Optimal Circuits for Parallel Multipliers

Paul F. Stelling, *Member*, *IEEE*, Charles U. Martel,
Vojin G. Oklobdzija, *Fellow*, *IEEE*, and R. Ravi

Abstract—We present new design and analysis techniques for the synthesis of parallel multiplier circuits that have smaller predicted delay than the best current multipliers. In [4], Oklobdzija et al. suggested a new approach, the Three-Dimensional Method (TDM), for Partial Product Reduction Tree (PPRT) design that produces multipliers that outperform the current best designs. The goal of TDM is to produce a minimum delay PPRT using full adders. This is done by carefully modeling the relationship of the output delays to the input delays in an adder and, then, interconnecting the adders in a globally optimal way. Oklobdzija et al. suggested a good heuristic for finding the optimal PPRT, but no proofs about the performance of this heuristic were given. We provide a formal characterization of optimal PPRT circuits and prove a number of properties about them. For the problem of summing a set of input bits within the minimum delay, we present an algorithm that produces a minimum delay circuit in time linear in the size of the inputs. Our techniques allow us to prove tight lower bounds on multiplier circuit delays. These results are combined to create a program that finds optimal TDM multiplier designs. Using this program, we can show that, while the heuristic used in [4] does not always find the optimal TDM circuit, it performs very well in terms of overall PPRT circuit delay. However, our search algorithms find better PPRT circuits for reducing the delay of the entire multiplier.

Index Terms—Multiplier design, partial product reduction, algorithms, circuit design.

1 Introduction

The design of efficient logic circuits is a fundamental problem in the design of high performance processors. The design of fast parallel multipliers is important, since multiplication is a commonly used and expensive operation. This is particularly critical for specialized chips that support multiplication intensive operations, such as digital signal processing and graphics. It can also be useful for pipelined CPUs, where faster multiplier components and multipliers can result in smaller clock cycles and/or shorter pipelines. There have been many research projects and papers on the design of fast parallel multipliers; these results are surveyed in [1], [3], [17]. Continuing research in the area has led to a steady improvement in the designs for Partial Product Reduction Trees (PPRTs) for parallel multiplier designs, as evidenced in the progression of work in [18], [2], [12], [10], [11], [6]. However, almost all of this prior work focused on finding good basic building blocks (compressors) that could be connected in a regular pattern to build a PPRT. A compressor operates in a single column of the PPRT, taking as input some number (b) of bit signals for the column and a number (c_i) of carry-in signals from the previous column. It outputs some lesser number ($o < b$) of signals in its column, as well as a like number ($c_o = c_i$) of carry-out signals. These compressors are made up of full adders that are interconnected in a way to minimimize the compressor's delay. In contrast, our approach is to design a faster PPRT by finding a globally optimal way of interconnecting the low-level components (adders).

We now discuss the design problem in more detail (a complete description is given in Section 2). Parallel multiplication problem of two n-bit numbers is done in three steps: Computing the partial products; using a PPRT to add $(2n - 1)$ columns of bits, giving two bits for each column (carrys from each column are incorporated into the next); and adding the two $(2n - 1)$-bit numbers using a final (carry-propagate) adder (see Fig. 1). The basic problems we address here relate to designing fast PPRTs. Our results also apply to asymmetric multiplication, where the bit sizes of the two factors differ, and to the Multiply-Accumulate (MAC) operation.

In [4], the Three-Dimensional Method (TDM) for globally designing the PPRT of a parallel multiplier circuit is described. The goal of the TDM is to produce a minimum delay PPRT using full adders ((3, 2) adders) and a small number of half adders ((2, 2) adders). In [4], it was shown that the TDM has the advantages of minimizing the number of devices required for the PPRT and allowing the use of standard ASIC formats and utilities for generating layouts for tree-based. The speed improvements are achieved by carefully modeling the relationships of the output delays to the input delays in an adder, and then interconnecting the adders in a globally optimal way. They describe a linear-time heuristic for constructing PPRTs that outperform the best current designs. However, the question of whether the algorithm always derives the optimal TDM-based PPRT was left open. In this paper, we describe a number of new techniques for designing and analyzing PPRTs. Specifically, we describe:

- *P.F. Stelling is with The Aerospace Corporation, P.O. Box 92957/M1/102, Los Angeles, CA 90009-2957. E-mail: stelling@aero.org.*
- *C. Martel is with the Department of Computer Science, University of California at Davis, Davis CA 95616. E-mail: martel@cs.ucdavis.edu.*
- *V. Oklobdzija is with Integration, 1442A Walnut Street #412, Berkeley, CA 94709. E-mail: vojin@nuc.berkeley.edu.*
- *R. Ravi is with the Graduate School of Industrial Management, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15217. E-mail: ravi+@andrew.cmu.edu.*

Manuscript received 17 Sept. 1996; revised 19 Jan. 1998.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 102099.

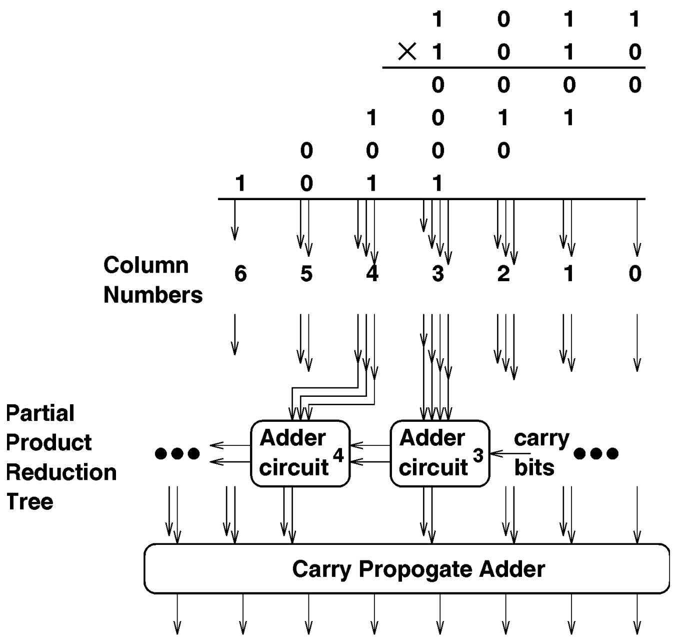


Fig. 1. The basic design of the parallel multiplier. Each column of partial product bits is added in a circuit that generates two sum bits and a number of carry bits that feed into the adder circuit for the next column. The output bits from the columns are summed in a fast adder to produce the final result of the multiplication.

1) An optimal class of circuits that can be used in constructing PPRTs.
2) A linear time algorithm for producing a delay-optimal circuit for summing a set of input bits. This circuit uses a network of full adders to implement a general parallel counter.
3) A program that finds optimal PPRT circuits.

Using the program, we have found optimal PPRT circuits for multiplying two numbers of size up to 40 bits. With improvements, we will be able to solve larger problems. Our results show that the heuristic in [4] gives PPRT circuits with optimal or near-optimal overall delay. However, our program finds PPRT circuits with better delay profiles. The techniques we developed allow the program to sharply prune the search space of PPRT circuits and, thus, allow optimal circuits of larger sizes to be designed.

The basic component we use in our PPRT design is a *Full Adder*, which takes three input bits available at times $a \le b \le d$ and produces a sum bit at time $s = \max(b + x_2, d + x_3)$ ($x_2 \ge x_3 \ge 0$) and a carry bit at time $c = d + y_3$ ($y_3 \ge 0$), where x_2, x_3, and y_3 are technology dependent (the subscript i is used for the incremental delay that gets added to the ith input of the gate). We denote such a full adder by $g = (a, b, d)$, and use subscripting to refer to a specific adder (e.g., $g_i = (a_i, b_i, d_i)$ with sum output s_i and carry output c_i). This very general model applies to current technology and is likely to hold for any foreseeable technologies. Except where noted, our results hold for this general model, and, in fact, apply so long as the function for c is nondecreasing in d, and the function for s is nondecreasing in b and also nondecreasing in d for fixed b. Some of our results apply only to more restricted cases that reflect current technology.

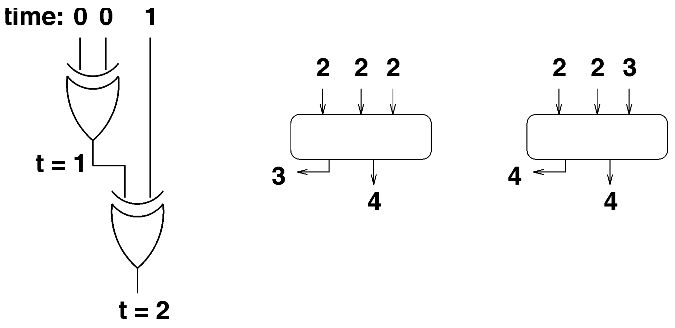


Fig. 2. Adder delays in the standard problem. Two inputs available at time 0 traverse two XOR gates and one input available at time 1 traverses a single XOR gate. Therefore, for inputs $a \le b \le d$, the sum s is available at time $s = \max(b + 2, d + 1)$.

Based on current technology, the time needed by an adder to generate its outputs from its inputs can be normalized to XOR delay units, corresponding to the number of XOR gates (or approximate equivalent) traversed by the input signals. Thus, the values "$x_2 = 2$" and "$x_3 = 1$" can be used as the delays for the sum, corresponding to the number of XOR gates traversed by the b and d inputs, respectively (see Fig. 2). Similarly, the value "$y_3 = 1$" can be used for the carry delay, corresponding to the time needed for the d input to traverse two NAND gates, each having delay roughly half that of an XOR gate [19]. Under these conditions, the global optimization problem can be viewed as minimizing the number of equivalent XOR gates on the longest path in the PPRT circuit. Thus, our more restricted results apply to any technology where the number of equivalent XOR gates on a path remains the critical delay of the circuit, and the relative delays of XOR and NAND gates remain unchanged. Both these assumptions have continued to hold during the rapid development of logic technologies in the past [3]. We will refer to the case with the normalized values of $x_2 = 2$, $x_3 = 1$, $y_3 = 1$ as the *standard problem*, and will, for simplicity, assume the standard problem in all examples.

