blocks of size no larger than $w \times w^2$ are available. We will evaluate the performance of the hardware-algorithms in terms of the number of broadcasts executed on the blocks of size $w \times w^2$. Our designs compute the sum and the prefix sums of w^k bits in $2k-2$ and $3k-4$ broadcasts for any $k \geq 2$, respectively. Using pipelining, one can devise a hardware-algorithm to compute the sum of a kw^k-bit binary sequence in $3k + \lceil \log_w k \rceil - 3$ broadcasts using $2w^{k-2} + O(kw^{k-3})$ blocks. Using this design, the corresponding prefix sums can be computed in $4k + \lceil \log_w k \rceil - 5$ broadcasts using $(k + \lceil \log_w k \rceil + 2)w^{k-2} + O(kw^{k-3})$ blocks. Due to the stringent page limitation, we omit the description of the pipelining-scaled architecture.

2 Linear arrays of shift switches

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, $(1 \leq i \leq n)$, *Binary:* $x_i = \lfloor \frac{z}{2^{i-1}} \rfloor$ mod 2; *Unary:* $x_i = 1$ if $i = z+1$ and 0 otherwise; *Filled unary:* $x_i = 1$ if $1 \leq i \leq z$ 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, $(w \geq 2)$, $x_i = \lfloor \frac{z}{w^{i-1}} \rfloor$ mod w; *Unary base w:* Exactly like the base w representation except that x_i is represented in unary form. The task of converting back and forth between these representation is straightforward and can be readily implemented in VLSI [2, 3, 9]

Fix a positive integer w, typically a small power of 2. At the heart of our designs lays a simple device that we call the $S(w)$. Referring to Figure 1, the input to the $S(w)$ consists of an integer b, $(0 \leq b \leq w-1)$, in unary form and of a bit a. The output is an integer c, $(0 \leq c \leq w-1)$, in unary form, along with a bit d, such that

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

We implement the $S(w)$ using the shift switch concept proposed in [5]. As illustrated in Figure 1, in this implementation, the $S(w)$ features w switching elements with the state changes controlled by a 1-bit state register. 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

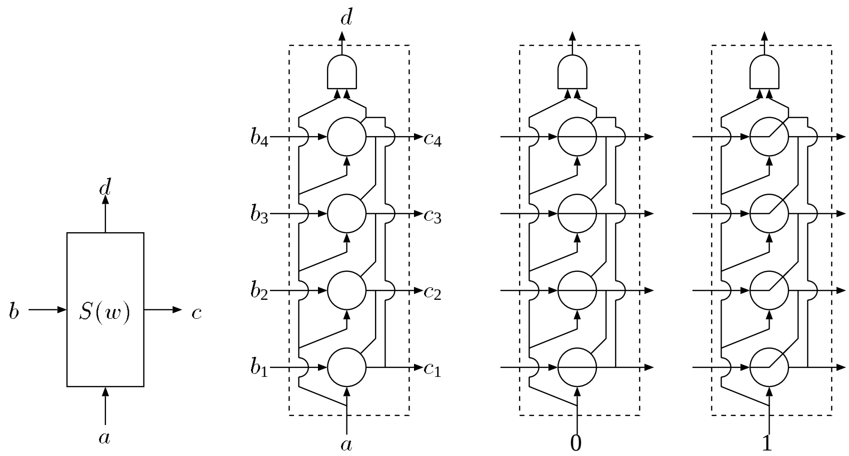

Fig. 1. *Illustrating the functional view of the $S(w)$ and its shift switch implementation*

shifts b if $a = 1$; 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*.

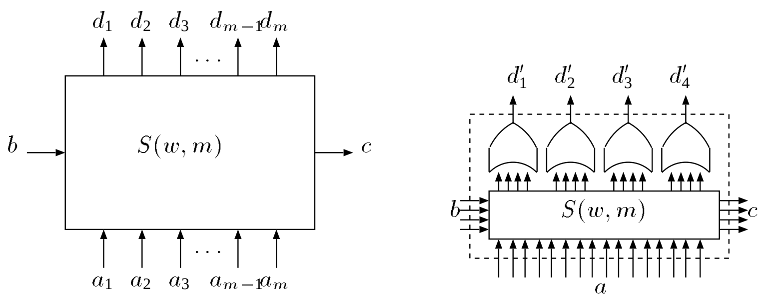

Fig. 2. *A functional view of the array $S(w, m)$ and of block $T(w, m)$*

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 Figure 2.

