An Efficient Hierarchical Clustering Method for the Multiple Constant Multiplication Problem

Akihiro Matsuura, Mitsuteru Yukishita, Akira Nagoya

NTT Communication Science Laboratories
2 Hikaridai, Seika-cho, Soraku-gun,
Kyoto, 619-02, Japan
E-mail: matsu@cslab.kecl.ntt.jp

Abstract— In this paper, we propose an efficient solution for the Multiple Constant Multiplication(MCM) problem. The method exploits common subexpressions among constants based on hierarchical clustering and reduce the number of shifts, additions, and subtractions. The algorithm defines appropriate weights which indicate the operation priorities and selects the common subexpressions which results in the least number of local operations. It can also be extended to various high-level synthesis tasks such as arbitrary linear transforms. Experimental results show the effectiveness of our method.

I. Introduction

Recently, high-level synthesis methodologies have played an important role in VLSI design automation. Many tasks in high-level synthesis such as design representation, transformation, scheduling, and allocation have been exploited. In addition, powerful high-level synthesis systems with these techniques have been introduced over the last decade [1], [2].

Among the high-level synthesis tasks, transformation to behavioral descriptions and to internal flow-graph representations is quite important to achieve high-quality design. Such transformation involves various compiler-like optimizations such as constant folding, redundant operator elimination. Another powerful transformation technique, the tree-height reduction, uses the algebraic properties of operators such as commutativity and distributivity to decrease the height of the parse tree. This technique has been successfully applied to high-level synthesis to improve the parallelism of the design [3].

The problem we consider in this paper is one of these flow-graph transformations, that is, substituting multiplications with a single constant by shifts and additions, and minimizing the number of these operations. The motivation for such a substitution is simply that in ASICs and processors, several multiplication costs can be substantially reduced by implementing with shifts, adders, and subtractors rather than with multipliers. The importance of such optimization has been recognized for high-level synthesis. Only recently, however, has this transformation been investigated to reduce power consumption [4] and area size [5].

The significant advance for the transformation was achieved by Potkonjak *et al.* in [6], [7]. They first formulated the Multiple Constant Multiplication(MCM) problem in high level synthesis by considering the multiplications of one variable with several constants at a time and also reduced the number of the shifts and additions based on iterative pairwise matching. Mehendale *et al.*[8] considered the problem by examining the coefficient matrix and the iterative elimination of two-element common subexpressions.

In this paper, we first present new weights for subexpressions indicating priorities to be executed earlier, and present the algorithm to minimize the numbers of additions+subtractions and shifts by using them. Since the definition of the weights takes into account not only a direct common subexpression but also subexpressions which can be computed by just shifting the other, it can fully express the operation priorities and explore the search space quite effectively. Our method can be applied for various number representations such as Signed Digit(SD) representation. Furthermore, it can be extended to a general linear transform with arbitrary elements, and thus, applied to various kinds of high-level sysnthesis tasks, especially for numerically intensive applications.

This paper is organized as follows: In the next section, we summarize the MCM problem using an example. In Section III, we show our hierarchical clustering method to thoroughly explore common subexpressions. In Section IV, we extend

ASP-DAC '97
0-89791-851-7$5.00 © 1997 IEEE

our basic scheme to arbitrary linear transforms to aim at several high-level synthesis tasks Section V is devoted to the experimental results. Finally, we conclude in Section VI.

II. Problem Formulation

A. Previous Work

Optimization of multiplications with a constant has been investigated from several viewpoints, such as on software compilers [9], computer architecture [10], DSP [11], and so forth. Chatterjee *et al.*[5] have addressed this problem to improve area size and have presented algorithms based on number splitting. However in these tecniques, the simultaneous optimization of multiplications of one variable by multiple constants has not been fully explored.