Optimizing for delay using the full adder model described above leads to improved performance in real circuit designs. In [4], designs in 1 micron CMOS-ASIC technology were simulated using a timing simulator from LSI Logic [19]. The simulated delays closely matched those predicted by using the delays of the standard full adder model described above, and the new design outperformed competing designs by 11-25 percent. Thus, the optimization problem we address here seems to be a sound model of actual circuit delays, and can provide substantial improvements in performance. Further, the PPRT designs we develop use local connections between gates and are similar in overall structure to classical PPRT designs. Thus, there should be no special problems in their layout or wiring.

Paterson et al. [7], [8], [9] also looked at optimal delay circuits for adding multiple columns of bits down to two bits per column. However, they considered designs where the columns use identical circuits, rather than designs that optimize across all columns which is our focus. Using global optimization, we are able to design multiplication circuits whose delays beat the upper bounds provable for their more restricted circuits. Similarly, our design approach for the circuit corresponding to each column uses a different model than theirs: We assume that the carries coming into each column can come at arbitrary times rather than from an identical column circuit. Thus, for the single column problem, our results are orthogonal to their work.

Our approach also recognizes that, with the TDM (as with other typical PPRT designs), the latest sum bits for the columns do not all arrive at the same time. As a result, we are not just concerned with the maximum delay for any column, but also with the profile of the individual column delays and its impact on the output delay of the final adder. We address this issue further in Subsection 2.1 and Section 6.

In the next section, we describe the multiplier design problem in more detail. In Section 3, we study properties of optimal carry vectors and of circuits used in optimal TDM designs. In Section 4, we study the objective of minimizing only the delay of an output bit in designing adder circuits and present an optimal strategy for doing this. In Section 5, we present a lower bound for the global problem derived by analyzing a relaxed version of the problem. In Section 6, we describe our program that uses the prior results to search for optimal PPRT circuits. Finally, in Section 7, we present some open problems related to this work.

2 The Multiplier Setting

We now give a more detailed description of the PPRT design problem. In the long multiplication method, we compute the partial products and arrive at $(2n - 1)$ addition problems (one for each column of the long multiplication table and incorporating the (possible) carries generated by the preceding columns), the results of which are input to the final adder, which outputs the product. We number the columns right to left (least significant to most significant) from 0 to $2n - 2$ (see Fig. 1). We assume that all of the n^2 partial product bits are available at time 0, thus ignoring variations in their propagation delays.

All the bits in column i represent ith significant bits in the product and, thus, have weight 2^i. A full adder working on three inputs of weight 2^i for any $i \geq 0$ produces a sum bit of weight 2^i and a carry bit of weight 2^{i+1}. The carry bits from the full adders summing the bits in column i are, thus, fed as inputs to the addition problem in column $(i + 1)$. The sorted list of times at which the carry bits are produced in a circuit is called its *carry vector*. Thus, the inputs for each column consist of the partial product bits for that column available at time zero and the carry bits from the previous column available at the times designated in the carry vector. The problem of building a fast PPRT using the TDM therefore reduces to one of finding column circuits that use full adders to yield the two sum bits with little delay and generate "good" carry vectors.

2.1 Two Addition Problems

The PPRT of a parallel multiplier outputs two bits for each column i (except column 0, with one bit). These bits are used to form two $(2n - 1)$-bit numbers X and Y, which are fed into a carry propagate adder to produce the final multiplication value (see Fig. 1). The total time to carry out the multiplication is the time to generate partial product bits, complete the column additions in the PPRT, and add the two final $(2n - 1)$-bit numbers.

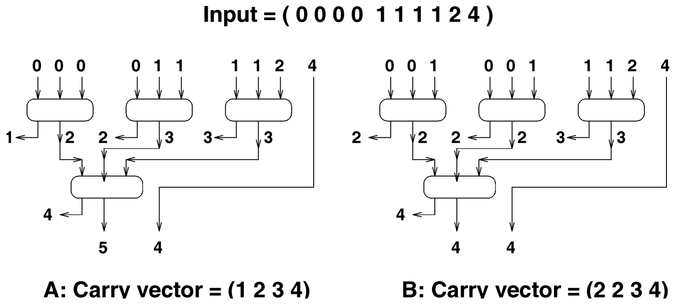

Input = (0 0 0 0 1 1 1 1 2 4)
0 0 0 0 1 1 1 1 2 4
1 2 2 3 3 3
4 5 4
A: Carry vector = (1 2 3 4)
0 0 1 0 0 1 1 1 2 4
2 2 2 2 3 3
4 4 4
B: Carry vector = (2 2 3 4)

Fig. 3. Under the delays of the standard model, circuit *A* produces a better carry vector than circuit *B*, while *B* achieves better final delay than *A* in summing all the input bits to two bits.

In the TDM, each column circuit takes as input the partial product bits corresponding to that column and any carry bits generated by the previous column and adds them to produce two sum bits and a number of carry bits for input to the next column. Fig. 3a illustrates a circuit that takes 10 input bits of weight 2^i. The circuit produces two output bits of weight 2^i at times 4 and 5, and four carry bits of weight 2^{i+1} at times 1, 2, 3, and 4.

In this section, and continuing through Section 3, we describe ways to decrease multiplication time by minimizing the time when the last sum bit from any column is produced. In Section 6, we look at a more global optimization that, by considering the times when the last sum bit from each column is produced, jointly optimizes the PPRT with the final adder to give an optimal multiplier.

2.2 Adjusting the Parity

It is not possible to produce two sum bits from a circuit that uses full adders alone (and all inputs of those adders) when the number of bits to be summed is odd. Consider, e.g., the case when three bits are to be summed. Using a full adder on these bits produces a single sum bit. A full adder "consumes" three of the bits to be summed and produces one new bit to be summed, thus reducing the number of bits to be summed by two. If we start with an even number of bits to be summed, we can use full adders to get down to two sum bits as required for each column. Note that if we have $2(i + 1)$ bits to be summed, we can do this using i full adders each of which produces a carry bit. However, starting with an odd number of bits results in a state where only one bit remains.

We fix this parity problem using a half adder, which takes two input bits and produces a sum bit and a carry bit. (We use a half adder because it is faster and more efficient than using only two inputs of a full adder.) We denote by (a, b) a half adder that takes in two bits at times $a \leq b$, and produces a sum bit at time $s = b + v_2$ ($v_2 \geq 0$) and a carry bit at time $c = b + w_2$ ($w_2 \geq 0$) [4]. As in the case of full adders, for analysis relating to current technology, we use values corresponding to normalized XOR delay units, i.e., $v_2 = 1$ and $w_2 = 0.5$ (for one AND gate). Since the half adder reduces the number of bits to be summed by one, a single half adder fixes the parity problem when the number of bits to be summed is odd. Following [4], we use the half adder on

TABLE 1
NUMBER OF INPUT BITS AND GATES IN A PPRT FOR n-BIT MULTIPLICATION ($2 \le i \le 2n - 2$)

Column i:	$i \le n - 1$	$i = n$	$i \ge n + 1$
# Partial Product Bits	$i + 1$	$i - 1$	$2n - i - 1$
# Carry-In Bits	$i - 2$	$i - 2$	$2n - i - 1$
Total Input Bits	$2i - 1$	$2i - 3$	$4n - 2i - 2$
# Half Adders	1	1	0
# Full Adders	$i - 2$	$i - 3$	$2n - i - 2$
Total Half Adders	$n - 1$		
Total Full Adders	$(n - 1)(n - 3)$		

the earliest two inputs in the input vector if necessary to change the parity of the number of bits to be even. To add $(2i + 1)$ bits for a column in this way, we use a single half adder and $(i - 1)$ full adders, generating a total of i carry bits. We leave as future work the issue of using the half-adder on other (later) inputs.

The number of input bits and gates for the columns are shown in Table 1. Note that the number of half adders and full adders used varies only with the size of the multiplication, and not with the manner in which they are interconnected.

2.3 Circuit Notation and Definitions

We now present a formal description of the TDM.

DEFINITION. *A* **TDM PPRT** *uses column circuits to produce two sum bits for each column (except column 0). The (acyclic) circuit for each column uses the minimum number of full adders (plus one half adder on the earliest inputs when the number of inputs is odd) needed to produce the output bits from the partial product bits for that column and the carry-in bits (if any) from the previous column.*

The following definitions and notations will be used to simplify our discussion of column circuits. (For conciseness, we use *circuit* to mean a column circuit in a TDM PPRT.) We denote by I the vector representing the bits input to a circuit. We use $C(I)$ to denote the circuit C applied to input vector I. The values in I correspond to the *times* at which the signals are available in non-decreasing order. If C has m full adders and I consists of k input bits, then $C(I)$ will produce a vector V_s of the times when the $(k - 2m)$ output sum bits become available in nondecreasing order, and a similar vector V_c for the m carry bits. (If C has $(m - 1)$ full adders and one half adder, then V_s will have $(k - 2m + 1)$ output sum bits and V_c has m carry bits.)

We now prove a number of properties about circuits of adders that we will apply in the construction of column circuits for PPRTs.

We use the following definition when comparing input or output vectors:

DEFINITION. *For two vectors $V = (v_1, v_2, \ldots, v_k)$, $U = (u_1, u_2, \ldots, u_k)$, we say that V* **dominates** *U iff $v_i \le u_i$ $\forall i \in \{1, 2, \ldots, k\}$. We write $V \le U$ to denote that V dominates U. If the two vectors are unequal, then V* **strictly dominates** *U, and we can write $V < U$. Two vectors are* **incomparable** *if there exists i, j such that $v_i < u_i$ and $v_j > u_j$. For a set of equal length vectors S, a vector $V \in S$ is said to be* **undominated** *in S if no other vector in S strictly dominates it.*

For example, if $U = (1, 2, 3)$, $V = (2, 2, 3)$, and $W = (1, 2, 4)$ then $U < V$, $U < W$, and V and W are incomparable.