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

$$c = (a_1 + a_2 + \cdots + a_m + b) \bmod w \quad (2)$$

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 a 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, $(1 \leq i < m')$,

$$d_{(i-1)\cdot w+1} + d_{(i-1)\cdot w+2} + \cdots + d_{i \cdot w} = d_{(i-1)\cdot w+1} \vee d_{(i-1)\cdot w+2} \vee \cdots \vee d_{i\cdot w} \tag{4}$$

and, similarly,

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

Now (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 above, the linear array $S(w, m)$ will be denoted by $T(w, m)$. We refer the reader to Figure 2 for an illustration. Consequently, we have the following result.

Lemma 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.*

3 Computing sums on short buses

Our design relies on a new block $U(w, w^k)$, $(k \geq 1)$, 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 $a_1 + a_2 + \cdots + a_{w^k}$. We note that A_{k+1} is 1 only if $a_1 = a_2 = \cdots = a_{w^k} = 1$. Recall that for every i, $(1 \leq i \leq k)$, $0 \leq A_i \leq w - 1$.

The block $U(w, w^k)$ is defined recursively as follows. $U(w, w^1)$ is just a $T(w, w)$. $U(w, w^2)$ is implemented using blocks $T(w, w)$ and $T(w, w^2)$: The w^2 input are supplied to $T(w, w^2)$. $T(w, w^2)$ outputs the LSD (Least Significant Digit) of the sum in unary base w representation, and the MSD (Most Significant Digit) of the sum in distributed representation. $T(w, w)$ is used to convert the distributed representation into the unary base w representation of the MSD of the sum.

Let $k \geq 3$ 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 Figure 3 for an illustration of this construction for $w = 4$ 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 i-th 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, $(2 \le i \le k)$,

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

Similarly, $A_{k+1} = D_k$. The equations above confirm that the design in Figure 3 computes the unary base w representation of the sum $a_1 + a_2 + \cdots + a_{w^k}$. Specifically, the rightmost $T(w, w^2)$ computes A_1. It receives the unary representation of 0 (i.e. $00 \cdots 01$) 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. 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. In this figure, the circles indicate circuitry that converts from unary to filled unary representation.

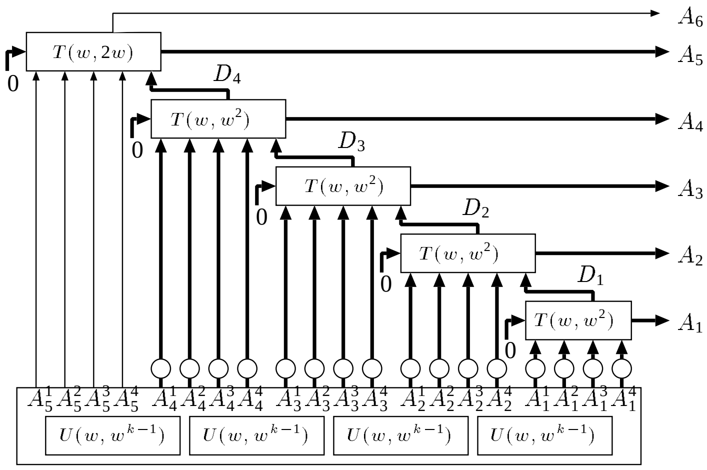

Fig. 3. *Illustrating the recursive construction of* $U(w, w^k)$

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

Lemma 2. *For every k, $(k \geq 2)$, and for every i, $(1 \leq i \leq k)$, 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 $U(w, w^2)$ 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, $(1 \leq i \leq k)$, is computed in $k+i-2$ broadcasts and that A_{k+1} is computed in $2k-2$ broadcasts, as claimed.

