

A Heuristic Approach for Multiple Restricted Multiplication

Nalin Sidahao, George A. Constantinides, and Peter Y. K. Cheung

Department of Electrical & Electronic Engineering,

Imperial College, London, UK,

Email: {nalin.sidahao, g.constantinides, p.cheung}@imperial.ac.uk

Abstract—This paper introduces a heuristic solution to the multiple restricted multiplication (MRM) optimization problem. MRM refers to a situation where a single variable is multiplied by several coefficients which, while not constant, are drawn from a relatively small set of values. The algorithm involves deriving directed acyclic graphs representing multiple constant multiplication obtained for each time step and then merging these graphs into a single MRM graph. For FPGA implementation, the proposed approach results in significant area savings, especially for large problem sizes, and is time-efficient compared to a previous optimum approach using Integer Linear Programming.

I. INTRODUCTION

Many Digital Signal Processing (DSP) or arithmetic-intensive applications involve frequent use of multiplication. The multiplication operation is considered to be expensive, as it consumes significant logic and routing resources for implementation. For constant multiplications, it is common to use a combination of shift-and-add operations rather than using full multipliers, to reduce the hardware usage [1]. Since the shift operation can be implemented by wiring, it has negligible cost in a bit parallel implementation. Therefore, the total hardware cost approximately corresponds to the area of the adders required.

For the problem of multiple constant multiplication (MCM), common sub-expression elimination (CSE) can be applied to a set of constant multipliers in order to minimize the number of additions required [2], [3].

This paper introduces a heuristic approach to a related problem, the recently proposed multiple restricted multiplication (MRM) optimization [4]. MRM refers to a situation where a single variable is multiplied by several coefficients that, while not constant, are drawn from a finite set of constants that change with time. Such a situation arises commonly in high level synthesis tasks due to resource sharing, for example in a folded implementation of a FIR filter [5] or polynomial evaluation [6], [7].

In summary, the main contributions of this paper are:

- A novel heuristic for the multiple restricted multiplication problem.
- FPGA implementation comparisons between the proposed approach, a standard approach, and the optimal approach previously published [4].

The remainder of this paper is structured as follows. Section 2 of introduces the proposed heuristic, Section 3 presents the

results compared both a standard alternative implementation, and a previous optimal approach, and Section 4 concludes the paper.

II. PROPOSED HEURISTIC

Existing synthesis approaches to CSE are unable to take advantage of the MRM situation, resulting in the use of expensive general multipliers, as shown in Fig. 1. Fig. 1(a) shows a Data Flow Graph (DFG) containing two multipliers with a common input x , and sets of constant multiplicands labelled as $\{c_{11}, c_{12}, \dots, c_{1T}\}$ and $\{c_{21}, c_{22}, \dots, c_{2T}\}$. The first subscript here refers to the spatial index and the second to the time index, *i.e.* c_{it} is the value of multiplicand i at time t . A standard implementation using ROMs and multipliers is depicted in Fig. 1(b).

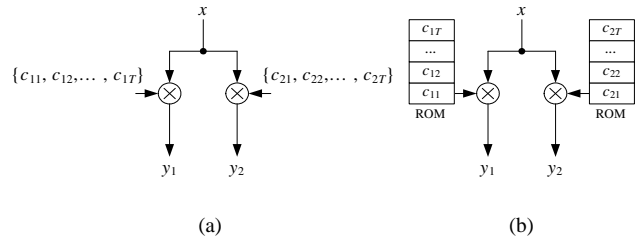


Fig. 1. (a) The DFG of a simple MRM problem. (b) The standard implementation with ROMs and multipliers.

In [4], it was shown that the MRM problem can be addressed through extending the basic unit of operation from an addition, used in MCM, to an adder/multiplexer combination shown in Fig. 2(a). It was further demonstrated that for Xilinx-based implementations, the Xilinx Virtex / Virtex-II slice architecture [8] can be used to implement such a basic computational unit with no area overhead compared to the equivalent adder used in MCM [9].

