GOMIL: Global Optimization of Multiplier by Integer Linear Programming

Weihua Xiao[1], Weikang Qian[1,2,3], Weiqiang Liu[4]
[1]University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, Shanghai, China
[2]MoE Key Laboratory of Artificial Intelligence, Shanghai Jiao Tong University, Shanghai, China
[3]State Key Laboratory of ASIC & System, Fudan University, Shanghai, China
[4]College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing, China
Emails: 019370910014@sjtu.edu.cn, qianwk@sjtu.edu.cn, liuweiqiang@nuaa.edu.cn

***Abstract*—Multiplier is an important arithmetic circuit. State-of-the-art designs consist of a partial product generator (PPG), a compressor tree (CT), and a carry propagation adder (CPA), with the last two components dominating the area and delay. Existing representative works optimize the CT and the CPA separately, adding a rigid boundary between these two components. In this paper, we break the boundary by proposing GOMIL, a global optimization for multiplier by integer linear programming. Two ILP sub-problems are first formulated to optimize the CT and the prefix structure in the CPA, respectively. Then, they are unified to provide a global optimization to the multiplier. The proposed method is applicable to not only multipliers with the AND gate-based PPG, but also those with Booth encoding-based PPG. The experimental results showed that the multipliers optimized by GOMIL can reduce the power-delay product by up to 71%, compared to the state-of-the-art multipliers developed in industry. The code of GOMIL is made open-source.**

***Index Terms*—Multiplier, Compressor Tree, Prefix Tree, Integer Linear Programming, Optimization**

I. Introduction

Digital multiplier has widespread applications in many fields, such as digital signal processing and artificial neural networks, in which the power dissipation and processing performance are dominated by it. Hence, many architectures have been proposed to design multipliers with low power consumption and high speed [1]–[3]. A multiplier is composed of three main parts: a *partial product generator (PPG)*, a *compressor tree (CT)*, and a final *carry propagation adder (CPA)* [4].

There are two major types of PPGs: AND gate-based and modified Booth encoding (MBE)-based [5]. As Booth encoding can reduce the circuit delay, it is often applied in parallel multipliers [6]. The output of the PPG is a *bit matrix (BM)*, with each entry corresponding to a partial product. CT further reduces each column in BM to only 1 or 2 remaining bits by using the basic operator, the compressor. There are several famous partial product reduction schemes such as Wallace tree and Dadda tree. To generate the final product, a CPA is applied to sum up the final output BM of the CT.

Many previous works aim at improving the CT due to its occupancy of most area in a multiplier. The main focus of them is to reduce BM rapidly by improving the existing reduction schemes [3], [7]. The CPA is a key arithmetic circuit and some works propose methods to reduce its area and delay. A Kogge-Stone parallel prefix network for carry computation is proposed to significantly reduce the circuit delay, but it takes more area [8]. A high-speed CPA consisting of a prefix tree and a carry select adder (CSL) is proposed to reduce the area [9]. Moreover, a carry select and skip adder (CSSA) improving the CSL is designed for reducing the delay [10].

However, the research works focusing on improving the CT are mainly based on the heuristic methods [7], [11], [12], indicating additional room for further optimization. Furthermore, all these works optimize each part of the multiplier separately, which implies that the design is not optimal and we may achieve improvement if a global optimization is adopted.

In this paper, we propose GOMIL, a global optimization for multiplier by integer linear programming. First, an integer linear programming (ILP) problem is set up to optimize the area of the CT. Then, a dynamic programming-based method is proposed to co-optimize the area and delay of the prefix structure of the CPA, which is further transformed into an ILP problem. The above two ILP problems are then unified to globally optimize the major part of a multiplier that dominates both area and delay. For simplicity, the proposed method is illustrated on multipliers with two operands of the same length. However, it can be easily adapted to handle more general case with unequal operand length. The proposed method is applicable to not only multipliers with the AND gate-based PPG, but also those with MBE-based PPG. The experimental results showed that the GOMIL-optimized multipliers using AND gate-based PPG and MBE-based PPG can reduce the power-delay product (PDP) by up to 71% and 9%, respectively, compared to the state-of-the-art multipliers developed in industry. The code of GOMIL is made open-source at https://github.com/SJTU-ECTL/GOMIL.

II. Preliminaries and Related Works

In this section, we discuss preliminaries and related works.

A. Partial Product Generator and Compressor Tree

The output of a PPG is a BM. For example, for a 6-bit multiplier, the output BM of an AND gate-based PPG is the BM BM_0 in Fig. 1. Its height is 6.

A CT compresses the output BM of the PPG to a BM with two rows. In this work, 3:2 and 2:2 compressors are used. Note that there exist some other compressors, like 4:2 and 7:3 compressors [13]. However, they are built from 3:2 and 2:2 ones. Therefore, 3:2 and 2:2 compressors are the basic ones, which give finer granularity to build CT, leading to a larger optimization space and a potential better solution. A 3:2 (resp. 2:2) compressor is a 1-bit full (resp. half) adder. It takes 3 (resp. 2) bits as input and outputs 2 bits: a sum bit and a carry-out bit. If a 3:2 (resp. 2:2) compressor is applied at column i, then the bit number in column i reduces by 2 (resp. 1) and

This work is supported in part by the National Key R&D Program of China under Grant 2020YFB2205501 and the State Key Laboratory of ASIC & System Open Research Grant 2019KF004. Corresponding author: Weikang Qian.