We now estimate the number of blocks of type $T(w, m)$ with $m \leq w^2$ involved in the construction of $U(w, w^k)$.

Lemma 3. *A block $U(w, w^k)$, $(k \geq 2)$, involves $2w^{k-2} + O(w^{k-3})$ 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 Figure 6, $U(w, w^k)$ 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 $g(k) = 2w^{k-2} + O(w^{k-3})$, thus completing the proof of the lemma.

Now, Lemmas 2 and 3, combined, imply the following important result.

Theorem 4. *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 $T(w, m)$, with $m \leq w^2$.*

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.

Observe that each block $U(w, w^k)$ has, essentially, the structure of a w-ary tree as illustrated in Figure 4. The tree has nodes at $k-1$ levels from 2 to k. For an arbitrary node v at level h in 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 Figure 5. Let $t(v)$ and $p(v)$ denote the *sum of* v and the *prefix sum of* v defined as follows: $t(v) = a_{s+1} + a_{s+2} + \cdots + a_{s+w^h}$, and $p(v) = a_1 + a_2 + \cdots + a_s$. 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

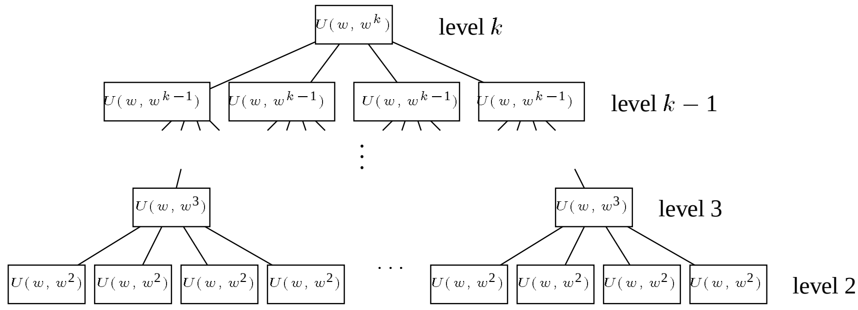

Fig. 4. *Illustrating the w-ary tree structure of $U(w, w^k)$ for $w = 4$*

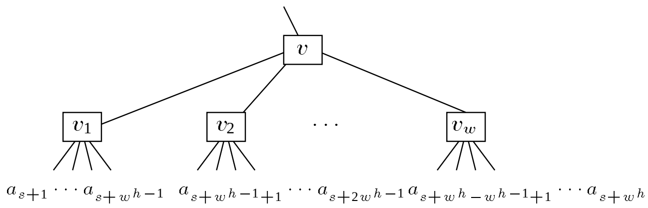

Fig. 5. *A node v and its w children $v_1, v_2, \ldots, v_w$*

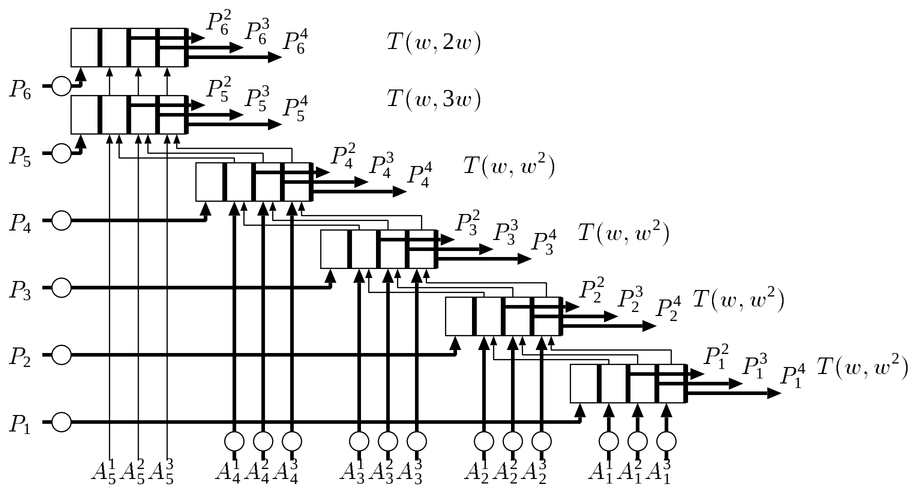

Fig. 6. *Illustrating a design for computing the prefix sums $P_k^j P_{k-1}^j \cdots P_1^j$*