As an example, assume we have a single variable input x multiplied by two sets of coefficients $\{165, 132, 32\}$ and $\{40, 32, 8\}$. An optimized implementation of this MRM problem can be seen in Fig. 2(b), and is described below.

Fig. 2(b) is recognizable as a standard MCM solution, containing two addition nodes [10], and generating the two values $165x$ and $40x$. Figs. 2(c) and (d), illustrate how the same DFG structure, can be used to obtain the remaining

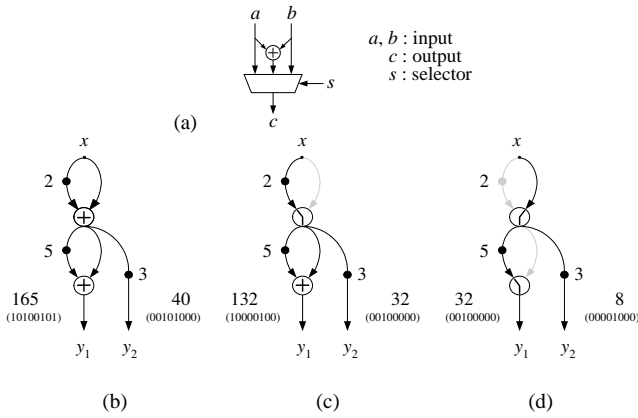


Fig. 2. (a) An adder/multiplexer node, and configurations to create the coefficient sets $\{165, 132, 32\}$ and $\{40, 32, 8\}$ in (b) 1st time step, (c) 2nd time step, and (d) 3rd time step. (Black dots represent left-shift operations)

coefficients, by selecting the behaviour of the nodes from the possibilities shown in Fig. 2(a).

In general, we may encode an instance of the MRM problem as a $T \times C$ matrix, where T is the number of rows corresponding to time steps and C is number of columns representing distinct outputs. For example, Fig. 2 is a 3×2 problem.

The question arises: how many computational nodes are required for a given MRM problem? The authors have previously presented an approach to find solutions to the minimum-cost of this optimization problem by formulating it as an Integer Linear Program (ILP) [4]. However the solution time required for this ILP increases rapidly with the problem size, and in practice this approach is unsuitable for solving MRM problems with more than three time steps or three outputs. This is the motivation for the heuristic presented in this paper.

The proposed methodology involves the merging of several directed acyclic graphs (DAGs), one per time-step, where each node represents an addition operation, into a single MRM graph.

A. Generating MCM Graphs

The first step of the algorithm is therefore to generate the graphs for each time-step. This is multiple constant multiplication (MCM) problem, for which there are many known approaches. Our synthesis system follows the algorithm proposed in [10], summarized as follows for convenience. 1. Reduce all coefficients to their equivalent odd ‘fundamental’, by repeated division by two. 2. Remove repeated fundamentals and the unit fundamental, as these does not incur implementation cost. 3. Create a graph with the remaining cost-1 fundamentals as nodes. 4. Try to form remaining fundamentals through pairwise sums of form $x + 2^k x$ or $x + 2^k y$, where x and y are existing nodes, and k is a positive integer. 5. Repeat (4) until no more possible. In case this procedure does not produce all required coefficients, it is required to introduce additional nodes and repeat (see [10] for details). As a simple example, assume we have a 3×3 problem

$$\begin{pmatrix} 16 & 23 & 35 \\ 28 & 11 & 14 \\ 28 & 34 & 33 \end{pmatrix}.$$

The MCM graphs for each time step are depicted in Fig. 3.

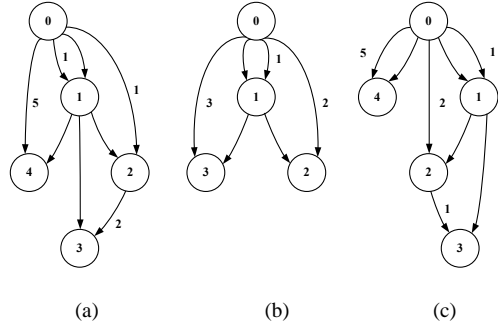


Fig. 3. The MCM graphs for (a) $\{16, 23, 35\}$. (b) $\{28, 11, 14\}$. (c) $\{28, 34, 33\}$.