Recently, another significant advance was achieved by Potkonjak *et al.*[6], [7] and by Mehendale *et al.*[8]. Potkinjak *et al.* considered all of the constants multiplied by the same variable at the same point in time and formulated the MCM problem as follows: *Substitute all multiplications with constants by shifts and additions* (*and subtractions*)*, and use common subexpressions between various multiplications to minimize the number of additions* (*and subtractions*). They explored common subexpressions among multiple constants using the iterative pairwise matching algorithm. Note that this algorithm works better than the simple bipartite matching, even though it possibly has some defects in searching for common subexpressions. Mehendale *et al.* also considered the MCM problem by looking for common 2-bit subexpressions across bit locations and those within a coefficient. In doing this, they suceeded in further reducing in the number of additions+subtractions. However, since they explore common subexpressions across constants and those within a constant separately and explore only those of '2-bit' types, better common subexpressions which exist in many bit locations across the constants and within a constant can be missed. Thus, we explore the common subexpressions across the constants and those within each constant simultaneously and decide the best subexpression to be executed by the priority criteria which takes both types of common subexpressions into account. Furthermore, we extend the basic algorithm to a wider range of problems, such as to the MCM problem for a linear transform with arbitrary entries.

B. Exploring Common Subexpressions – An Example

We introduce the MCM problem using an example and mention the basic idea.

	Constant	Binary representation
a	815	1100101111
b	365	0101101101
c	831	1100111111
d	121	0001111001

Fig. 1. Values of the constants and their binary representations

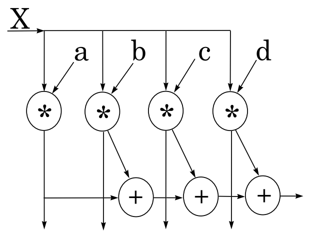

Fig. 2. Example of multiple constant multiplications

Example 1. First, consider the following description: $y = a * X + b * X + c * X + d * X$. The concrete values of the constants from a to d are shown in Fig. 1 and the simplified circuit in Fig. 2.

Not only are the constants integers but even arbitrary fixed point numbers are permissible. Before explaining the algorithm, we need to make one assumption. If all we need is the totazl value of the right side of the description, then it does not make sense to find the common expression, because in that case, all of the constants can be added in advance using the distributive property. Thus, we assume that we need each of the product elements $a * X, b * X, c * X$, and $d * X$ and that we do not add the constants in advance. This assumption is appropriate when we formulate the MCM problem on various applications.

We first represent all of the constants in binary form as shown in Fig. 1. By this representation, an addition between two shifted numbers can be shared by all of the constants that have common 1s in the same two figures. We will demonstrate how to share the subexpressions: If we first shift and add the numbers corresponding to the common 1s in the first, fourth, and sixth figures as shown in Fig. 3, we need only two additions among the three figures and two shifts for the 1s in the fourth and sixth figures. However, without any sharing, we need eight additions and eight shifts for the computation. Thus, six additions and six shifts are saved by that sharing.

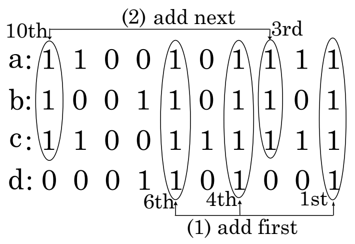

Fig. 3. Clustering based on 2-bit common subexpressions

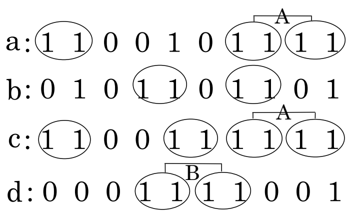

Fig. 4. Clustering with a substitution of shifts for additions

We then process the computation for the third and 10th figures. Continuing in the same way, we need 10 additions and eight shifts, while initially 22 additions and 22 shifts were needed for the multiplications of a, b, c, and d.