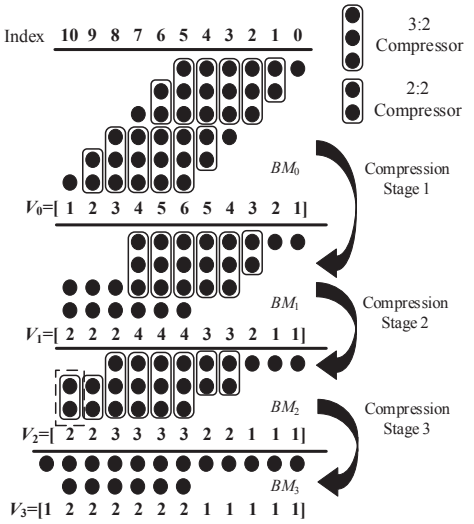


Fig. 1. The compressing process of a 6-bit Wallace tree.

that in column $(i+1)$ increases by 1. A CT has multiple compression stages. An example of the compressing process of a 6-bit Wallace tree with 3 stages is shown in Fig. 1. By the convention used in this paper, compression stage i is applied at BM BM_{i-1} and produces BM BM_i, as shown in Fig. 1.

B. Prefix Structure-based CPA

In this section, we describe prefix structure-based CPA used in multiplier. First, we introduce some definitions.

An n-bit CPA has two operands $A = a_{n-1}a_{n-2}\dots a_0$ and $B = b_{n-1}b_{n-2}\dots b_0$. For the i-th bit $(0 \le i \le n-1)$, the sum signal s_i, the carry-out signal c_i, the generate signal g_i, and the propagate signal p_i are:

$$s_i = a_i \oplus b_i \oplus c_i,\ c_i = g_i + p_i c_{i-1},\ g_i = a_i b_i,\ p_i = a_i + b_i.$$

Following [14], we define the *generate and propagate (GP) pair* of the i-th bit as (g_i, p_i). An initial GP pair is produced from an input pair (a_i, b_i) through a circuit, which is represented as an *input node* ■. We also define a Boolean operator $\circ$ on two GP pairs as [14]:

$$(g,p) \circ (g',p') = (g + p \cdot g', p \cdot p').$$

The operator $\circ$ satisfies the associative law [14]. The *group GP (GGP) pair* $(G_{i:j}, P_{i:j})$ $(0 \le j \le i \le n-1)$ is calculated from the GP pairs from continuous bit positions $i, i-1, \dots, j$ as:

$$(G_{i:j}, P_{i:j}) = (g_i, p_i) \circ (g_{i-1}, p_{i-1}) \circ \dots \circ (g_j, p_j).$$

A basic GP pair is a special case of a GGP pair when $i = j$.

By the associative law of the operator $\circ$ and the definition of the GGP pair, we have

$$(G_{i:j}, P_{i:j}) = (G_{i:k}, P_{i:k}) \circ (G_{k-1:j}, P_{k-1:j}), \qquad (1)$$

where $j < k \le i$. Thus, in more general cases, the inputs of the operator $\circ$ are two GGP pairs. The circuit that implements the operator $\circ$ is represented as an internal node ● [14].

When the CPA is used within a multiplier, its carry-in signal c_{in} is 0. Given this, the i-th carry-out signal c_i equals $G_{i:0}$, since $c_i = G_{i:0} + P_{i:0} \cdot c_{in} = G_{i:0}$. Thus, we have $(c_i, P_{i:0}) = (G_{i:0}, P_{i:0}) =$ $(G_{i:j_1}, P_{i:j_1}) \circ (G_{j_1-1:j_2}, P_{j_1-1:j_2}) \circ \dots \circ (G_{j_r-1:0}, P_{j_r-1:0})$, where $0 < j_r < j_{r-1} < \dots < j_1 \le i$. This indicates that c_i can be computed from multiple continuous GGP pairs. One classic prefix structure to generate each carry-out signal is the Kogge-Stone prefix network [8]. With a prefix structure, the final CPA can be implemented.

In [14], a hybrid parallel-prefix/carry-select (PPF/CSL) adder is proposed to reduce the area of Kogge-Stone prefix network at the cost of delay. In our work, this adder architecture is selected as the final CPA. It consists of a prefix structure for generating carry signals and a set of CSLs for producing the final sum. To further reduce the delay of the CSL, a CSSA [10] is proposed to replace the CSL. We refer the readers to [10] for details on CSL and CSSA.

C. Related Works

We point out some previous works applying ILP to optimize multipliers. In [15], [16], ILP is applied to optimize the CT of the multiplier. The basic compressor is the *generalized parallel counter* that can be efficiently realized in FPGA. The work [17] proposes to split a large multiplier into small sub-circuits that are implemented by LUTs/DSPs in FPGA. It is modeled as the tiling problem and solved by ILP. Note that the above works all aim at designing efficient multipliers for FPGA. In contrast, our work focuses on ASIC multipliers. Another work applies ILP to optimize constant multipliers [18], which are different from the focus of this paper, general multipliers.

III. Optimization of Multiplier

In this section, we present the proposed global optimization method for multipliers. We use multipliers with AND gate-based PPG to illustrate our method. It should be noted that the method is also applicable to multipliers with MBE-based PPG.

GOMIL jointly optimizes the CT and the prefix structure in the CPA. Note that the CT dominates the area of a multiplier, while the CT and the prefix structure together dominate the delay of a multiplier. Thus, by jointly optimizing the CT and the prefix structure, the majority of the area and the delay of a multiplier is optimized.

In the following, Section III-A will introduce an ILP-based formulation to optimize the CT. Section III-B will present another ILP-based formulation to optimize the prefix structure. Finally, Section III-C will describe the joint optimization by integrating these two ILP formulations together.