We also extend the definition of dominance to circuits and gates: If m-adder circuits $C(I)$ and $C'(I)$ produce vectors V_c and V_s, and V_c' and V_s', respectively, then $C(I)$ *dominates* $C'(I)$ iff $V_c \le V_c'$ and $V_s \le V_s'$, and we use the notation $C(I) \le C'(I)$. $C(I)$ *strictly dominates* $C'(I)$ if $C(I) \le C'(I)$ and $V_c \ne V_c'$ or $V_s \ne V_s'$, and we can write $C(I) < C'(I)$. If no m-adder circuit for I strictly dominates $C(I)$, then we say that $C(I)$ is an *undominated circuit*.

Our main focus is on finding PPRTs for minimum delay parallel multipliers and, therefore, on minimum delay PPRT circuits. Thus, for column circuits in a TDM PPRT, we will only be concerned with the output delay of the latest sum bit and the carry vector. (The delay of the final adder depends on the delay of the latest (sum) input bit for each of the columns, and each column circuit depends on the carry bits from the previous column.) Note that an undominated column circuit for input I need not be part of an optimal PPRT, since two circuits $C(I)$ and $C'(I)$ may be incomparable. However, if we construct a multiplier circuit by summing the bits in each column i (which are the original bits of weight 2^i and the carry bits from column $i - 1$), then, for column circuits $C(I) \le C'(I)$, we have that $C(I)$ generates its latest sum output at the same or earlier time than $C'(I)$ and produces an equal or better carry vector. Thus, it is clear that we can always construct an optimal circuit by using an undominated circuit for each column. Similar to the case for circuits, we say that for two gates g_i and g_i', g_i *dominates* g_i' ($g_i \le g_i'$) if $c_i \le c_i'$ and $s_i \le s_i'$. Note that, in the case of gates, we remove the restriction that the inputs be the same. This leads us immediately to the following observation:

OBSERVATION 2.1. *The outputs s_i and c_i of gate g_i depend only on the inputs b_i and d_i. Therefore, g_i dominates g_j if $b_i \le b_j$ and $d_i \le d_j$.*

We now introduce notation that simplifies our analysis of circuits on the gate level. Given an input vector I, we consider strategies for constructing circuits for I. In our constructions, it is convenient to think of the circuits as being constructed in the following way: The first gate, g_1, has all three of its inputs from $I_0 = I$, we now update I_0 to I_1, which has the three inputs to g_1 removed, but s_1, the output sum bit of g_1, is added. We then construct g_2 using three inputs from I_1, and update I_1 to get I_2, and so on.

2.4 Trade-Off Between Delay and Good Carry Vectors

We now examine the problem of designing the column circuits for the TDM PPRT. Suppose we aim to produce the two bits from each column within delay d. Then, the circuit for each column must, for its given input vector I, produce the two sum bits in delay $d' \le d$ along with a "good" carry vector. An optimal carry vector in this case must be undominated in the set of carry vectors for circuits $C(I)$ which produce their sum bits in time $\le d'$. Such a carry vector may not be undominated in the set of all carry vectors generated by circuits for I.

There is often a trade-off between making the carry vector good in the sense that it is undominated and minimizing the delay of the output bits in a circuit. For example, in Fig. 3, circuit A generates a carry vector that strictly dominates the carry vector of circuit B but does not achieve as good delay for the two sum bits. The design of the PPRT thus involves a judicious choice of the delay of the sum bits produced in each of the columns coupled with a strategy for finding a circuit that achieves that delay and produces an undominated carry vector, i.e., a PPRT with optimal delay will have column circuits, each of which produces an undominated output vector consisting of the carry vector concatenated with the larger of the two sum outputs.

2.5 Three-Greedy Approach

In [4], a heuristic for TDM design is proposed that we will call the *three-greedy approach*. The three-greedy approach is as follows: Take for the inputs of each gate g_i the three smallest values in I_{i-1}. For example, the adder circuit shown in Fig. 3a is three-greedy. Gate $g_1 = (0, 0, 0)$ takes as its inputs the three smallest values in $I_0 = I = (0, 0, 0, 0, 1, 1, 1, 1, 2, 4)$, and generates a sum bit at time 2. Thus, $I_1 = (0, 1, 1, 1, 1, 2, 2, 4)$, and we build gate $g_2 = (0, 1, 1)$. Similarly, $I_2 = (1, 1, 2, 2, 3, 4)$, $g_3 = (1, 1, 2)$, $I_3 = (2, 3, 3, 4)$, $g_4 = (2, 3, 3)$, and $V_s = (4, 5)$. Also, $V_c = (1, 2, 3, 4)$.

Note that the three-greedy strategy produces undominated circuits, since the three-greedy approach produces the lexicographically smallest carry vector, but it may be possible to produce circuits with better sum delays and/or incomparable carry vectors using other strategies.

3 Optimal Column-Addition Circuits

In this section, we consider properties of circuits that sum a vector of input bits and produce undominated output vectors. As mentioned earlier, the optimal PPRT uses undominated circuits to sum the bits in each column. We show that we need only consider a restricted class of addition circuits that have certain nice properties.

We now discuss the *two-greedy strategy*, which is defined as follows: Always construct gate g_i using the two smallest values in I_{i-1} plus a third value in I_{i-1}. The key fact we prove is that, for any column circuit $C(I)$, there exists a two-greedy column circuit $C'(I)$ that dominates $C(I)$. Thus, in searching for optimal combinations of column circuits, we can restrict our attention to two-greedy circuits. (In Fig. 3 circuit A is three-greedy and both circuits are two-greedy.)

3.1 The Two-Greedy Strategy

In this section, we prove several properties of two-greedy circuits. We begin with a few definitions.

DEFINITION. *A* **lexicographic ordering** *of the gates of a circuit $C(I)$ (or, loosely, a lexicographical ordering of $C(I)$) is one in which, for each pair of gates $g_i = (a_i, b_i, d_i)$ and $g_j = (a_j, b_j, d_j)$, g_i precedes g_j in the ordering whenever:*

- $a_i < a_j$;
- $a_i = a_j$ *and* $b_i < b_j$; *or*
- $a_i = a_j$, $b_i = b_j$, and $d_i < d_j$.

If $a_i = a_j$, $b_i = b_j$, and $d_i = d_j$, then the gates can appear in either order. We assume that all the circuits $C(I)$ which we discuss have their gates numbered in lexicographic order. While this may seem to conflict with the construction numbering of gates that we used in Subsection 2.3, we shall show that every circuit is dominated by a circuit (possibly itself) for which the lexicographic order is also a construction order.

The following definitions give some of the ways that gates can be related to each other:

DEFINITION. *Gate g_j is an* **immediate descendant** *of gate g_i if s_i is an input to g_j. Similarly, g_j is a* **descendant** *of g_i if g_j is an immediate descendant of g_i or of a gate g_k that is a descendant of g_i. Two gates are said to be* **independent** *if neither gate is a descendent of the other.*

We begin by proving that, when constructing a gate g_i, we can always use three values in I_{i-1}, rather than using an input that is the sum bit from a higher numbered gate. A circuit is *topologically ordered* iff, when the gates are numbered in lexicographic order, no gate g_i has an input that is the sum bit from g_j where $j > i$.

LEMMA 3.1. *For any circuit $C(I)$ that is not topologically ordered, there is a topologically ordered circuit $C'(I)$ such that $C'(I) \le C(I)$.*

PROOF. If $C(I)$ is not topologically ordered, then there exists a gate g_i with input s_j, where $i < j$. We now show that we can restructure the circuit $C(I)$ to make it topologically ordered without degrading the outputs of the circuit. We start by showing how to remove the feedback from g_j to g_i.

By definition, $a_i \le a_j < s_j$. Thus, $s_j = b_i$ or $s_j = d_i$.

Case 1. $s_j = d_i$.

Case 1a. $b_i \le b_j$. (See Fig. 4.)

$$
\begin{aligned}
g_j &= (a_j, b_j, d_j) & c_j &= d_j + y_3 \\
& & s_j &= \max(b_j + x_2, d_j + x_3) \\
g_i &= (a_i, b_i, s_j) & c_i &= s_j + y_3 \\
& & &= \max(b_j + x_2 + y_3, d_j + x_3 + y_3) \\
& & s_i &= \max(b_i + x_2, s_j + x_3) \\
& & &= \max(b_i + x_2, b_j + x_2 + x_3, d_j + 2x_3) \\
& & &= \max(b_j + x_2 + x_3, d_j + 2x_3).
\end{aligned}
$$

By rearranging inputs we can get:

$$
\begin{aligned}
g_i' &= (a_i, b_i, d_j) & c_i' &= d_j + y_3 \\
& & s_i' &= \max(b_i + x_2, d_j + x_3) \\
g_j' &= (a_j, b_j, s_i') & c_j' &= s_i' + y_3 \\
& & &= \max(b_i + x_2 + y_3, d_j + x_3 + y_3) \\
& & s_j' &= \max(b_j + x_2, s_i' + x_3) \\
& & &= \max(b_j + x_2, b_i + x_2 + x_3, d_j + 2x_3).
\end{aligned}
$$

Thus, by Observation 2.1, g_i' dominates g_j (since $b_i \le b_j$). Further, $c_j' \le c_i$ and $s_j' \le s_i$ (because $b_i \le b_j$), so g_j' dominates g_i. Therefore, the sum and carry values from g_i' and g_j' are at least as good as those for g_j and g_i. Thus, replacing g_j and g_i by g_i' and g_j' and using s_j' to replace s_i results in a circuit $\hat{C}(I)$ that dominates $C(I)$ (see Fig. 4).