However, this is not sufficient because it does not utilize common subexpressions within each constant. That is, as in Fig. 4 where the sum of the shifted numbers corresponding to the 1s in the third and fourth figures of the constant a can be just computed by shifting the sum for its first and second figures as follows:

$$X \ll 2 + X \ll 3 = (X + X \ll 1) \ll 2. \qquad (1)$$

Similarly, the sum corresponding to the subexpression B of d in Fig. 4 can be computed in terms of the subexpression A in Fig. 4 as follows:

$$\sum_{i=3}^{6}(X \ll i) = \{\sum_{i=0}^{3}(X \ll i)\} \ll 3 \qquad (2)$$

By rigorously executing such substitutions of shifts for additions, we can reduce the number of additions to nine, where the number of shifts is unchanged, eight in total.

We extract common subexpressions in such an order that a pair of bits that exists most frequently across the constants and within a constant is given the highest priority and shifted and added first. We progress with the algorithm until all bits have been added and the whole computation is completed.

The above example describes techniques for exploring the common subexpressions across constants and those within each constant. We will show the details of the algorithm in the next section.

III. Hierarchical Clustering Algorithm

A. Preliminaries

The MCM problem can be formally stated as the minimization problem of a weighted sum of the numbers of shifts, additions, and subtractions. Thus, we first minimize the number of additions+subtractions and then, minimize the number of shifts under the assumption that varrel shifts are used and cost of all shifts is identical.

We first need to introduce some notation to support the algorithm. Consider a variable X multiplied by multiple constants $\{a_k\}(0 \le k \le n-1)$ where each of them has m bits. We first express all of the constants in some representation taking the values 1, -1, and 0. Let $V = \{v_i\}(0 \le i \le m-1)$ be the set of m vertices where v_i represents the ith figure from the right of all constants. We express the value of the ith figure of a_k by $v_i[k]$. Then, the representation of a_k is written as $v_{m-1}[k]v_{m-2}[k] \cdots v_1[k]v_0[k]$, which implies that the values $(v_i[k])$ $(0 \le k \le n-1, 0 \le i \le m-1)$ form a $n \times m$ matrix whose elements are 1, -1, or 0. We call $(v_i[k])$ *the digit matrix*.

$$(v_i[k]) = \left(\begin{array}{cccc} v_{m-1}[0] & \ldots & v_1[0] & v_0[0] \\ v_{m-1}[1] & \ldots & v_1[1] & v_0[1] \\ \ldots & \ldots & \ldots & \ldots \\ v_{m-1}[n-1] & \ldots & v_1[n-1] & v_0[n-1] \end{array}\right)$$
$$\begin{array}{cccc} v_{m-1} & \ldots & v_1 & v_0 \end{array}$$

Fig. 5. Digit matrix

Next, we define two kinds of weights between v_i and v_j to express priorities among the subexpressions as follows:

$$\begin{aligned} W^{+}_{i,j} = \#\{(v_p[k], v_q[k]) = (\pm1, \pm1) \mid 0 \le \exists l \le m-2\ s.t. \\ X \ll p + X \ll q = \pm(X \ll i + X \ll j) \ll l\} \\ W^{-}_{i,j} = \#\{(v_p[k], v_q[k]) = (\pm1, \mp1) \mid 0 \le \exists l \le m-2\ s.t. \\ X \ll p - X \ll q = \pm(X \ll i - X \ll j) \ll l\} \end{aligned}$$

Each of the weights $W^{\pm}_{i,j}$ is equal to the number of 2-bit subexpressions which can be computed by just shifting the addition(or subtraction) result of $X \ll i \pm X \ll j$ to the left. Hence, all the subexpressions used to calculate each weight can be performed with only one addition(or subtraction).

B. Hierarchical Clustering Procedure

We explain how to calculate $\{W^{\pm}_{i,j}\}$ and how to select subexpressions by using the following example.