A. Compressor Tree Optimization

In this section, we present an ILP-based formulation for optimizing the CT. Denote the word length of the multiplier as m. As described in Section II-A, CT reduces the output BM of the PPG into a BM with two rows through multiple compression stages. We denote the number of compression stages as s. In order to minimize the delay of the CT, we fix s as the number of compression stages of an m-bit Wallace tree, as this reduction scheme provides the minimum stage number [4].

We model a BM by a *bit count vector (BCV)* $V = [x_{l-1}, x_{l-2}, \dots, x_0]$, where l is the number of columns of the BM and x_i is the number of bits at column i of the BM. We denote the BCV for the output BM of the PPG as V_0. An example of V_0 for the AND gate-based PPG of a 6-bit multiplier is shown in Fig. 1. For a general m-bit multiplier, we have

$$V_0 = \{1, 2, \dots, m-1, m, m-1, \dots, 1\}.$$

After each compression stage, since the BM changes, the corresponding BCV changes. We denote the BCV after compression stage i $(1 \le i \le s)$ as V_i. Fig. 1 shows the BCVs after the 1st, 2nd, and 3rd compression stages, i.e., V_1, V_2, and V_3.

In general case, a compressor can be applied at the leftmost column in a BM, e.g., the compressor shown in the dashed rectangle in Fig. 1. However, in this work, we avoid applying compressors at the leftmost column to simplify the later optimization formulation for the prefix structure. Although this limits the design space of the multiplier, however, the application of a compressor to the leftmost column in a BM within the compression process occurs very infrequently. Thus, the optimality is minimally affected. Under this constraint, the lengths of all the BCVs remain with the same value of $2m-1$.

We denote the numbers of 3:2 and 2:2 compressors applied at column j ($0 \leq j \leq 2m-2$) of the BM processed at stage i ($1 \leq i \leq s$) as $f_{i,j}$ and $h_{i,j}$, respectively. They are the unknowns to be solved and they determine the area of the CT. The ILP formulation is shown below.

$$\min \quad \alpha F + \beta H \tag{2}$$

$$s.t. \quad F = \sum_{i=1}^{s} \sum_{j=0}^{2m-2} f_{i,j}, \;\; H = \sum_{i=1}^{s} \sum_{j=0}^{2m-2} h_{i,j}, \tag{3}$$

$$f_{i,2m-2} = h_{i,2m-2} = 0, \text{ for } 1 \leq i \leq s, \tag{4}$$

$$f_{i,j} \geq 0, \;\; h_{i,j} \geq 0, \text{ for } 1 \leq i \leq s, 0 \leq j \leq 2m-3, \tag{5}$$

$$3f_{i,j} + 2h_{i,j} \leq V_{i-1}[j], \text{ for } 1 \leq i \leq s, 0 \leq j \leq 2m-2, \tag{6}$$

$$V_i[j] = V_{i-1}[j] - (2f_{i,j} + h_{i,j}) + (f_{i,j-1} + h_{i,j-1}), \tag{7}$$
$$\text{for } 1 \leq i \leq s, 1 \leq j \leq 2m-2,$$

$$V_i[0] = V_{i-1}[0] - (2f_{i,0} + h_{i,0}), \text{ for } 1 \leq i \leq s, \tag{8}$$
$$0 \leq V_s[j] \leq 2, \text{ for } 0 \leq j \leq 2m-2.$$

We aim at minimizing the area of all the compressors, which is modelled by Eq. (2). In Eq. (2), α and β are two constants, representing the areas of the 3:2 and 2:2 compressors, respectively; F and H are calculated by Eq. (3), which correspond to the total numbers of the 3:2 and 2:2 compressors, respectively. In this work, we set α and β as 3 and 2, respectively, according to their approximate area ratio in the NanGate 45nm Open Cell Library [19]. Eq. (4) corresponds to our enforced constraint mentioned above that no compressor can be applied at the leftmost column in the BM at any stage. Eq. (5) corresponds to the natural requirement that $f_{i,j}$ and $h_{i,j}$ should be non-negative. Eq. (6) corresponds to the requirement that the number of inputs of all the compressors applied at column j ($0 \leq j \leq 2m-2$) of the BM processed at stage i ($1 \leq i \leq s$) should be no more than the length of column j of the BM, i.e., $V_{i-1}[j]$. For example, as shown in Fig. 1, one 3:2 compressor and one 2:2 compressor are applied at column 4 of the BM processed at stage 0, BM_0. The total number of inputs of both compressors is 5, no more than $V_0[4] = 5$.

By the compression procedure depicted in Section II-A, the bit count in column j ($1 \leq j \leq 2m-2$) of the BM produced by stage i ($1 \leq i \leq s$), $V_i[j]$, should satisfy Eq. (7), since each 3:2 (resp. 2:2) compressor applied at column j reduces the bit count in column j by 2 (resp. 1), while each compressor applied at column $(j-1)$ increases the bit count in column j by 1. Eq. (8) corresponds to the special case of $V_i[j]$ occurs at column 0. Finally, Eq. (9) corresponds to the requirement that each entry in the BCV after the final stage, V_s, should be non-negative and no more than 2.

B. Prefix Structure Optimization

As described in Section II-B, the basic operands of the prefix structure in a CPA are the GGP pairs, while the basic operators are the input nodes ■ and the internal nodes ●, both taking two GGP pairs as input and outputting a GGP pair.

In the context of designing multipliers, the input to the prefix structure in a CPA is an irregular BM of two rows, with some columns containing just 1 bit, e.g., the BM BM_3 shown in Fig. 1, which will be further processed by a CPA. This is different from the general adder where each bit position has two input bits. Such a difference leads to a further optimization opportunity, which we explore in this section.