Case 1b. $b_i > b_j$ ($\Rightarrow a_i < a_j$). By rearranging inputs, we can get $g_i' = (a_i, b_j, d_j)$, and $g_j' = (a_j, b_i, s_i')$. Now g_i' has the same sum and carry as g_j, thus, $s_i' = s_j$ ($\ge b_i$) and g_j' has the same sum and carry as g_i. Thus, replacing gates as in Case 1a yields a circuit $\hat{C}(I)$ with the same sum and carry vectors as $C(I)$.

Case 2. $s_j = b_i$ ($\le d_i$). In this case, we rearrange inputs to get $g_i' = (a_i, b_j, d_j)$, and $g_j' = (a_j, s_i', d_i)$. As above, g_i' has the same sum and carry as g_j, thus, $s_i' = s_j$ and g_j' has the same sum and carry as g_i. Thus, as in Case 1b, the new circuit $\hat{C}(I)$ has the same sum and carry vectors as $C(I)$.

Thus, in all cases, we have removed the feedback from g_j to g_i without degrading the circuit. We now show that repeated applications of these transformations remove all feedback. Consider the vector consisting of the inputs to the k lexicographically ordered gates of $C(I)$: (a_1, b_1, d_1, a_2, b_2, d_2, …, a_k, b_k, d_k). Clearly, the vector for the new circuit $\hat{C}(I)$ resulting from the transformation is lexicographically smaller than the vector before the transformation (since the inputs to g_i' < the inputs to g_i and g_1, …, g_{i-1} remain the same). Also, the vector is bounded from below by the vector for the three-greedy circuit. Therefore, the number of applications of the transformations is bounded, and repeated applications of those transformations to the lexicographically smallest (i, j) pair violating the topological ordering property will convert $C(I)$ to the desired topologically ordered circuit $C'(I) \le C(I)$, proving the lemma. □

LEMMA 3.2. *For any topologically ordered circuit C(I) that is not two-greedy, there is a two-greedy circuit C′(I) such that C′(I) ≤ C(I).*

PROOF. The proof of the lemma is similar to that of the previous lemma. We demonstrate swaps on a topologically ordered circuit $C(I)$ that is not two-greedy to convert it to a circuit $C'(I)$ that is two-greedy such that $C'(I) \le C(I)$. (Note that, by definition, all two-greedy circuits are topologically ordered.) The details are omitted. □

The prior two lemmas show that, given any circuit $C(I)$, we can convert it to a circuit $C'(I) \le C(I)$ such that, if the gates of $C'(I)$ are numbered lexicographically, each gate g_j in $C'(I)$ has as inputs the two smallest values in I_{j-1} plus a third input from I_{j-1}, i.e., $C'(I)$ can be constructed for the input vector $I = I_0$ using the two-greedy strategy.

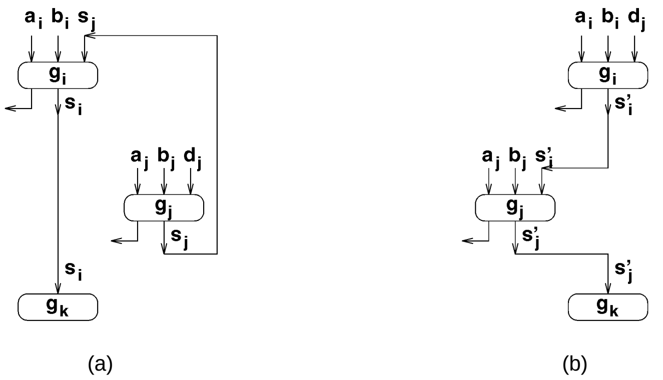


Fig. 4. Subcircuit (a) is not topologically ordered. Subcircuit (b) reflects the modifications made for Case 1a in the proof of Lemma 3.1. Note that both subcircuits take the same input, but that the output from subcircuit (b) dominates the output from subcircuit (a).

OBSERVATION 3.3. *Let C be a circuit constructible in lexicographic order using a two-greedy approach. If g_i appears before g_j in the lexicographic ordering of C, then g_i is created before g_j in a two-greedy lexicographic construction of C and, as a result, $a_i \le b_i \le a_j \le b_j$.*

3.2 Other Constraints on Undominated Circuits

We have already shown that any adder circuit that is not two-greedy is dominated by a two-greedy circuit, so that we need only consider two-greedy circuits when searching for all undominated circuits for a column. We now give additional constraints that we can use to further restrict the search.

In Subsection 2.4, we presented evidence that there may be a number of undominated circuits for a given input. As a reminder, consider the input vector I = (0, 0, 0, 0, 1, 1, 1, 1, 2, 4) as in Fig. 3. By Lemma 3.2 and Observation 3.3, we know that any undominated output can be achieved by a circuit that has 0, 0 as the first two inputs to g_1. The three-greedy strategy uses a third 0 as the final input, resulting in $c_1 = 1$ and $s_1 = 2$ (as in Fig. 3a). This minimizes c_1 over all choices of d_1. Alternatively, choosing $d_1 = 1$ also results in $s_1 = 2$, yet "consumes" an input of later delay, thus possibly improving the sum and/or carry of later gates (as in Fig. 3b). This example helps show why the two-greedy approach is attractive, and suggests that, if d_i is not chosen by the three-greedy strategy, then d_i should be close to the three-greedy value and/or be subject to other constraints.

The following lemmas formalize the above intuition by providing rules that eliminate from consideration circuits that are guaranteed to be dominated by other circuits that conform to the rules. These rules reduce the number of circuits that need to be constructed (in addition to the three-greedy circuit) for an input vector I when attempting to find all undominated circuits $C(I)$. In particular, Corollary 3.8 ensures that we can restrict our search to certain "regular" circuits for I. The proofs for the following use similar techniques to those used in the proofs for Lemmas 3.1 and 3.2 and are omitted.

LEMMA 3.4. *Let $C(I)$ be a two-greedy circuit. Then, there is a two-greedy circuit $C'(I) \le C(I)$ in which, for each pair of gates g_i and g_j, if $i < j$ and $d_i \le b_j$, then $d_i \le a_j$.*

COROLLARY 3.5. *Let $C(I)$ be a two-greedy circuit. Then, there is a two-greedy circuit $C'(I) \le C(I)$ in which, for each pair of gates g_i and g_j, if $i < j$ and $d_i = b_j$, then $a_j = b_j = d_i$.*

PROOF. Immediate from Lemma 3.4. □

LEMMA 3.6. *Let $C(I)$ be a two-greedy circuit. Then, there is a two-greedy circuit $C'(I) \le C(I)$ in which, for each pair of gates g_i and g_j, if $i < j$, then $d_i \le d_j$.*

LEMMA 3.7. *Let $C(I)$ be a two-greedy circuit, and let $x_2 \le 2x_3$. Then, there is a two-greedy circuit $C'(I) \le C(I)$ such that, for each pair of gates g_i and g_j, if $i < j$ and $b_j < d_i \le d_j$, then $d_j < d_i + x_2 - x_3$.*

Note that Lemma 3.7 applies to the standard problem as described earlier, i.e., $x_2 = 2$, $x_3 = 1$, $y_3 = 1$. When $x_2 \le 2x_3$, we define the following class of circuits:

DEFINITION. *A two-greedy circuit C for a vector I that has no pairs of gates of the form excluded by Lemma 3.1, Lemma 3.2, Lemma 3.4, Lemma 3.6, and Lemma 3.7 is said to be in* **regular form**, *or, loosely, simply said to be* **regular**.

COROLLARY 3.8. *If $x_2 \le 2x_3$, then, for any circuit $C(I)$, there is a regular circuit $C'(I) \le C(I)$.*

PROOF. If $C(I)$ is not regular, then we alternately apply the transformations from the proofs of Lemmas 3.1, 3.2, 3.4, 3.6, and 3.7 until we derive a regular circuit $C'(I)$. We know that a regular circuit $C'(I)$ must eventually result from this process by the same argument as was first used in the proof of Lemma 3.1. □

The above lemmas can be used to severely limit an exhaustive search for an optimal solution to the standard problem, since they allow us to restrict the search to regular circuits for each input I. In the next two sections, we describe a lower bound that can be used in conjunction with the above lemmas in a branch-and-bound search.

4 OPTIMAL DELAY CIRCUITS

In this section, we consider the problem of finding, for a given vector I, the minimum delay circuit $C(I)$ that outputs k sum bits. Thus, for input vector I and integer $k \ge 1$, we want a circuit of full adders that reduces I to k output bits of weight 1 and a carry vector. *In this section, we are not concerned with the delays of the carry vector generated by the circuit, and we restrict our attention to circuits consisting entirely of full adders.*

To simplify notation, in this section, we will denote the input vector by $V = (v_1, v_2, \ldots, v_n)$. We assume the standard problem previously described, where $x_2 = 2$ and $x_3 = 1$. We define a canonical circuit for each value of delay that can sum the maximum number of input bits available at time 0 within the given delay. We then show some properties of canonical circuits and develop measures for evaluating a given vector of input bits I. We show how to use these measures to efficiently determine the minimum delay in which a circuit made up of full adders can reduce the input bits given by I to a specified number of bits ($k \ge 1$).

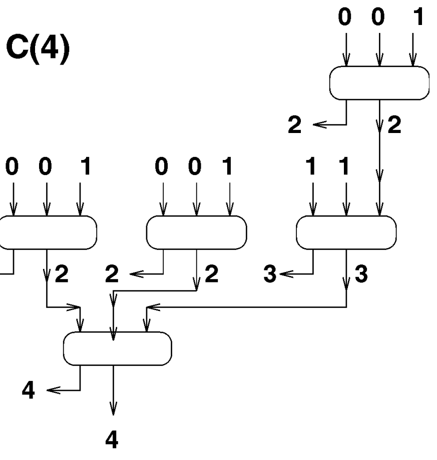


Fig. 5. The canonical circuit for delay four in which input bits of delay one replace input bits of delay zero in all cases where it does not affect the delay for the final sum bit.