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 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 $A_h^jA_{h-1}^j\cdots A_1^j$, ($1 \le j \le w$), 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 $P_{k+1}^jP_k^jP_{k-1}^j\cdots P_1^j$, ($1 \le j \le w$), be the unary base w representations the prefix sum $p(v_j)$ of v_j. Notice that, 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 $p(v) = P_kP_{k-1}\cdots P_1$ of v and the sum $t(v_j) = A_h^jA_{h-1}^j\cdots A_1^j$ of each child v_j, ($1 \le j \le w-1$) of v;
Output: The prefix sum $p(v_j) = P_k^jP_{k-1}^j\cdots P_1^j$ of each child v_j, ($2 \le j \le w$), of v.

Note that we do not have to compute $p(v_1) = P_k^1P_{k-1}^1\cdots P_1^1$ 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. Figure 6 illustrates this task for $w = 4$, $h = 5$, and $k = 6$.

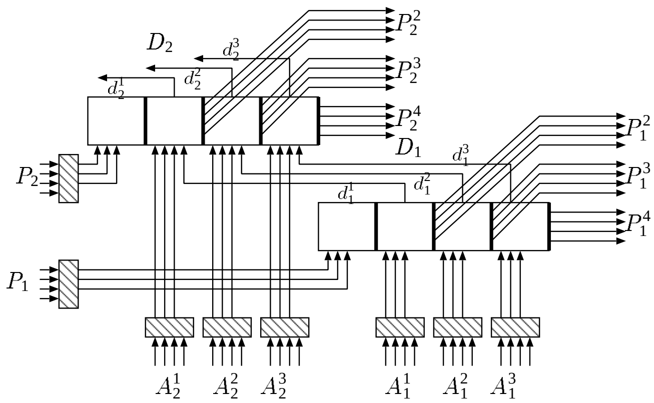

Fig. 7. *Illustrating the details of the prefix sums design of Figure 6*

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 $t(v_j) = A_k^jA_{k-1}^j\cdots A_1^j$ in $2k-4$ broadcasts. By Lemma 2 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 Figure 6, P_i^j, $(1 \leq i \leq k)$, 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 5. *Our prefix sums design returns* P_i^j, $(1 \leq i \leq k; 1 \leq j \leq w)$, *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 4 $U(w,w^k)$ 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 6. *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.

References

1. G. Blelloch, Scans as primitives parallel operations, *IEEE Transactions on Computers*, C-18, (1989), 1526–1538.
2. J. J. F. Cavanaugh, *Digital Computer Arithmetic Design and Implementation*, McGraw-Hill, New York, 1984.
3. H. T. Kung and C. E. Leiserson, Algorithms for VLSI processor arrays, in C. Mead and L. Conway, Eds, *Introduction to VLSI Systems*, Addison-Wesley, Reading MA, 1980.
4. R. E. Ladner and and M. J. Fischer, Parallel prefix computation, *Journal of the ACM*, 27, (1980), 831–838.
5. R. Lin, and S. Olariu, Reconfigurable buses with shift switching – architectures and applications, *IEEE Transactions on Parallel and Distributed Systems*, 6, (1995), 93–102.
6. K. Nakano, An efficient algorithm for summing up binary values on a reconfigurable mesh, *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, E77-A, 4, (1994), 652–657.
7. K. Nakano, Prefix-sums algorithms on reconfigurable meshes, *Parallel Processing Letters*, **5**, (1995), 23-35.
8. S. Olariu, J. L. Schwing, and J. Zhang, Fundamental data movement algorithms for reconfigurable meshes, *International Journal of High Speed Computing*, **6**, (1994), 311–323.
9. E. E. Swartzlander, Jr., *Computer Arithmetic* Vol. 1, IEEE CSP, 1990.
10. Thinking Machines Corporation, Connection machine parallel instruction set (PARIS), July 1986.

This article was processed using the LaTeX macro package with LLNCS style