1) Modelling: In order to illustrate the proposed optimization method later, we first present some related modelling. We denote the input pair at column i as (u_i, v_i). First, an input node ■ for a bit position can be simplified if that position has just 1 input bit. In this case, the input pair can be represented as $(0, v_i)$ and by definition, the GP pair (g_i, p_i) reduces to $(0, v_i)$, which requires no logic gate in the implementation. To capture this special case, we denote the input node for this case as □.

Furthermore, an input GGP pair to an internal node, $(G_{i:k}, P_{i:k})$, can be a special case with $G_{i:k} = 0$, leading to simplification of the internal node. Given this observation, we distinguish two types of GGP pairs, one with $G_{i:j}$ as a constant 0 and the other with $G_{i:j}$ as a normal variable. We introduce a binary variable $b_{i:j}$ to indicate these two types: $b_{i:j} = 0$ and 1 correspond to the first and the second types, respectively. Note that the type of an input node (i.e., ■ or □) is characterized by the type of its input GP pair $(G_{i:i}, P_{i:i})$ and thus, can be uniquely identified by the variable $b_{i:i}$, which is listed in Table I.

TABLE I
THE ENCODING, AREAS, AND DELAYS OF THE BASIC NODES.

Node	Symbol	Encoding	Area	Delay
Input	□	$b_{i:i} = 0$	0	0
	■	$b_{i:i} = 1$	2	1
Internal	○	$(b_{i:k}, b_{k-1:j}) = (0,0)$	1	1
	▲	$(b_{i:k}, b_{k-1:j}) = (0,1)$	2	1
	△	$(b_{i:k}, b_{k-1:j}) = (1,0)$	1	1
	●	$(b_{i:k}, b_{k-1:j}) = (1,1)$	3	2

For an internal node, it takes two GGP pairs $(G_{i:k}, P_{i:k})$ and $(G_{k-1:j}, P_{k-1:j})$ as input. Its logic function can be simplified differently for different type combinations of the two GGP pairs. Specifically, there are four type combinations. Encoded by the pair $(b_{i:k}, b_{k-1:j})$, they are $(0,0)$, $(0,1)$, $(1,0)$, and $(1,1)$. Correspondingly, we introduce four types of internal nodes, ○, ▲, △, and ●, with Boolean functions listed below:

$$\begin{aligned}
&\bigcirc : (G_{i:j}, P_{i:j}) = (0, P_{i:k}) \circ (0, P_{k-1:j}) \\
&\quad = (0,\ P_{i:k} \cdot P_{k-1:j}) \\
&\blacktriangle : (G_{i:j}, P_{i:j}) = (0, P_{i:k}) \circ (G_{k-1:j}, P_{k-1:j}) \\
&\quad = (P_{i:k} \cdot G_{k-1:j}, P_{i:k} \cdot P_{k-1:j}) \\
&\triangle : (G_{i:j}, P_{i:j}) = (G_{i:k}, P_{i:k}) \circ (0, P_{k-1:j}) \\
&\quad = (G_{i:k},\ P_{i:k} \cdot P_{k-1:j}) \\
&\bullet : (G_{i:j}, P_{i:j}) = (G_{i:k}, P_{i:k}) \circ (G_{k-1:j}, P_{k-1:j}) \\
&\quad = (G_{i:k} + P_{i:k} \cdot G_{k-1:j}, P_{i:k} \cdot P_{k-1:j})
\end{aligned} \tag{9}$$

Their logic-level implementations can be easily derived from the above equations. Table I lists the one-to-one mapping between the node type and the corresponding $(b_{i:k}, b_{k-1:j})$ pair.

Next, we mention a few properties of $b_{i:j}$, as they will be used in later solution of the optimization problem. First, we consider $b_{i:i}$, which corresponds to the GGP pair produced by an input node. It relates to the input to a prefix structure, which is the output BCV V_s of the CT. As mentioned in Section III-A, we avoid applying compressors at the leftmost column of a BM.

This ensures that there is no leading 0 in V_s. Consequently, each entry of V_s can only be 1 or 2. By definition, $b_{i:i}$ equals 0, if $V_s[i] = 1$, and 1, if $V_s[i] = 2$. Thus, we have

$$b_{i:i} = V_s[i] - 1. \tag{10}$$

Second, we derive $b_{i:j}$ for the output GGP pair of an internal node taking input GGP pairs with the type combination encoded by $(b_{i:k}, b_{k-1:j})$. By Eq. (9), we have the following relation

$$b_{i:j} = \begin{cases} 0 & \text{if } b_{i:k} = b_{k-1:j} = 0, \\ 1 & \text{otherwise,} \end{cases}$$

which is equivalent to

$$b_{i:j} = b_{i:k} + b_{k-1:j} - b_{i:k}b_{k-1:j}. \tag{11}$$

Obviously, the areas and delays of the two input nodes and four internal nodes differ. The area and delay of these nodes used in our modeling are listed in Table I, which are based on the approximate area and delay ratios of the gates in the NanGate 45nm library [19]. To facilitate the later optimization solution, we further establish an analytical relation between the area/delay of a node and its encoding. Based on the values listed in Table I, for an input node, its area A and delay D satisfy

$$A(b_{i:i}) = 2b_{i:i}, \quad D(b_{i:i}) = b_{i:i}. \tag{12}$$

For an internal node, its area A and delay D satisfy

$$\begin{aligned} A(b_{i:k}, b_{k-1:j}) &= b_{i:k}b_{k-1:j} + b_{k-1:j} + 1, \\ D(b_{i:k}, b_{k-1:j}) &= b_{i:k}b_{k-1:j} + 1. \end{aligned} \tag{13}$$

Note that the above relations depend on the particular area and delay values we use. However, for a different set of area and delay values, we can easily derive a similar set of relations, only with different coefficients.