4.1 Canonical Circuits for a Given Delay

We define $S(t)$ as the maximum number of bits available at time zero that can be added with full adders to produce a single sum by time t. We can write a simple recurrence for $S(t)$ as follows:

$$S(0) = S(1) = 1$$
$$S(t) = S(t-1) + 2S(t-2) \quad t \ge 2 \tag{4.1}$$

The recurrence follows from the observation that the best way to accommodate the most inputs completing in delay t is to have the last adder with output t from this circuit be $(t-2, t-2, t-1)$. Each of the three inputs to the last adder in this circuit is the output of a circuit of delay $t-2$ or $t-1$ that sums the maximum possible number of inputs. (The recurrence solves to $S(t) = \frac{2^{t+1}+(-1)^t}{3}$ for $t \ge 0$.) This observation can also be used to easily infer that the circuit that sums $S(t)$ input bits all available at time zero with delay t is unique. This unique circuit is termed the *Canonical circuit for delay t* and is denoted $C(t)$.

We denote by $S_i(t)$, $1 \le i$, the maximum number of subcircuits of $C(t)$ that can be replaced by input bits available at time i. For $i = 1$, this is the number of (0, 0, 0) adders (each of which can have its third input bit replaced by a 1) plus two times the number of (0, 0, 2) adders. For $i > 1$, $S_i(t)$ is the number of times $C(i)$ occurs as a subcircuit of $C(t)$. An illustration of $C(4)$, in which input bits of delay one replace input bits of delay zero in all $S_1(4) = 5$ situations, where this can be done without affecting the time of the final sum bit, is given in Fig. 5. Similarly, we can see from Fig. 5 that $S_2(4) = 3$, the number of (0, 0, 1) gates in $C(4)$.

LEMMA 4.1. *If we define $S(t) = 0$ for all $t < 0$, then $S_i(t) = S(t-i)$, i.e., $S_i(t) = 0$ for $t < i$ and $S(t-i) = \frac{2^{t-i+1}+(-1)^{t-i}}{3}$ for $t \ge i$.*

PROOF. By construction. Proofs of the lemmas and corollaries in this section will be omitted due to space limitations. □

Note that, after substituting $S_1(t)$ ones for zeroes in $C(t)$, all adders in the resulting circuit have inputs of the form $(k, k, k+1)$ and generate their sum output at time $k+2$.

OBSERVATION 4.2. *If our goal is to produce k sum bits within delay t, then, at most, $k \cdot S(t)$ bits available at time zero can be summed (by using k copies of $C(t)$) and, at most, $k \cdot S_1(t) = k \cdot S(t-1)$ of those time zero bits can be replaced by bits available at time 1. Similarly, if our goal is to produce k sum bits within delay t using input bits of delay i, then, at most, $k \cdot S(t-i) = k \cdot S_i(t)$ bits can be summed.*

We define the *weight* of a vector of input delays $V = (v_1, v_2, \ldots, v_n)$ as $W(V) = S(v_1) + S(v_2) + \ldots + S(v_n)$. Thus, any circuit that can sum inputs with delay V to produce a final sum bit with delay t could, by replacing each v_i with $C(v_i)$, sum $W(V)$ inputs each of delay zero in time t. Thus, a lower bound on the time to sum inputs of delay V is given by the smallest t where $S(t) \geq W(V)$. By a similar definition and interpretation, we let the *i-weight* of V be $W_i(V) = S_i(v_1) + S_i(v_2) + \ldots + S_i(v_n)$.

LEMMA 4.3. *If, for any $v \in V$, $W_v(V) > k \cdot S_v(t)$, then inputs with delay V cannot be summed to k bits with delay $d \leq t$.*

This lemma is useful for proving lower bounds on multiplier circuits, since it bounds the delay for a column given a carry vector from the previous column. We now show that the lower bound given by Lemma 4.3 is achievable.

LEMMA 4.4. *If, for all values $v \in V$, $W_v(V) \leq k \cdot S_v(t)$, then inputs with delay V (possibly with some additional bits available at time zero) can be summed to k bits with delay $d \leq t$.*

We now show that, with appropriate data representation, we can compute $W_v(V)$ for all values $v \in V$ in time linear in the number of values in V.

LEMMA 4.5. *Let V be represented as a list of ordered pairs (v_i, k_i), $1 \leq i \leq m$ in decreasing order of v_i, where k_i = (the number of elements of V with value v_i). If, in constant time, we can multiply by 2^j, $1 \leq j \leq v_1$ (e.g., by shifting bits to the left), then we can calculate $W_{v_i}(V)$ for all i, $1 \leq i \leq m$, in $O(m)$ time.*

Since we can derive the above representation of V from an unsorted vector in $O(n \log m)$ time, we get the following theorem.

THEOREM 4.6. *Given a vector V of n input delays with m distinct values, we can, in $O(n \log m)$ time, find the smallest time t for which we can generate k output bits each of delay $\leq t$.*

Finally, we show that, if $W(V) \leq S(t)$, then inputs with delay V can be summed to a single bit with delay d, $t \leq d \leq (t+1)$.

THEOREM 4.7. *If $W(V) \leq S(t)$, then $W_{v_i}(V) \leq S_{v_i}(t+1)$ for all v_i, $1 \leq i \leq n$.*

COROLLARY 4.8. *If $W(V) \leq S(t)$, then $W_j(V) \leq S_j(t+1)$ for all j, $1 \leq j \leq t$.*

COROLLARY 4.9. *If $W(V) \leq S(t)$, then inputs with delay V (possibly with some additional bits available at time zero) can be summed to k bits with delay $d \leq (t+1)$.*

5 A LOWER BOUND USING A RELAXED PROBLEM

Recall that, under the standard problem, we model the full adder such that the sum is generated at time $s = \max(b+2, d+1)$ and the carry at time $c = d + 1$. Now, consider the design of the multiplier circuit with a full adder that, for inputs $a \leq b \leq d$, produces a sum at time $s = d + 1$ and a carry at time $c = d + 1$ as well. Note that we do not claim that such a full adder can be built, but simply use this as a relaxed version of the original problem. Using techniques as in Lemma 3.2, we can show that, for this relaxed problem, the three-greedy approach described in Section 2.5 (i.e., repeatedly putting the three earliest bits into a full adder) is globally optimal for both sum and carry.[1] In fact, this strategy is identical to that used in [4] in the design of a 16-bit multiplier circuit that completes within a delay of eight units. Note that the three-greedy strategy can also be considered *carry-greedy*, since this choice of inputs reflects a local minimization of the carry vector generated.

The optimality of the three-greedy strategy for the relaxed problem motivates the following lower bound for the original problem. Since the relaxed problem is a less constrained problem than the original, the optimal delay of a multiplier for the relaxed problem is a lower bound on the delay of the multiplier for the original problem. Furthermore, if we determine, using other techniques, a lower bound on the carry vector being input into a certain column in the multiplier, we can apply the three-greedy strategy to subsequent columns under the relaxed and original sum output models to derive lower and upper bounds for the original problem. This strategy can also be combined with the results of Lemma 4.4 and Corollary 4.9 to provide a number of different approaches for computing partial lower bounds to the original problem.

6 TDM PPRT DESIGN: PROBLEM DEFINITION AND AN APPLICATION

We now address the problem of finding optimal TDM PPRT circuits for multipliers of two n-bit numbers subject to the standard problem conditions described earlier (e.g., $s = \max(b+2, d+1)$, $c = d+1$). As stated previously, the TDM strategy is to generate two sum bits for each column by using full adders to add the partial product bits for that column plus the carry-in bits from the previous column. (As noted earlier, one half adder will be used on the two earliest inputs if the number of inputs is odd.) First, we examine the problem at its most basic level, that of minimizing the delay of the latest sum output bit generated in any column. We will refer to this problem as the Mini-max problem. Later, we will look at optimizing the vector of outputs from the PPRT to minimize the total delay from the PPRT and the final adder.

We have developed a program that solves the Mini-max problem. For inputs n and m, the program searches for a TDM PPRT circuit for an n-by-n multiplication in which the maximum delay for the latest sum bit is $\leq m$. If it finds such circuits, then it prints them out; otherwise, it states that

1. If the number of inputs is odd, then we assume, as before, that the two earliest are fed into a half adder.

there is no such circuit. The program uses the results of the previous sections to severely prune the search space for the optimal PPRT. It does this as follows: Recall from Section 2.3 that an optimal PPRT circuit can always be constructed using only undominated column circuits. But, by Corollary 3.8, we have that, given the standard problem, every column circuit for an input vector I is dominated by a regular column circuit for I. Thus, only the regular column circuits need be considered for each column when searching for optimal PPRT circuits. The program does this as follows:

The program processes the columns of the PPRT in order of increasing significance of bit value (i.e., 2^0, 2^1, 2^2, …). It first generates the trivial circuit for columns 0, 1, and 2, and the (unique) carry vector from column 2 for input to column 3. Then, for each i, $3 \le i \le (2n - 2)$, it computes each of the regular circuits for column i. It does this by first generating the possible input vectors for column i (consisting of the partial products of delay 0 for column i and each of the carry vectors from column $(i - 1)$) and, then, generating all regular circuits for each input vector. The program eliminates all circuits for which the delay of the latest sum bit is $> m$. The carry vectors from the remaining circuits are then compared to each other, and one circuit is retained for each undominated carry vector. If the sum delay for all circuits is $> m$, then the program reports that the delay bound m cannot be met by a TDM PPRT for size n-by-n multiplication. Otherwise, it processes the next column. It continues processing columns until either it learns m cannot be met or all columns have been processed, in which case it reports one of the satisfying TDM PPRT circuits.

The approach above severely restricts the number of circuits generated for each column compared to an unrestricted approach, or even one that limited the search to all nonisomorphic circuits. However, if we let $U(i, n)$ be the number of undominated circuits for column i in a TDM PPRT for an $n \times n$-bit multiplier, then $U(i, n)$ grows rapidly with i for $i \le n$. As a result, the space requirements (to store the undominated carry vectors from column i) and time (to compare each carry vector from a column against the others to ensure that only undominated vectors are kept) also increase tremendously with n. This growth has restricted the sizes of n for which we have been able to solve the standard problem exactly.