B. Graph Merging

The novelty of the proposed approach comes in the merging of these MCM graphs to form a single implementation structure for the MRM problem. One can imagine that finding an MCM graph for a particular time step within an MRM graph is similar to subgraph isomorphism [11], which involves mapping of a graph into a subgraph of another having the same structure.

However the proposed process is different in that a *path* of the MRM graph can be mapped onto an edge of the other graph, in such a way that sum of the weight of edges in the path is equal to the weight of that edge. This path corresponds to a path through multiplexers in the MRM implementation.

The process of merging is shown in Algorithm Graph Mapping, but before describing the algorithm, it is instructive to consider an example.

An example of merging the MCM graphs in Fig. 3 is illustrated in Fig. 4 and Fig. 5. The procedure starts by choosing one of the MCM graphs as the core MRM graph, which is then expanded as necessary. Let us take Fig. 4(a) as the MRM graph L , and and 4(b), as the MCM graph S . These figures illustrate how nodes 1 and 3 of each graph can be matched together with the corresponding node 1 and 3 of the other graph. Consider node 3, where the dashed line in Fig. 4 represents the path $\langle 3, 2, 0 \rangle$, which has the same weight as the edge $\langle 3, 0 \rangle$ in Fig. 4. Similarly, the dotted path represents another mapping, between the path $\langle 3, 1 \rangle$ and the edge $\langle 3, 1 \rangle$. The result of graph merging is depicted in Fig. 4(c), where an extra node 5 has been introduced to account for the unmapped portion of Fig. 4(b), *i.e.* node 2.

The process is then repeated, trying to add the MCM graph of Fig. 5(b) to the result of the previous phase, illustrated again in Fig. 5(a). Nodes 1 and 2 in Fig. 5(b) can be matched to nodes 1 and 5, respectively, in Fig. 5(a). The final minimized MRM graph is shown in Fig. 5(c).

Algorithm Graph Mapping, which performs this process automatically, consists of 5 recursive subroutines: Main_Map,

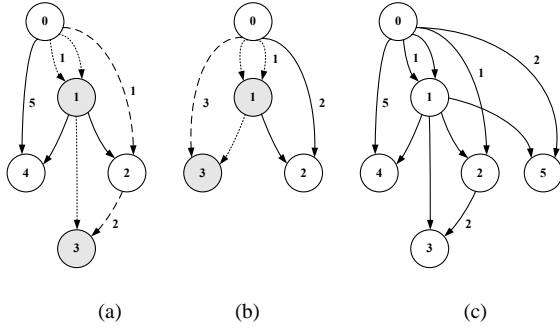


Fig. 4. Graph merging (a) the initial MRM graph, (b) the MCM graph to be added, and (c) the result of merging.

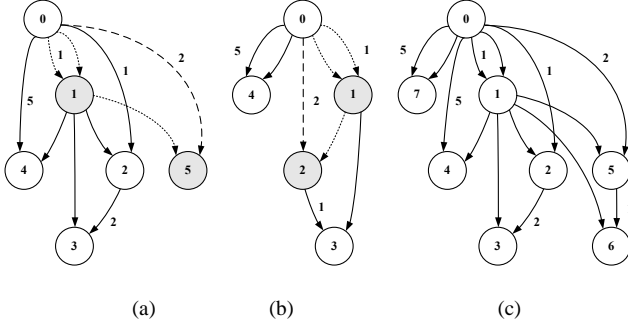


Fig. 5. Further graph merging (a) the MRM graph resulting from the previous merging process, (b) the MCM graph to be added, and (c) the result of merging.

Map_Inedge, Check_Start_Nodes, Check_Weights, and Check_Zero_Weight. The algorithm takes as input graphs L and S , together with the source node s_0 and l_0 of L and S , respectively. It then proceeds to systematically search the edges of L from every computational node l back to l_0 , and l is said to be ‘matched’ to a node s when there is a set of paths in the graph L , for which each element can be mapped an edge in the path from s_0 to s .