2) Optimization Problem and Solution: In this section, we present the optimization problem and the proposed solution.

For an input BCV V_s of length $2m-1$, we focus on synthesizing the prefix tree producing the final GGP pair $(G_{2m-2:0}, P_{2m-2:0})$. First, we observe that different internal node connections may lead to different prefix trees with different areas and delays, which leads to an optimization problem.

Example 1. *Suppose that the input BCV of a prefix structure is* $[2, 2, 1, 2, 1, 1]$. *Two prefix trees producing the final GGP pair* $(G_{5:0}, P_{5:0})$ *with different connections of the internal nodes are shown in Fig. 2. Note that we represent the two types of GGP pairs using different lines. The red lines in each figure plot a critical path of the prefix tree. According to Table I, the area and delay of the first prefix tree are* 16 *and* 6, *respectively, while those of the second are* 16 *and* 5, *respectively.* □

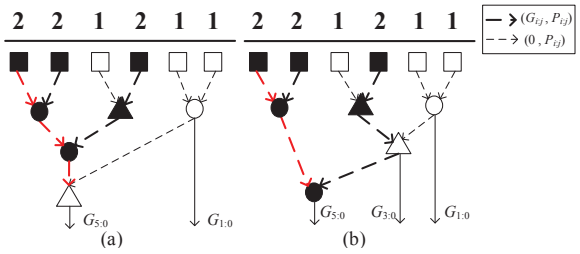


Fig. 2. Two prefix trees for the same input BCV.

Since the operator $\circ$ satisfies the associative law, the final computed GGP pair is independent of the connection order of the internal nodes. Our goal is to find the best connection order for an input BCV to co-minimize the area and delay of the prefix tree.

In order to solve the optimization problem, we first define a general problem $Q_{i:j}$, which is constructing an optimal prefix tree for producing the GGP pair $(G_{i:j}, P_{i:j})$. Then, the original problem to be solved is just $Q_{2m-2:0}$. We denote the optimal solution of $Q_{i:j}$ as $S_{i:j}$. Since our target is to co-minimize the area and delay, we use a linear combination of the area A and the delay D of a prefix tree as the optimization target, i.e., $C = A + wD$, where w is a parameter controlling the weight of delay in the optimization target. Denote the optimal cost of problem $Q_{i:j}$ as $c_{i:j}$ and the area and delay of the optimal solution $S_{i:j}$ as $a_{i:j}$ and $d_{i:j}$, respectively. Clearly, we have

$$c_{i:j} = a_{i:j} + wd_{i:j}.$$

As shown in Fig. 2(a), producing a GGP pair $(G_{i:j}, P_{i:j})$ with $i > j$ reduces to producing two GGP pairs $(G_{i:k}, P_{i:k})$ and $(G_{k-1:j}, P_{k-1:j})$ for a particular $k \in \{j+1, j+2, \ldots, i\}$ and then combining the GGP pairs by a proper internal node. We refer to the value k as the *cut point*. For example, producing the GGP pair $(G_{5:0}, P_{5:0})$ in Fig. 2(a) is achieved by producing the GGP pairs $(G_{5:2}, P_{5:2})$ and $(G_{1:0}, P_{1:0})$ first and then combining them by an internal node $\triangle$. The cut point is $k = 2$.

Given the above construction pattern, we can see that in order to construct an optimal solution for problem $Q_{i:j}$, we need to find a proper cut point k and the optimal solutions for two sub-problems $Q_{i:k}$ and $Q_{k-1:j}$. This leads to a dynamic programming-based solution as described next.

For the base case where $i = j$, the prefix tree just contains a single input node. Thus, we have

$$a_{i:i} = A(b_{i:i}), \quad d_{i:i} = D(b_{i:i}), \tag{14}$$

where $A(b_{i:i})$ and $D(b_{i:i})$ are given by Eq. (12).

Now suppose we know $c_{i:j}$ for any $0 \le j \le i \le 2m-2$ satisfying that $i - j \le l$ together with the area $a_{i:j}$ and delay $b_{i:j}$ of the optimal solution. Next we try to get $c_{i:j}$ for any $0 \le j \le i \le 2m-2$ satisfying that $i - j = l + 1$.

If the cut point is k, then we have the area of the optimal solution $S_{i:j}$ as $a_{i:j} = a_{i:k} + a_{k-1:j} + A(b_{i:k}, b_{k-1:j})$ and the delay as $d_{i:j} = \max\{d_{i:k}, d_{k-1:j}\} + D(b_{i:k}, b_{k-1:j})$, where $A(b_{i:k}, b_{k-1:j})$ and $D(b_{i:k}, b_{k-1:j})$, given by Eq. (13), correspond to the area and delay, respectively, of the internal node combining the GGP pairs $(G_{i:k}, P_{i:k})$ and $(G_{k-1:j}, P_{k-1:j})$. Thus, we have

$$\begin{aligned} c_{i:j} &= a_{i:j} + wd_{i:j} = a_{i:k} + a_{k-1:j} + A(b_{i:k}, b_{k-1:j}) \\ &\quad + w\left(\max\{d_{i:k}, d_{k-1:j}\} + D(b_{i:k}, b_{k-1:j})\right). \end{aligned}$$