6.1 PPRT Delay Profile Problem

We can refine our notion of an optimal TDM PPRT by incorporating the times at which the latest bit for each column is generated, rather than just the latest sum bit over all columns. We call the vector of these times for a TDM PPRT circuit C its delay profile, and denote it by $P_C(n) = (p_0, p_1, \ldots, p_{2n-2})$. For any given multiplier size n with minimum maximum TDM PPRT delay m, there can be a large number of distinct profiles $P(n)$ with maximum delay m, each generated by one or more TDM PPRT circuits. Since these output profiles determine when the bits will be made available to the final adder, we can also consider the problem of finding the TDM PPRT circuits with the “best” profiles. Clearly, we can restrict ourselves to undominated profiles, since, if $P_A < P_B$, then any adder applied to inputs available according to profile P_A must yield its latest output bit no later than the same adder applied to inputs available according to profile P_B. Additional considerations among undominated profiles also exist, and are briefly discussed later in this subsection. We call the problem of generating TDM PPRT circuits with (all) undominated delay profiles the Profile problem. We let $\mathcal{P}(n)$ be the set of undominated profile vectors $P_C(n)$ that can be generated by a TDM PPRT circuit for an n-by-n multiplier.

We now address how the undominated profiles and their corresponding circuits are generated. Later, we will discuss additional criteria that can be used for evaluating incomparable profile vectors $P_i(n)$ and $P_j(n)$. First, we introduce some notation we will use to simplify discussion of the program and how it works.

We let $C_i(n)$ be the set of TDM PPRT circuits for columns 0 through i of an n-by-n multiplication. As stated earlier, we need only consider PPRT circuits that are made up of undominated column circuits. Recall that a column circuit is undominated if its output vector is undominated by any other column circuit with the same input, and that the output vector of a column circuit consists of the delay of the latest sum bit it produces and its carry vector. We now extend this definition for circuits in $C_i(n)$. Each circuit in $C_i(n)$ has as output two sum bits for each column 0, 1, …, i and a carry vector from column i for input to column $(i + 1)$. For purposes of the Profile problem, the relevant outputs of a circuit $C \in C_i(n)$ are the partial profile vector P_C of delays of the latest sum bits generated in each of the columns (0 through i) and the carry vector V_C from column i. We let $R_C = (P_C, V_C)$ be the result of circuit C, and define $\mathcal{R}_i(n)$ as the class of undominated results from circuits in $C_i(n)$. It is possible that a result $R \in \mathcal{R}_i(n)$ could be generated by more than one circuit from $C_i(n)$. Therefore, we define a set $\mathcal{U}_i(n)$ containing one such circuit $C \in C_i(n)$ for each undominated result $R \in \mathcal{R}_i(n)$.

The general approach of the Profile program is similar to that of the Mini-max program. The program processes the columns of the PPRT in order of increasing significance of bit value (i.e., 2^0, 2^1, 2^2, …). It first generates the trivial circuit for columns 0, 1, and 2 (the unique circuit in $C_2(n) = \mathcal{U}_2(n)$). Then, for each i, $3 \le i \le (2n - 2)$, it generates $\mathcal{R}_i(n)$ and $\mathcal{U}_i(n)$ from $\mathcal{R}_{i-1}(n)$ and $\mathcal{U}_{i-1}(n)$. $\mathcal{R}_i(n)$ and $C_i(n)$ are initialized as empty sets. We want to ensure that each regular column circuit for column i is generated only once. To do this, the program maintains a list of the unique carry vectors in $\mathcal{R}_{i-1}$, and separately records for each circuit $C \in \mathcal{U}_{i-1}(n)$ pointers to the corresponding partial profile and carry-out vectors. The program thus can process the carry vectors one at a time.

At a high level, it does this by processing each carry vector V_s as follows: First, it creates the corresponding input vector I for column i (consisting of the delay 0 partial product bits for the column and the carry bit delays). Then, it generates all regular column circuits for I, keeping only the undominated ones. Each such column circuit is then combined with each circuit $C_{i-1}(n)$ which had carry vector V_C to give new circuits $C_i(n)$. The results from these circuits are compared to the previously generated (undominated) results in $\mathcal{R}_i(n)$. If one of the new results is dominated by a result in $\mathcal{R}_i(n)$, then it is discarded; if one of the new results strictly dominates a result $R \in \mathcal{R}_i(n)$, then R is discarded, as is the corresponding circuit from $C_i(n)$. The (undominated) new output vectors that are not discarded are added to $\mathcal{R}_i(n)$, and their corresponding circuits to $C_i(n)$.

Once $\mathcal{R}_{2n-2} = \hat{\mathcal{P}}(n)$ and $C_{2n-2}(n)$ have been completed, the undominated profiles and their corresponding circuits are output. The number of such profiles increases very quickly with the size of the multiplication (5 for 8-by-8 bit multiplication, 57 for 16-by-16 bit, 447 for 24-by-24 bit, 5,634 for 30-by-30 bit). Of these, we want to select one that will give us the best parallel multiplier.

For the purposes of parallel multiplier design, we are interested in the total delay for the PPRT and the final adder. The optimal profile vector(s) will be the one(s) for which a (possibly custom) final adder will generate the product with minimum delay. We do not yet have a full understanding of the optimal delay for a final adder given a profile. Thus, we use heuristics to identify "good" profiles. Since delays propagate (with the carry values) from least significant to most significant bit, we will use the following "latest-earliest" heuristic for ordering the profiles in $\mathcal{P}(n)$. Profile P_i comes before P_j if the largest value that does not appear in exactly the same positions in P_i and P_j:

- Does not appear in *Pi*,
- First appears later in P_i, or
- First appears in the same position of both P_i and P_j, but last appears later in P_j.

As an example, profile (1, 2, 3, 3.5, 3, 2) would come before (1, 2, 2.5, 4, 3, 1) because delay 4 does not occur in the first profile and occurs (in column 4) in the second profile. Similarly, (1, 2, 3, 4, 3, 2) would come before (1, 2, 3.5, 4, 2, 1) because the maximum delay 4 occurs in the same column of both profiles, the next smaller delay of 3.5 occurs in the second profile but not the first. Also, (1, 2, 3, 4, 2, 2) would come before (1, 2, 3, 4, 3, 1) because the maximum delay 4 occurs in the same column of both profiles, the next smaller delay of 3 first occurs in the same column of both profiles, but the last column of delay 3 occurs earlier in the first profile than in the second.

As a partial check on the quality of the profile ordering, we also generate a super-optimal profile consisting of the smallest delays for each column in any solution. These profiles conform closely to the latest-earliest profiles in all columns except those with the maximum delay value (whose position we have selected by the latest-earliest rule).

Given n, our profile program searches over TDM circuits for the PPRT and gives the undominated profile vectors $P \in \hat{\mathcal{P}}(n)$ sorted in the order given by the latest-earliest heuristic (as well as a circuit that will generate each vector P). We have run the profile program successfully on problem sizes up to 31-by-31 bits, and have used heuristics to get solutions we believe to be optimal up to size 48-by-48 bits. However, further improvements will be needed to deal with time and memory constraints in order to solve larger problems. The Mini-max program runs faster and uses less memory, but it too requires considerable time for problem sizes greater than 47-by-47 bits. The use of parallel programming may help with both the Mini-max and Profile problems by distributing the processing of input vectors for a column over many processors, with dominated circuits being pruned both at the "column" processors and as their results are accumulated.

TABLE 2
STANDARD PROBLEM RESULTS: MINI-MAX DELAYS FOR THREE-GREEDY VS. OPTIMAL TDM PPRT CIRCUITS

Multiplication Size in Bits	Maximum XOR Delay	
	3-Greedy	Optimal
7-8	5	5
9-10	6	6
11-12	7	7
13	7.5	7
14-16	8	8
17	9	8
18-20	9	9
21	9.5	9
22	10	9
23-26	10	10
27-28	11	10
29-35	11	11
36	12	11
37-44	12	12
45-47	13	12
48-57	13	13
58	13.5	13
59	14	13
60-76	14	??
77-98	15	??
99-128	16	??

Times in Normalized XOR Delays

6.2 Results

We used the programs to determine the optimal solutions to both the Mini-max and the Profile versions of the TDM PPRT design problem. We have shown, using the Mini-max program, that, for most sizes of circuits up to that for multiplication of two 57-bit numbers, the mini-max delay achieved by the three-greedy method is optimal. Further, it is within one XOR delay of the optimal for PPRT circuit designs in those cases where it is not optimal. These results are summarized in Table 2. The results for sizes up through 48 were found directly, using the Mini-max program. The optimal results in the table for sizes 49 through 57 follow from the fact that the optimal mini-max delay for those sizes is no larger than the three-greedy mini-max delay and is bounded from below by the mini-max delay for size 48. Finally, the optimal results for sizes 58 and 59 are also bounded from below by the mini-max delay for size 48, and were found by a version of the Mini-max program that limits its search to the five lexicographically smallest circuits for each carry-in vector processed. The best delay found by that version of the program for problem size 60 was 14 (same as three-greedy), but we believe that the optimal delay for that size problem is 13 as well.

The Profile program shows that the output profile of the three-greedy solution is strictly dominated by the profiles of other TDM PPRT circuits for all problem sizes ≥ 6. This is because a non-three-greedy circuit for column 4 yields a lower maximum value for that column than the three-greedy circuit, while generating a carry-out vector that has the same three-greedy result for column 5. Similar effects also manifest in higher columns for larger size problems.