In algorithm Graph Mapping below, if $\text{Main_Map}(s, l)$ evaluates to TRUE, then we record that node l can be mapped to node s by setting $\text{mark_node_match}[s][l]$. Main_Map tries to build a correspondence between two nodes l and s in the MRM and MCM graphs, respectively. Algorithm Map_Inedge performs mapping a path l to an edge s , the central difference between this procedure and one for subgraph isomorphism. Subroutine Check_Start_Nodes checks whether each node s and l has reached the graph source nodes s_0 and l_0 . Algorithm Check_Weights is called whenever both edges of node l and s has its same predecessor, in order to ensure that the path weight is correct. Finally, Algorithm Check_Zero_Weight searches an edge of l that has zero weight and then consider its predecessor with s by calling Main_Map algorithm.

Algorithm Graph Mapping

Input: Two Computational MCM Graphs L and S

Output: An MRM Graph

begin

mark_node_match[l][s] \leftarrow FALSE for all $l \in L, l \neq l_0, s \in S, s \neq s_0$

foreach $s \in S$ **do**

foreach $l \in L$ **do**

if $\text{Main_Map}(l, s)$ **do**

 mark_node_match[l][s] \leftarrow TRUE
 end if
 end foreach
end

Main_Map

Input: Two Computational Nodes l and s

Output: TRUE if node l can be mapped to s , FALSE otherwise

begin

if $\text{pred}_1(l) \neq \text{pred}_2(l)$ && $\text{pred}_1(s) \neq \text{pred}_2(s)$ **do**
 return $\text{Map_Inedge}(\text{inedge}_1(l), \text{inedge}_1(s), w(\text{inedge}_1(s)))$
 && $\text{Map_Inedge}(\text{inedge}_2(l), \text{inedge}_2(s), w(\text{inedge}_2(s)))$
 || $(\text{Map_Inedge}(\text{inedge}_1(l), \text{inedge}_2(s), w(\text{inedge}_2(s)))$
 && $\text{Map_Inedge}(\text{inedge}_2(l), \text{inedge}_1(s), w(\text{inedge}_1(s))))$
else if $\text{pred}_1(l) \neq \text{pred}_2(l)$ && $\text{pred}_1(s) = \text{pred}_2(s)$ **do**
 return $\text{Check_Zero_Weight}(w(\text{inedge}_1(l)), w(\text{inedge}_2(l)))$
else if $\text{pred}_1(l) = \text{pred}_2(l)$ && $\text{pred}_1(s) = \text{pred}_2(s)$ **do**
 return $\text{Check_Weights}(\text{inedge}_1(l), \text{inedge}_2(l), \text{inedge}_1(s), \text{inedge}_2(s))$
else
 return FALSE
end if
end

Map_Inedge

Input: $\text{inedge}_x(l), \text{inedge}_x(s), w_x$

Output: TRUE if $\text{inedge}_x(l)$ can be mapped to $\text{inedge}_x(s)$, FALSE otherwise

begin

$w_{dif} = w_x - w(\text{inedge}_x(l))$

if $w_{dif} = 0$ **do**

return $\text{Check_Start_Nodes}(\text{inedge}_x(l), \text{inedge}_x(s))$

else if $w_{dif} > 0$ **do**

if $\text{pred}_x(l) \neq l_0$ **do**

return $\text{Map_Inedge}(\text{inedge}_x(l), \text{inedge}_1(\text{pred}_x(s)), w_{dif})$
 || $\text{Map_Inedge}(\text{inedge}_x(l), \text{inedge}_2(\text{pred}_x(s)), w_{dif})$

else

return FALSE

end if

else

return FALSE

end if

end

Check_Start_Nodes

Input: $\text{inedge}_x(l), \text{inedge}_x(s)$

Output: TRUE if $\text{inedge}_x(l)$ can be mapped to $\text{inedge}_x(s)$,

or both $\text{pred}_x(l)$ and $\text{pred}_x(s)$ reach their start nodes, FALSE otherwise