However, since the cut point is unknown, the actual $c_{i:j}$ should be calculated as

$$\begin{aligned} c_{i:j} = \min_{j<k\le i}\{&a_{i:k} + a_{k-1:j} + A(b_{i:k}, b_{k-1:j}) \\ &+ w\left(\max\{d_{i:k}, d_{k-1:j}\} + D(b_{i:k}, b_{k-1:j})\right)\}. \end{aligned} \tag{15}$$

The above equation solves $c_{i:j}$ for any $i - j = l + 1$. In order to solve for $c_{i:j}$ with any $i - j > l + 1$ later, it is critical to also record the values $a_{i:j}$ and $b_{i:j}$ of the optimal solution $S_{i:j}$. In other words, suppose $p_{i:j}$ is the *optimal cut point* giving the minimum $c_{i:j}$. We need to obtain the following two values:

$$\begin{aligned} a_{i:j} &= a_{i:p_{i:j}} + a_{p_{i:j}-1:j} + A(b_{i:p_{i:j}}, b_{p_{i:j}-1:j}), \\ d_{i:j} &= \max\{d_{i:p_{i:j}}, d_{p_{i:j}-1:j}\} + D(b_{i:p_{i:j}}, b_{p_{i:j}-1:j}). \end{aligned} \tag{16}$$

By repeating the above procedure until $i - j = 2m-2$, we can eventually obtain the optimal solution for problem $Q_{2m-2:0}$.

Note that the optimization of CT is based on ILP, a special case of integer programming (IP). In order to facilitate a global

optimization over the CT and the prefix structure together, we further transform the above dynamic programming-based solution into an IP, as shown below.

$$\min \quad c_{2m-2:0} \tag{17}$$

$$s.t. \quad b_{i:i} = V_s[i] - 1, \text{ for } 0 \le i \le 2m-2, \tag{18}$$

$$b_{i:j} = b_{i:i} + b_{i-1:j} - b_{i:i}b_{i-1:j}, \text{ for } 0 \le j < i \le 2m-2, \tag{19}$$

$$a_{i:i} = 2b_{i:i}, \ d_{i:i} = b_{i:i}, \text{ for } 0 \le i \le 2m-2, \tag{20}$$

$$c_{i:j} = \min_{j<k\le i}\{(a_{i:k}+a_{k-1:j}+b_{i:k}b_{k-1:j}+b_{k-1:j} + 1) + w(\max\{d_{i:k}, d_{k-1:j}\}+b_{i:k}b_{k-1:j}+1)\}, \text{ for } 0 \le j < i \le 2m-2, \tag{21}$$

$$t_{ijk} \in \{0,1\}, \text{ for } 0 \le j < i \le 2m-2, j < k \le i, \tag{22}$$

$$\sum_{k=j+1}^{i} t_{ijk} = 1, \text{ for } 0 \le j < i \le 2m-2, \tag{23}$$

$$a_{i:j} = \sum_{k=j+1}^{i} t_{ijk}(a_{i:k}+a_{k-1:j}+b_{i:k}b_{k-1:j}+b_{k-1:j}+1), \text{ for } 0 \le j < i \le 2m-2 \tag{24}$$

$$d_{i:j} = \sum_{k=j+1}^{i} t_{ijk}(\max\{d_{i:k}, d_{k-1:j}\}+b_{i:k}b_{k-1:j}+1), \text{ for } 0 \le j < i \le 2m-2 \tag{25}$$

$$c_{i:j} = a_{i:j} + wd_{i:j}, \text{ for } 0 \le j \le i \le 2m-2. \tag{26}$$

Eq. (17) gives the final optimization target, which is the optimal cost of problem $Q_{2m-2:0}$. The variable $b_{i:j}$'s introduced before play an important role in the IP formulation, as the area and delay of a node can be characterized by them. Thus, the formulation has Eqs. (18) and (19); the former is just Eq. (10) and the latter is derived from Eq. (11) by setting the variable k as i. Eq. (20) corresponds to the base case values in the dynamic programming-based solution. It is derived from Eqs. (12) and (14). Eq. (21) corresponds to the key recurrence relation in the dynamic programming-based solution. It is derived from Eqs. (13) and (15).

As we mentioned before, the values $a_{i:j}$ and $d_{i:j}$ of the optimal solution $S_{i:j}$ are needed to drive the recurrence in the dynamic programming-based solution. They depend on the optimal cut point $p_{i:j}$, as shown in Eq. (16). However, it is challenging to obtain $p_{i:j}$ directly in an IP formulation and hence, challenging to obtain $a_{i:j}$ and $d_{i:j}$ by Eq. (16). To solve this problem, we introduce binary variables t_{ijk} to indicate the optimal cut point $p_{i:j}$: $t_{ijk} = 1$ ($j < k \le i$) if and only if $p_{i:j} = k$. Based on the definition of t_{ijk}, Eqs. (22) and (23) give the basic constraint on it. Then, by Eqs. (24), (25), and (26), we are able to get $a_{i:j}$ and $d_{i:j}$.

The above formulation is not an ILP since it contains three basic non-linear components: $\max\{x, y\}$, $\min\{x, y\}$, and $b \cdot x$ with b as a binary variable. Fortunately, all of them can be transformed into linear constraints [20]. Thus, the above IP formulation can be transformed into an ILP formulation. Due to the space limit, we omit the final ILP formulation.

C. Global Optimization

In this section, we present the joint optimization for the CT and the prefix structure. For this purpose, we only need to put the constraints of the two ILP formulations together and adjust the final objective function as:

$$\alpha F + \beta H + c_{2m-2:0}. \tag{27}$$

Note that the previous two ILP formulations share the common variable $V_s[i]$'s, which serve as the connection between the two ILP formulations.