The current version of the Profile program solves problems of size 16 and 24 in just over one second and just over 16 minutes, respectively, on a DEC 5000/240, and in under 0.7 seconds and 6.75 minutes, respectively, on a DEC Alpha. We have been unable to run the program to get all undominated profiles on problem sizes larger than 30 due to memory constraints, but are working on a version that uses less memory. (A previous version of the program used less memory but took over nine days to solve a problem of size 27). We have focused our main attention on finding the latest-earliest circuit for each problem size. This is easily accomplished in the obvious way when all undominated TDM PPRT circuits are generated. We have extended these results using a heuristic based on the observation that the profiles for problems of size $(n - 1)$ and n agree for the first $(n - 2)$ columns for all $n \leq 30$. We derive the (believed) latest-earliest solution for size $(n + 1)$ by pruning out all circuits for which the column delays of (any of) the first $(n - 1)$ columns exceed the corresponding delays of latest sum bits in the latest-earliest solution for the problem of size n and, then, taking the latest-earliest of the derived circuits. In this manner, we have been able to find good TDM PPRT circuits for n up to 48. Although we do not know that the derived solutions for sizes larger than 30 bits are the latest-earliest solutions, their profiles are significantly better than the three-greedy profiles. For example, the three-greedy profile for size 45 has one column of delay 13 (column 49) and 22 columns of delay 12 (columns 39-48 and 50-61). By contrast, the derived solution has 17 columns with the maximum delay of 12 (columns 41-57). We believe the derived TDM PPRT circuits are, in fact, the latest-earliest circuits, but do not have a proof that that is indeed the case (even when we limit our analysis to the standard model). Further examination has shown that, in addition to having the same profile, the latest-earliest circuits of size $(n - 1)$ and n agree exactly for at least the first $(n - 3)$ columns. We have also used this observation to further extend our results (by specifying actual column circuits, as opposed to just partial profiles).

Our preliminary work on final adder design strongly suggests that the optimal final adders for the derived latest-earliest solutions will in most cases complete more quickly than the optimal final adder for the three-greedy solution. Also, the final adder will be simpler for those cases where they generate the final sum with the same delay. In either case, it is clear that the latest-earliest solution adds a good alternative to the three-greedy solution for TDM PPRTs.

Fig. 6 contrasts the profiles for the three-greedy and latest-earliest TDM PPRT circuits for 24-by-24 bit multiplication, demonstrating both that latest-earliest improves on the delays of three-greedy for many columns (worse only for column 44), and has only four columns with the maximum equivalent XOR delay of 10, compared to eight columns for the three-greedy solution. Fig. 7 shows the latest-earliest profile of Fig. 6 compared to the super-optimal profile described above, showing that the latest-earliest profile does as well as *any* profile for almost all columns.

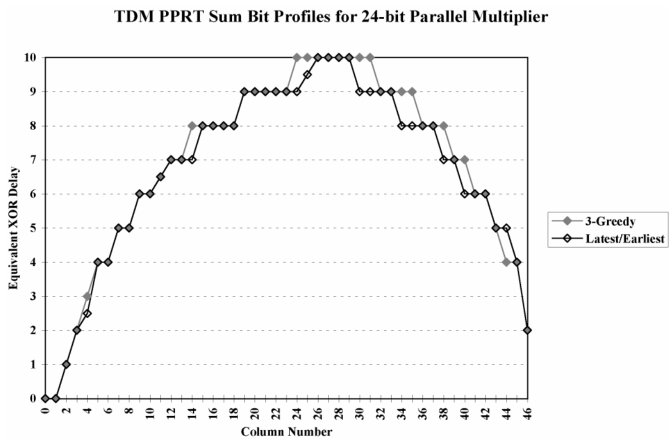


Fig. 6. Sum bit delay profiles for three-greedy and Latest-earliest TDM PPRT circuits for a 24 × 24 bit multiplier under the standard problem (sum = max(b + 2, d + 1), carry = d + 1).

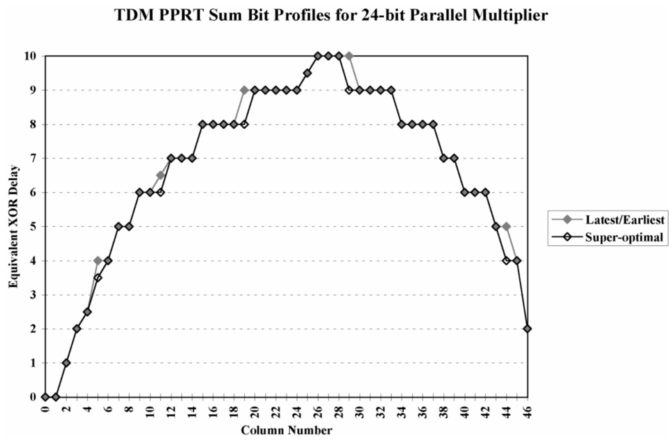


Fig. 7. Latest-earliest and Super-optimal TDM PPRT sum bit delay profiles for a 24 × 24 bit multiplier under the standard problem (sum = max(b + 2, d + 1), carry = d + 1).

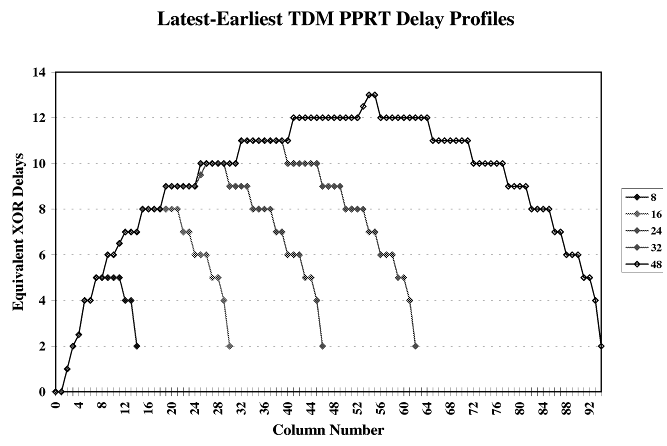


Fig. 8 The sum bit delay profiles for the latest-earliest and derived TDM PPRT circuits for standard multiplier sizes (and largest completed) given the standard problem.

Fig. 8 gives the latest-earliest and derived profiles for the standard multiplier sizes 8, 16, 24, and 32, as well as the derived profile for size 48 (the largest we have completed so far).

7 Conclusions

We have presented several techniques for analyzing the optimality of multiplier circuits designed using full adder cells. We develop the class of regular adder circuits for adding a vector of bits of varying delays. We typify the outputs of these circuits based on the vectors of the delays of the sum and carry bits that they generate, and use the notion of vector domination to describe optimal circuit outputs. We show that every optimal TDM adder output can be generated by a regular adder circuit, and describe how to use these results to build and find optimal TDM PPRTs, both in terms of minimizing the maximum overall delay (Mini-max problem) and in terms of finding undominated sum delay profiles (Profile problem). We have incorporated these results into programs that find optimal circuits for the standard problem that assumes full adder input-to-output delays based on equivalent XOR delays. We used these programs to solve the Mini-max delay problem up to size 57, and the optimal sum delay profile problem up to size 30 percent. We have extended our results to size 48 percent, using heuristics that seem to produce optimal or near-optimal solutions. Based on the program results, we find that the three-greedy heuristic works quite well for the mini-max problem, though it yields suboptimal maximum sum delay for a number of problem sizes (although it is always within one equivalent XOR delay of optimal for the standard problem). Also, the three-greedy heuristic results in suboptimal sum delay profiles for all problem sizes ≥ 6.

Many interesting issues remain open. We plan to do a simulation of a full design using the circuits produced by our program to validate that our abstract model of delay results in good performance in practice. We also plan to improve our search program to solve larger problems to optimality. Additionally, we will continue experimenting with a number of heuristics for finding solutions that are better than the three-greedy solution, yet possibly suboptimal, for larger problems. These heuristics all look at different ways of limiting the number of regular circuits generated in a column for each carry-in vector. One approach we have used with some success is to provide a maximum (partial) profile for columns 0 through i, for some $i < n$. This approach appears very promising because, for each size of n for which our Profile program has found all undominated profiles, the TDM PPRT circuit with the latest-earliest profile has had a profile that is identical to the latest-earliest TDM PPRT circuit of size $(n - 1)$ for the first $(n - 2)$ columns. Empirical investigation has shown that the delay of the latest sum bit from a given column i increases with n until n reaches a threshold value at $(i + 2)$ or $(i + 3)$, after which the maximum delay for column i remains the same. Additionally, the column circuits for column i are empirically the same for all latest-earliest TDM PPRTs where $n \geq i + 4$. For example, the latest-earliest TDM PPRT solution for size 16 uses the same column circuits for columns 0 through 12 as for all larger sizes that we have been able to solve. (Both of these empirical results hold up through size 48, the largest size we have solved so far using the profile pruning heuristic.) We have also taken advantage of this by expanding our approach to include specification of the circuits (and, hence, carry vectors) for some of the columns. Later, we will consider alternative approaches to more aggressive pruning, such as limiting the search to the k lexicographically smallest column circuits, or limiting the difference $(d_i - b_i)$ in the inputs to a gate.

Our analytical results apply to a number of interesting variations on the TDM PPRT design problem that bear additional investigation. In some of these cases, modification of our programs could yield additional worthwhile results. One of these is to investigate the impact of using more accurate delay estimates for the sum and carry, so that $s = \max(b + x_2, d + x_3)$ and $c = d + y_3$ where $x_3 \neq y_3$ and $x_2 \neq 2x_3$. The results relating to regular circuits all apply to the case where the sum bit is available at time $\max(b + x_2, d + x_3)$ and the carry bit is available at time $\max(b + y_2, d + y_3)$. A special case of this is where both the sum and carry are available at time $\max(b + x_2, d + x_3)$, as might be the case for a full adder based on multiplexers rather than XOR gates and NAND gates. Also, our analytical results apply equally to asymmetric multiplication, where the two numbers being multiplied are of differing bit sizes, and which is of interest in the design of special processing units. We plan to further investigate each of these situations.