if $\text{pred}_x(l) \neq l_0$ && $\text{pred}_x(s) \neq s_0$ **do**

return $\text{Main_Map}(\text{pred}_x(s), \text{pred}_x(l))$

else if $\text{pred}_x(l) \neq l_0$ && $\text{pred}_x(s) = s_0$ **do**

return $\text{Map_Inedge}(\text{inedge}_x(s), \text{inedge}_1(\text{pred}_x(l)), 0)$
 || $\text{Map_Inedge}(\text{inedge}_x(s), \text{inedge}_2(\text{pred}_x(l)), 0)$

else if $\text{pred}_x(l) = l_0$ && $\text{pred}_x(s) = s_0$ **do**

return TRUE

else

return FALSE

end if

end

Check_Weights

Input: $\text{inedge}_1(l), \text{inedge}_2(l), \text{inedge}_1(s), \text{inedge}_2(s)$

Output: TRUE if weight edges can be mapped and Check_Start_Nodes returns TRUE, FALSE otherwise

begin

if $(w(\text{inedge}_1(l)) = w(\text{inedge}_1(s)) \ \&\& \ w(\text{inedge}_2(l)) = w(\text{inedge}_2(s)))$

 || $(w(\text{inedge}_1(l)) = w(\text{inedge}_2(s)) \ \&\& \ w(\text{inedge}_2(l)) = w(\text{inedge}_1(s)))$ **do**

return $\text{Check_Start_Nodes}(\text{inedge}_1(l), \text{inedge}_1(s))$

else

return FALSE

end if

end

Check_Zero_Weight

Input: $w(\text{inedge}_1(l)), w(\text{inedge}_2(l))$

Output: TRUE if there is a zero weight edge of l and Main_Map returns TRUE, FALSE otherwise

begin

if $\text{pred}_1(l) \neq l_0$ **do**

if $w(\text{inedge}_1(l)) = 0$ **return** $\text{Main_Map}(s, \text{pred}_1(l))$

else if $w(\text{inedge}_2(l)) = 0$ **return** $\text{Main_Map}(s, \text{pred}_2(l))$

else return FALSE

else

return FALSE

end if

end

Sometimes there is more than one node in L that can be matched to a node in S . In order to minimize the latency of the structure when implemented, our algorithm selects the l that has shortest unweighted path from the source l_0 to l . Graph merging is terminated when all graphs have been processed.

III. IMPLEMENTATION RESULTS

The area results for a Xilinx Virtex II [8] implementation are compared with a standard approach using ROMs and general multipliers. Experimental problems of sizes between 3×3 and 3×40 , and with coefficients of 4, 5 and 6 bits are presented in this paper. Although larger problem sizes can easily be solved, this size is appropriate for comparison with the optimal approach in [4], and it should be noted that savings over the ROM/multiply implementation grow with the problem size, as seen from Fig. 6. The results collected show area reductions of up to 37% over the ROM/multiply implementation, but larger problems will result in larger savings. Over the set of results that can also be addressed by the optimal approach in [4], the heuristic area is between 25% and 79% greater than the optimal value. This loss in area performance is because the structure of the MCM graphs is fixed, without reference to coefficient values in other time slots.

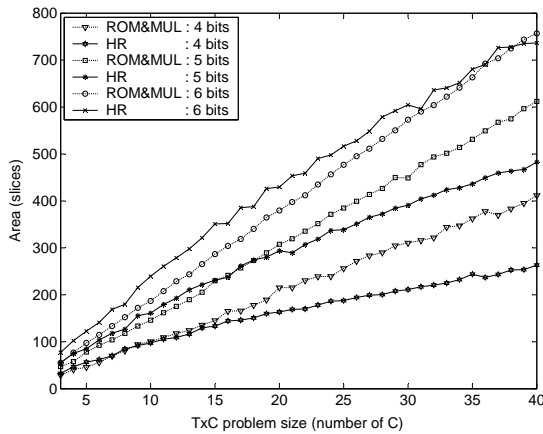


Fig. 6. Average area versus the problem size.