The final ILP formulation can be solved efficiently for a small word length m. However, as the size of the ILP increases with m, it causes a long time to solve for a large m. In order to improve the runtime, we introduce a speed-up technique by defining an upper bound parameter L ($1 \le L \le 2m-1$). With this L, we only keep those constraints in Eqs. (19) and (21)–(26) satisfying that $i-j < L$. We also update the final objective function to $\alpha F + \beta H + c_{L-1:0}$. After solving this modified global ILP problem, a specific BCV V_s will be obtained. We then reuse the normal ILP formulation for the prefix structure to generate the final prefix tree for this particular V_s.

After the joint optimization of the CT and the prefix structure, we further include CSLs for computing the final product. However, in some cases, the length of some CSLs is long, causing a CSL to dominate the delay of a CPA. In this case, a CSSA can be used to replace the CSL to reduce the delay.

IV. Experimental Results

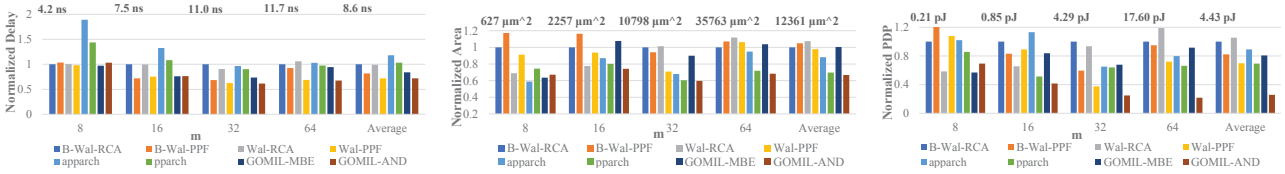
In this section, we present the experimental results of the proposed GOMIL method. The mixed ILP solver Gurobi Optimizer 9 [21] was used to solve the ILP problems. We also implemented a C++ program that takes the solution of the optimization problem as input and generates Verilog HDL code for the final optimized multiplier. It was further synthesized by the Synopsys Design Compiler using a 2008-version typical NanGate 45nm Open Cell Library and placed and routed by Cadence Innovus. Synopsys PrimeTime was used to measure the delay and power. As some designs have a delay larger than 10ns, power was measured at 50MHz working frequency.

The proposed method has two important parameters: the weight w for controlling the weight of delay in the optimization target in the ILP for the prefix structure and the parameter L for accelerating the ILP solving. We set $w = 8$ and $L = 10$, as this combination gives a small area-delay product, while ensuring an affordable runtime, according to our experimental results. Since our proposed method takes a very long runtime for a large L and m, we set a empirical runtime bound of $(3600 + L^3)$ seconds for each benchmark, where the term L^3 comes from the fact that the number of variables and the number of constraints of our ILP formulation are both $\Theta(L^3)$.

We compared the hardware cost of the GOMIL-optimized multiplier with existing fast multipliers. We applied GOMIL to optimize two types of multipliers, those with AND-gate based PPG and those with MBE-based PPG. They are denoted as *GOMIL-AND* and *GOMIL-MBE*, respectively.

The existing multipliers include *Wal-RCA*, *Wal-PPF*, *B-Wal-RCA*, *B-Wal-PPF*, *pparch*, and *apparch*. *Wal-RCA* and *Wal-PPF* are based on Wallace compressor tree structure. *Wal-RCA* uses the ripple carry adder (RCA) as the final CPA, while *Wal-PPF* uses the PPF/CSL structure. *B-Wal-RCA* and *B-Wal-PPF* are similar to *Wal-RCA* and *Wal-PPF* except that they are based on Booth encoding. *pparch* and *apparch* are provided by DesignWare [22]. They are state-of-the-art multipliers from industry, and optimized for delay and area, respectively. For any given word length, both *pparch* and *apparch* will consider various multiplier architectures, including those based on Radix-2 non-Booth, Radix-4 Booth recoded, and Radix-8 Booth recoded, and finally pick the optimal choice.

We considered 4 word lengths $m = 8, 16, 32, 64$. Fig. 3 plots the comparison between the proposed multipliers and the other existing multipliers on delay, area, and PDP, which are all normalized to those of *B-Wal-RCA*. Each figure also includes the average result over the 4 word lengths.



(a) (b) (c)

Fig. 3. Delays, areas, and power-delay products of the existing and the proposed multipliers for different word lengths. The values of *B-WAL-RCA* are listed above each group.

As shown in Fig. 3(a), for traditional multipliers, *Wal-PPF* is faster than *Wal-RCA*. This shows that the prefix structure-based CPA helps improve the delay. GOMIL multipliers also apply such adders. As a result, compared to *B-Wal-RCA* and *Wal-RCA*, *GOMIL-AND* reduces the delay by 27.45% and 27.26%, respectively, on average. Furthermore, due to the optimization of the prefix structure, *GOMIL-AND* is even 8.75% and 1% faster than the prefix structure-based multipliers *B-Wal-PPF* and *Wal-PPF*, respectively. Compared to *apparch* and *pparch* from industry, *GOMIL-AND* improves the delay by 41.02% and 31.84%, respectively. The other GOMIL multiplier, *GOMIL-MBE*, is also faster than *apparch* and *pparch*.