Note that our results on the optimality of regular circuits apply to both the *sum* and *carry* vectors produced. Thus, a search over regular circuits is sufficient to find optimal PPRT circuits for feeding the final carry-propagate adder. An additional problem is to optimize the final carry-propagate adder by taking into account the arrival time profile of the input bits. We have examined this problem and presented optimal hybrid adders that use Ripple-Carry, Carry-Skip, Carry-Select, and Conditional Sum blocks to produce final adders that exploit the good profiles [5], [13], [14], [16]. These hybrid adders generate the sum with significant delay improvement. For example, the (provably optimal) hybrid adder for a 32-bit multiplier that uses Conditional Sum blocks shows over 20 percent improvement in the incremental delay compared to traditional Conditional Sum adders, translating into > 7.9 percent reduction in total multiplier delay [14], [16]. Similarly, we have applied these innovations to Multiply-Accumulators (MACs) [15], [16], giving optimal MACs with the same delay as the optimal multiplier for almost all sizes of factors.

A final topic is to consider optimizing over a broader class of design strategies that allow other uses of half adders and even new components.

Acknowledgments

Most of this work was done when Paul Stelling was at the Department of Computer Science, University of California at Davis, and was supported by U.S. National Science Foundation grants CCR-94-03651 and CCR-91-03937. The research of Charles Martel was supported by U.S. National Science Foundation Grant CCR-94-03651. Most of this work was done when R. Ravi was at the Department of Computer Science, University of California at Davis. His research was supported in part by U.S. National Science Foundation Grant CCR-91-03937 and by a DIMACS postdoctoral fellowship. Prof. Oklobdzija's research has been supported by an unrestricted grant by Hitachi America Ltd., grants by LSI Logic Corp., Cascade Design Automation, and California MICRO Grants 94-097, 95-114, 96-137. Prof. Oklobdzija acknowledges help from Mr. R. Ulicki of LSI Logic Corp., Dr. R. Oettel of Cascade Design Automation, and Dr. K. Sasaki of Hitachi Research.

REFERENCES

[1] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*. McGraw-Hill, 1990.

[2] L. Dadda, "Some Schemes for Parallel Multipliers," *Alta Frequenza*, vol. 34, pp. 349-356, 1965.

[3] K. Hwang, *Computer Arithmetic: Principles, Architecture and Design*. John Wiley and Sons, 1979.

[4] V.G. Oklobdzija, D. Villeger, and S.S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," *IEEE Trans. Computers*, vol. 45, no. 3, pp. 294-306, Mar. 1996.

[5] V.G. Oklobdzija and D. Villeger, "Optimization and Analysis of a Carry-Propagate Adder Under the Non-Uniform Signal Arrival Profile," in preparation.

[6] V.G. Oklobdzija and D. Villeger, "Improving Multiplier Design by Using Improved Column Compression Tree and Optimized Final Adder in CMOS Technology," *IEEE Trans. VLSI*, vol. 3, no. 2, June1995.

[7] M.S. Paterson, N. Pippenger, and U. Zwick, "Faster Circuits and Shorter Formulae for Multiple Addition, Multiplication and Symmetric Boolean Functions," *Proc. 31st Foundations of Computer Science*, pp. 642-650, 1990.

[8] M.S. Paterson, N. Pippenger, and U. Zwick, "Optimal Carry Save Networks," *Boolean Functional Complexity: Selected Papers from the LMS Symp., Durham 1990*. Cambridge Univ. Press, 1992.

[9] M.S. Paterson and U. Zwick, "Shallow Multiplication Circuits and Wise Financial Investments," *Proc. 24th Symp. Theory of Computing*, pp. 429-437, 1992.

[10] M.R. Santoro, "Design and Clocking of VLSI Multipliers," PhD dissertation, Technical Report no. CSL-TR-89-397, 1989.

[11] P. Song and G. De Michelli, "Circuit and Architecture Trade-Offs for High Speed Multiplication," *IEEE J. Solid State Circuits*, vol. 26, 1991.

[12] W.J. Stenzel, "A Compact High Speed Parallel Multiplication Scheme," *IEEE Trans. Computers*, vol. 26, pp. 948-957, 1977.

[13] P.F. Stelling and V.G. Oklobdzija, "Design Strategies for the Final Adder in a Parallel Multiplier," *Proc. 29th Asilomar Conf. Signals, Systems, and Computers*, pp. 591-595, 1995.

[14] P.F. Stelling and V.G. Oklobdzija, "Design Strategies for Optimal Hybrid Final Adders in a Parallel Multiplier," *J. VLSI Signal Processing*, vol. 14, no. 3, pp. 321-331, 1996.

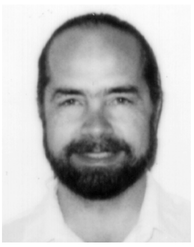
[15] P.F. Stelling and V.G. Oklobdzija, "Implementing Multiply-Accumulate Operation in Multiplication Time," *Proc. 13th Symp. Computer Arithmetic*, pp. 99-106, 1997.

[16] P.F. Stelling and V.G. Oklobdzija, "Optimal Designs for Multipliers and Multiply-Accumulators," *Proc. 15th IMACS World Congress 1997 on Scientific Computation, Modelling, and Applied Mathematics, vol. 4, Artificial Intelligence and Computer Science*, A. Sydow, ed., pp. 739-744. Berlin: Wissenschaft und Technik Verlag, 1997.

[17] E. Swartzlander, *Computer Arithmetic*, vols. 1 & 2. IEEE CS Press, 1990.

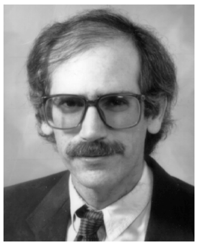
[18] C.S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Trans. Electronic Computers*, vol. 13, pp. 14-17, 1964.

[19] *1.0-Micron Array-Based Products Databook*. LSI Logic Corporation, 1991.



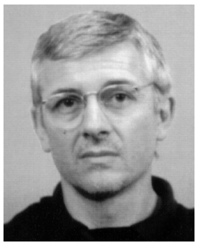
Paul F. Stelling received the BA degree in mathematics and economics from Claremont McKenna College in 1978, the MS degree in computer science from the University of Nebraska in 1982, and the PhD degree in computer science from the University of California at Davis in 1995.

After receiving his PhD, he remained at U.C. Davis as a postgraduate researcher and lecturer for a year, after which he joined the technical staff at The Aerospace Corporation in Los Angeles, California. His research interests include the design and analysis of algorithms, application of computing theory to such areas as circuit design and computational biology, distributed and parallel processing, and computer security. He is a member of the IEEE.



Charles U. Martel received the BS degree in computer science from the Massachusetts Institute of Technology in 1975 and the PhD degree in computer science from the University of California at Berkeley in 1980. Since 1980, he has been a member of the faculty of the Computer Science Department at the University of California at Davis, where he is currently a full professor. He has been chairman of the department since July 1, 1994.

Dr. Martel has worked on algorithms for parallel and distributed systems, scheduling, graph theory, and network algorithms. His current research interests include design and analysis of algorithms, parallel and distributed computing, and graph algorithms. As a four-time world bridge champion, he also has an interest in computer bridge playing programs.



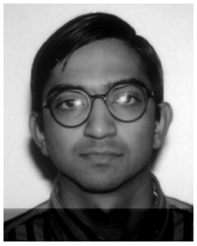
Vojin G. Oklobdzija obtained the Dipl Ing (MScEE) degree in electronics and telecommunications from the Electrical Engineering Department of the University of Belgrade, Yugoslavia, in 1971, where he stayed on the faculty until 1976 when he became a Fulbright Scholar at the University of California at Los Angeles. He obtained his MSc and PhD in computer science from UCLA in 1978 and 1982, respectively.

After receiving his PhD, he became a research staff member at the IBM T.J. Watson Research Center in New York, where he spent eight years making contributions to the development of RISC and super-scalar RISC architecture and processors. He left IBM in 1991 and is currently with Integration Berkeley California and the Electrical and Computer Engineering Department of the University of California at Davis.

As a visiting faculty member from IBM, Professor Oklobdzija taught courses in computer architecture, computer arithmetic, and computer design at the University of California at Berkeley from 1988 to 1990. His industrial experience includes positions at the Microelectronics Center of the Xerox Corporation and consulting positions at Sun Microsystems Laboratories, AT&T Bell Laboratories, Siemens Corp., Hitachi, and various others. Dr. Oklobdzija has published more than 80 papers in the areas of circuits and technology, computer arithmetic, and computer architecture. He has presented many invited talks in the U.S., Europe, Latin America, Australia, China, and Japan. In 1991, he was a Fulbright professor in Peru, teaching and helping in the creation of the Latin American University programs.

He holds one of the patents on the IBM RS/6000–"PowerPC" architecture and four U.S. and four European patents in the area of circuits and computer design. His current research interests are in VLSI and fast circuits, efficient implementations of algorithms, and computation.

Dr. Oklobdzija is a fellow of the IEEE and a member of the American Association of University Professors. He serves on the editorial boards of the *Journal of VLSI Signal Processing* and *IEEE Transactions on VLSI Systems*, and on the program committees of the International Solid-State Circuits Conference, the International Symposium on VLSI Technology, and the International Conference on Computer Design. He is currently serving as a general chair for the 13th International Symposium on Computer Arithmetic.



R. Ravi received a BTech in computer science and engineering from the Indian Institute of Technology, Madras, in 1989, and an MSc (1991) and PhD (1993) from Brown University. Following completion of his PhD, Dr. Ravi worked as a postdoctoral researcher in the Computer Science Department at the University of California at Davis for a year and as a postdoctoral researcher at the U.S. National Science Foundation Center for Discrete Mathematics and Theoretical Computer Science (DIMACS) in the Computer Science Department at Princeton University for a year.

Dr. Ravi has been an assistant professor of operations research at the Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, Pennsylvania, since Fall 1995. His primary research interests are in algorithms for combinatorial optimization.