Example 2. Suppose a variable X is multiplied by the four constants $a, b, c,$ and d where the binary representations of the constants are $a = 1\,1\,1\,1_2,\ b = 1\,1\,0\,1_2,\ c = 0\,1\,1\,0_2$, and $d = 1\,0\,1\,1_2$. Then, there are initially four vertices corresponding to the four bit elements, *i.e.* $V = \{v_0,\ v_1,\ v_2,\ v_3\}$. The digit matrix that consists of the bit elements of a, b , c, and d is shown in Fig. 6.

$$(v_i[k]) = \begin{array}{c} \left(\begin{array}{cccc} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{array}\right) \\ \begin{array}{cccc} v_3 & v_2 & v_1 & v_0 \end{array} \end{array}$$

Fig. 6. Digit matrix for Example 2

To calculate $W^{+}_{0,1}$, for example, we count the subexpressions that are obtained by shifting (1 1) in v_0 and v_1. There are two such pairs in v_0 and v_1, one such pair in v_1 and v_2, and two pairs in v_2 and v_3. Thus, $W_{0,1} = 2 + 1 + 2 = 5$. All $W_{i,j}$s are obtained in the same way. In this case, $W^{+}_{0,1}$ is maximal and all the pairs of (1 1) are clustered over columns and rows at a time. We generate three columns v_4, v_5, and v_6 corresponding to the above three kinds of subexpressions, respectively. Hence, the digit matrix is subsequently updated as in Fig. 7 and weights are computed again. Finally, we need four additions and four shifts for the whole computation.

$$\begin{array}{c} \left(\begin{array}{ccccccc} 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{array}\right) \\ \begin{array}{ccccccc} v_6 & v_5 & v_4 & v_3 & v_2 & v_1 & v_0 \end{array} \end{array}$$

Fig. 7. Updated digit matrix

Now, we outline the algorithm below.

Algorithm for Hierarcichal Clustering

1. Represent all constants in some representation form and transform them into the digit matrix $(v_i[k])$ where each element consists of a single digit.
2. Eliminate duplicates of identical constants as well as eliminate constants of less than two non-zero digits.
3. For all $v_i, v_j \in V$, compute the weights $\{W^{\pm}_{i,j}\}$.
4. Until all of the pairs of columns have no positive weights,
   - Select the best pair of digits, *i.e.* find the pair of columns (v_I, v_J) that takes the value $max\ \{W^{\pm}_{i,j}\}$.
   - Update the digit matrix according to the clustering of common subexpressions: let the clustered elements to be zero in the former matrix and decide the elements of the new columns which are produced by the clustering instead.
   - Recompute the weights $\{W^{\pm}_{i,j}\}$ for the updated digit matrix.
5. Output the total number of additions and shifts needed and the final data flow graph.

In the third step, we compute the weights which take into count both the direct common subexpressions and the subexpressions which are obtained by the substitution of shifts for additions. The fourth step is the main recursive procedure of the algorithm. The pairing that can reduce the number of additions most is selected. In case that there are weights with the same highest value, we select the subexpression that does not increase the height of the data-flow graph because balancing the heights of all trees tends to result in the better sharing of the subexpressions while also affecting the throughput.

As for the substitution of shifts for additions, generally, such shifts can be performed between the subexpressions which consist of more than two non-zero digits and the common subexpressions can be fully explored.

IV. Extension to arbitrary linear transforms

The MCM problem can be seen frequently in many problems which include linear transforms such as those in signal and image processing, error-correcting codes, and so forth. Thus, it is beneficial to extend our method to linear transforms with entries of arbitrary values.

The general linear transform has the form:

$$Y_i = \sum_{j=1}^{n} a_{ij} X_j,\ (i = 1,\ \ldots\ , n).$$

Such a case was formerly discussed in [7] in which they applied the iterative pairwise matching algorithm twice. However, we present a natural and effective method to explore the solution space at a single point in time using hierarchical clustering. In the case of matrix multiplication, two kinds of shift-for-addition substitution are taken into account for the weight computation; one is for the constants in the same columns, *i.e.* for those multiplied by the same variable X, and the other is for those in two different columns multiplied by different variables X and X'. In the former case, the pair of digits are in the same column and the weights are computed in the same way as before. To explore the latter case, we first need to define the digit matrix. It can be achieved by regarding each column in the original matrix as a set of multiple columns so that each element consists of just a single digit. Then, we can generalize the definition of the weights as follows:

$$
\begin{aligned}
W^{+}_{i,j} &= \#\{(v_p[k], v_q[k]) = (\pm 1, \pm 1) \mid 0 \leq \exists l \leq m-1 \; s.t. \\
&\qquad X \ll p + X' \ll q = \pm(X \ll i + X' \ll j) \ll l\} \\
W^{-}_{i,j} &= \#\{(v_p[k], v_q[k]) = (\pm 1, \mp 1) \mid 0 \leq \exists l \leq m-1 \; s.t. \\
&\qquad X \ll p - X' \ll q = \pm(X \ll i - X' \ll j) \ll l\}
\end{aligned}
$$

Then, $\{W^{\pm}_{i,j}\}$ within one column is a special case of this definition *s.t.* X and X' being the same variable.

Note that in regard to a linear transform with elements of only $1, -1$, and 0, common subexpressions are limited to those across the rows in the matrix, thus preventing substitution of shifts being performed. The weights and the procedure are almost the same as those used by Mehendale *et al.*[8], although they did not refer to a matrix computation.

We will now state the extended algorithm.

Extended Algorithm for Linear Transforms

1. Represent all of the elements of a matrix in some representation form.
2. Generate the digit matrix by regarding each column of the original matrix as a set of multiple columns so that each element consists of a single digit.
3. Compute the weights $\{W^{\pm}_{i,j}\}$ and apply the clustering algorithm to the digit matrix.

V. Experimental Results

We have implemented the algorithms in C under Unix and applied them to a set of benchmark examples as shown in Table I and II. Some of them were adopted from the examples used in [7]; we used as many examples as we could to compare the effects directly. In the two tables, the rows are devoted to the names of the benchmark examples. The columns "Initial" and "I" show the number of additions initially needed. "HC" is the abbreviation for our hierarchical clustering and the column "HC" shows the number of operations after applying our algorithm. The columns "HC/Initial", "HC/I", and "[7]/I" show the reduction ratios compared with the initial number of operations.

TABLE I
Benchmark Results on several linear transforms

(1) Examples for Linear Codes with binary values

Example	Additions			
	Initial	[7]	HC	HC/Initial
(7,4) Hadamard	21	16	16	0.762
(16,11) Reed-Muller	61	43	31	0.508
(15,7) BCH	74	48	47	0.635
(24,12,8) Golay	76	-	47	0.618

(2) Example for a Linear Transform with 0, 1, -1

Example	Additions/Subtractions			
	Initial	[7]	HC	HC/Initial
Hadamard Matrix H_8	56	24	20	0.357
Hadamard Matrix H_{16}	240	-	64	0.267

(3) Example for a Linear Code wit ternary values

Example	Shifts			Additions		
	I	HC	HC/I	I	HC	HC/I
(12,6,6) Ternary Golay	10	5	0.5	24	20	0.833

More specifically, (1) of Table 1 shows the set of benchmark examples of the representation matrices of some error-correcting codes with binary elements as in [7] and [12]. The Hadamard matrices in (2) were taken as examples with elements of values $1, -1$, and 0. They are often used for image and video compression.

For those matrices with elements of a single digit, our weights coincide with those of Mehendale *et al.*[8] as we note in IV and will be performed in the same way exept for the selection of the subexpressions with identical weight. The average reduction ratio for (1) and (2) is about 0.525 and the numbers of additions/subtactions are less or equal to those in [7]. Especially for the Hadamard matrix H_8, we discover the best digit-pairs in the 3rd and 7th columns from the left and as well as in the 4th and 8th columns from the left by the weights. This results in reducing four more additions than in [7].