As shown in Fig. 3(b), the areas of the prefix structure-based traditional multipliers are mostly larger than those of the RCA-based ones. Thus, prefix structure is area-consuming. Owing to the area optimization of GOMIL, *GOMIL-AND* reduces area by 36.41% and 31.88% over *B-Wal-PPF* and *Wal-PPF*, respectively. Moreover, *GOMIL-AND* is 33.36% and 37.99% smaller than *B-Wal-RCA* and *Wal-RCA*, respectively. Compared to *apparch* and *pparch*, *GOMIL-AND* reduces the area by 24.36% and 4.4%, respectively. In contrast, *GOMIL-MBE* is 18.37% and 28.29% larger than *apparch* and *pparch*, respectively. One reason is that the depth of the BM is smaller for GOMIL-MBE, leading to smaller optimization space for CT. Although *GOMIL-MBE* is larger than *GOMIL-AND* in most cases, it is more competitive for small m on both area and delay. Thus, it is more suitable for small multipliers.

The power comparison of GOMIL multipliers over the existing multipliers has a similar trend as the area, and is omitted due to space limit. In terms of the more comprehensive measure, PDP, as shown in Fig. 3(c), the DesignWare multipliers, particularly *pparch*, are the best ones among the 6 existing multipliers, which shows the superiority of the industrial design. *GOMIL-AND* can further reduce the PDP by 70.99% and 62.74% over *apparch* and *pparch*, respectively, demonstrating the effectiveness of GOMIL. The runtime of GOMIL for $m = 8$, 16, 32, and 64 is 2325s, 4840s, 5510s and 7200s, respectively. Although GOMIL takes a long runtime for a large word length, it is still affordable as the design of a multiplier is usually a one-time effort.

V. Conclusions

In this work, we propose GOMIL, a novel global optimization technique for designing better multipliers. The main idea of GOMIL is to establish an ILP formulation to jointly optimize the CT and the prefix structure in a multiplier, which together dominate the area and delay of a multiplier. GOMIL breaks the rigid boundary between the CT and the CPA in a multiplier and hence, can obtain multipliers that have a lower PDP than the state-of-the-art multipliers from industry. In our future work, we plan to extend GOMIL to synthesize multipliers for FPGA architecture and approximate multipliers.

References

[1] W.-C. Yeh and C.-W. Jen, "High-speed Booth encoded parallel multiplier design," *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 692–701, 2000.

[2] J.-Y. Kang and J.-L. Gaudiot, "A simple high-speed multiplier design," *IEEE Trans. Comput.*, vol. 55, no. 10, pp. 1253–1258, 2006.

[3] X.-V. Luu *et al.*, "A high-speed unsigned 32-bit multiplier based on Booth-encoder and Wallace-tree modifications," in *ATC*, 2014, pp. 739–744.

[4] K. A. C. Bickerstaff, M. Schulte, and E. E. Swartzlander, "Reduced area multipliers," in *ASAP*, 1993, pp. 478–489.

[5] F. Elguibaly, "A fast parallel multiplier-accumulator using the modified Booth algorithm," *IEEE Trans. Circuits Syst. II*, vol. 47, no. 9, pp. 902–908, 2000.

[6] S. Kuang, J. Wang, and C. Guo, "Modified Booth multipliers with a regular partial product array," *IEEE Trans. Circuits Syst. II*, vol. 56, no. 5, pp. 404–408, 2009.

[7] N. Itoh *et al.*, "A 32/spl times/24-bit multiplier-accumulator with advanced rectangular styled Wallace-tree structure," in *ISCAS*, 2005, pp. 73–76.

[8] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. Comput.*, vol. C-22, no. 8, pp. 786–793, 1973.

[9] S. Das and S. P. Khatri, "A novel hybrid parallel-prefix adder architecture with efficient timing-area characteristic," *IEEE Trans. VLSI Syst.*, vol. 16, no. 3, pp. 326–331, 2008.

[10] X. Cui *et al.*, "Design of high-speed wide-word hybrid parallel-prefix/carry-select and skip adders," *J. Sign. Process. Syst.*, vol. 90, pp. 409–419, 2018.

[11] J. Fadavi-Ardekani, "M*N Booth encoded multiplier generator using optimized Wallace trees," *IEEE Trans. VLSI Syst.*, vol. 1, no. 2, pp. 120–125, 1993.

[12] M. Faust and C. Chang, "Reduction of partial product matrix for high-speed single or multiple constant multiplication," in *PrimeAsia*, 2010, pp. 416–420.

[13] P. J. Song and G. De Micheli, "Circuit and architecture trade-offs for high-speed multiplication," *IEEE J. Solid-State Circuits*, vol. 26, no. 9, pp. 1184–1198, 1991.

[14] G. Dimitrakopoulos and D. Nikolos, "High-speed parallel-prefix VLSI Ling adders," *IEEE Trans. Comput.*, vol. 54, no. 2, pp. 225–231, 2005.

[15] M. Kumm and P. Zipf, "Pipelined compressor tree optimization using integer linear programming," in *FPL*, 2014, pp. 1–8.

[16] H. Parandeh-Afshar *et al.*, "Improving synthesis of compressor trees on FPGAs via integer linear programming," in *DATE*, 2008, pp. 1256–1261.

[17] M. Kumm *et al.*, "Resource optimal design of large multipliers for FPGAs," in *ARITH*, 2017, pp. 131–138.

[18] M. Kumm, "Optimal constant multiplication using integer linear programming," *IEEE Trans. Circuits Syst. II*, vol. 65, no. 5, pp. 567–571, 2018.

[19] "Nangate 45nm open cell library," 2008. [Online]. Available: http://www.nangate.com/

[20] Kolman and Bernard, *Elementary Linear Programming with Applications*. Academic Press, 1995.

[21] Gurobi Optimization, LLC, "Gurobi optimizer reference manual," 2020. [Online]. Available: http://www.gurobi.com

[22] "DesignWare IP," 2015. [Online]. Available: http://www.synopsys.com/designware