The execution time of small 4-bit problems of the proposed procedure to the optimal ILP from [4] is shown in Table I. For the biggest problem (3×3) that can be reasonably computed using ILP, it takes 61:34 hrs/mins, whereas the proposed approach uses 2.84 secs. This reduction in time allows the proposed procedure to be embedded within a high-level synthesis flow.

IV. CONCLUSION

The work presented in this paper can be considered as an extension of [4] which a novel heuristic algorithm has been proposed. It is based on merging graphs which involves partial mapping directed acyclic graphs representing multiple constant multiplication obtained from each set of time-step coefficient.

This paper seeks to provide a framework for future research on the problem of multiple restricted multiplication for FPGA

TABLE I

VARIATION OF EXECUTION TIME FOR HEURISTIC AND ILP APPROACH
The upper figure is heuristic result and the lower figure is ILP result
(in hrs:mins:secs)

Problem Size		C		
		1	2	3
T	1	00:00:1.33, 00:00:0.46	00:00:1.40, 00:00:0.54	00:00:1.39, 00:00:0.97
	2	00:00:1.96, 00:00:5.47	00:00:2.04, 00:00:11.23	00:00:2.04, 06:00:33.52
	3	00:00:2.70, 00:01:20.74	00:00:2.73, 01:52:19.49	00:00:2.84, 61:34:48.30

implementation. There are many ways in which the heuristic approach presented in this paper could be taken forward.

The algorithm for the MRM problem on which we have concentrated has dealt with the operation of addition. It will be simple in the future work to incorporate subtraction into our algorithm. Indeed, each node of MRM graph could be different operators, for instance, adder-multiplexer (in the case of this paper), adder-subtractor and subtractor-multiplexer. Future research could also develop FPGA hardware implementation for the different types of node. Some interesting steps in this direction have recently been made by Turner [12]. Since delay is not explicitly targeted, it may be useful in incorporating some function to improve the path delay.

REFERENCES

- [1] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. New York: Oxford University Press, 2000.
- [2] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, "Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, pp. 151–165, February 1996.
- [3] M. Chen, J. Y. Jou, and H. M. Lin, "An efficient algorithm for the multiple constant multiplication problem," in *Proc. Int'l Symp. VLSI Technology, Systems, and Applications*, 1999, pp. 119–122.
- [4] N. Sidahao, G. A. Constantinides, and P. Y. K. Cheung, "Multiple restricted multiplication," in *Proc. 14th Int'l Conf. on Field Programmable Logic and Application. FPL'04*, 2004, pp. 374–383.
- [5] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. New York: John Wiley & Sons, 1999.
- [6] G. Corbaz, J. Duprat, B. Hochet, and J.-M. Muller, "Implementation of a VLSI polynomial evaluator for real-time applications," in *Proc. Int'l Conf. Application Specific Array Processors*, 1991, pp. 13–24.
- [7] J. Villalba, G. Bandera, M. A. Gonzalez, J. Hormigo, and E. L. Zapata, "Polynomial evaluation on multimedia processors," in *Proc. Int'l Conf. Application-specific Systems, Architectures and Processors (ASAP 2002)*, San Jose, California, 2002.
- [8] Xilinx, Inc., "Xilinx Documentation and Literature," <http://www.xilinx.com/literature>; accessed 25 September 2004.
- [9] R. H. Turner, T. Courtney, and R. Woods, "Implementation of fixed DSP functions using the reduced coefficient multiplier," in *Proc. Int'l Conf. Acoustics, Speech, and Signal Processing*, 2001, pp. 881–884.
- [10] A. G. Dempster and Macleod M. D., "Constant integer multiplication using minimum adders," in *IEE Proc. Circuits, Devices and Systems*, vol. 141, October 1994, pp. 407–413.
- [11] G. Valiente, *Algorithms on Trees and Graphs*. Berlin: Springer-Verlag, 2002.
- [12] R. H. Turner, "Functionally diverse programmable logic implementations of digital signal processing algorithms," Ph.D. dissertation, Queen's University of Belfast, August 2002.