Finally, (3) of Table I and Table II are examples of linear transforms with entries of arbitrary values. We compare our results with the method previously proposed for linear transforms. Since the input numbers of shifts and additions are slightly different between [7] and our method, we compared the reduction ratios to be equitable. In these examples, with regard to the number of additions/subtractions, the average reduction ratio in our method is 0.242 and those in [7] is 0.276. Thus, our method achieved better level of reduction. In analyzing the results, in [7], the substitution of shifts were not taken into account and the multistep approach were taken for reducing

TABLE II
RESULTS FOR DCT -DISCRETE COSINE TRANSFORM

Example	# of bits	Shifts					
		I([7])	[7]	[7]/I	I	HC	HC/I
DCT	8	308	72	0.234	280	58	0.207
	12	376	74	0.197	376	75	0.199
	16	529	107	0.202	504	99	0.196
	Ave.	-	-	0.211	-	-	0.201

Example	# of bits	Additions/Subtractions					
		I([7])	[7]	[7]/I	I	HC	HC/I
DCT	8	300	94	0.313	272	73	0.268
	12	368	100	0.272	368	84	0.228
	16	521	129	0.248	496	114	0.230
	Ave.	-	-	0.278	-	-	0.242

the solution space. However, our weights take into account not only the common subexpresisons across the columns of the digit matrix but also those across the rows. Furthermore, we have also searched for the entire solution space at one time, which possibly acounts for the better results in the number of additions/subtractions. Accordingly, more shifts were shared across the rows and columns in our method and the number of shifts were also be more reduced.

VI. CONCLUSION

In this paper, we have proposed an efficient solution for the Multiple Constant Multiplication problem to minimize the number of additions, subtractions and shifts needed for the original multiplications with a constant. First, we transform the problem into a minimization problem of the number of additions+subtractions and then, solve it by hierarchical clustering based on the appropriate weights. Since the weights express operation priorities to be shifted and added(or subtracted), our method can efficiently select the best subexpression to be operated. Furthermore, the advantage of our method is not only in its efficiency and simplicity but also in its extendability to other high-level synthesis tasks which are reduced to linear transforms with arbitrary elements. The experimental results show the effectiveness of our method.

Future work is on applying the method for a resource-constrained case and on implementing DSP applications.

REFERENCES

[1] R. Jain, K. Kucukcakar, M. J. Milnar, and A. C. Parker, “Experience with the ADAM synthesis system,” DAC 1989, pp. 56-61.

[2] J. M. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, “Fast prototyping of datapath-intensive architectures,” IEEE Design Test, pp. 40-51, Jun. 1991.

[3] A. Nicolau and R. Potasman, “Incremental tree height reduction for high level synthesis,” DAC 1991, pp 770-774.

[4] A. Chandrakasan, M. Potkonjak, J. Rabaey, and R. W. Brodersen, “HYPER-LP: A system for power minimization using architectural transformations,” ICCAD-92, pp. 300-303.

[5] A. Chatterjee and R. K. Roy, “An architectural transformation program for optimization of digital systems by multi-level decomposition,” DAC 1993, pp. 343-348.

[6] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, “Efficient substitution of multiple constant multiplications by shifts and additions using iterative pairwise matching,” DAC 1994, pp. 189-194.

[7] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, “Multiple constant ultiplications: Efficient and versatile framework and algorithms for exploring common subexpression elimination,” Trans on CAD, Vol. 15, No. 2, Feb. 1996.

[8] M. Mehendale, S. D. Shelekar, and G. Venkatesh, “Synthesis of Multiplier-less FIR Filters with Minimum Number of Additions,” ICCAD-95, pp. 668-671.

[9] R. Bernstein, “Multiplication by integer constants,” Software - Practice and Experience, Vol. 16, No. 7, pp. 641-652, 1986.

[10] D. J. Magenheimer, L. Peters, K. Pettis, and D. Zuras, “Integer multiplication and division on the HP precision architecture,” ASPLOS II, IEEE Computer Soc. Press, pp. 90-97, 1987.

[11] F. Catoor et al., “SAMURAI: A general and efficient simulated annealing schedule with fully adaptive annealing parameters,” Integration, Vol. 6, pp.

[12] S. Roman, “*Coding and Information Theory*,” Springer-Verlag, Graduate Texts in Math., 1992.