Using Transposition to Efficiently Solve Constant Matrix-Vector Multiplication and Sum of Product Problems

Narges Mohammadi Sarband¹ · Oscar Gustafsson¹ · Mario Garrido¹

Received: 30 April 2019 / Revised: 30 December 2019 / Accepted: 26 May 2020 / Published online: 9 July 2020
 © The Author(s) 2020

Abstract

In this work, we present an approach to alleviate the potential benefit of adder graph algorithms by solving the transposed form of the problem and then transposing the solution. The key contribution is a systematic way to obtain the transposed realization with a minimum number of cascaded adders subject to the input realization. In this way, wide and low constant matrix multiplication problems, with sum of products as a special case, which are normally exceptionally time consuming to solve using adder graph algorithms, can be solved by first transposing the matrix and then transposing the solution. Examples show that while the relation between the adder depth of the solution to the transposed problem and the original problem is not straightforward, there are many cases where the reduction in adder cost will more than compensate for the potential increase in adder depth and result in implementations with reduced power consumption compared to using sub-expression sharing algorithms, which can both solve the original problem directly in reasonable time and guarantee a minimum adder depth.

Keywords Constant matrix multiplication (CMM) · Multiple constant multiplication (MCM) · Shift-and-add · Sum of products (SOP) · Minimum depth expansion algorithm.

1 Introduction

In many applications, primarily within the field of digital signal processing (DSP), computations are performed with constant coefficient multiplication. Hence, one may replace the general multiplier with a network of adders, subtracters, and shifts [1, 2]. The computations may have a different number of inputs and outputs and allow to reduce the number of adders and subtracters by utilizing common partial results.

The general case, supporting multiple inputs and outputs is constant matrix multiplication (CMM). This is the multiplication of a $M \times N$ constant matrix C , by a column vector I of dimension N that results a column vector O of dimension M , in which M is the number of rows and N is the

number of columns of the constant matrix C , $O = CI$ or on element form

$$\begin{bmatrix} O_1 \\ O_2 \\ \vdots \\ O_M \end{bmatrix} = \begin{bmatrix} C_{1,1} & C_{1,2} & \dots & C_{1,N} \\ C_{2,1} & C_{2,2} & \dots & C_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ C_{M,1} & C_{M,2} & \dots & C_{M,N} \end{bmatrix} \times \begin{bmatrix} I_1 \\ I_2 \\ \vdots \\ I_N \end{bmatrix} \quad (1)$$

The CMM problem is defined as finding a solution using adders, subtracters, and shifts that realizes the computation using a few adders and subtracters as possible [1–7]. As adders and subtracters have about the same complexity, we will from here refer to both as adders, and the number of adders as the adder cost.

When the number of columns (and therefore the number of inputs) of the constant matrix C decreases to 1 it becomes a multiple constant multiplication (MCM) problem, where a single input is multiplied with multiple constant coefficients [8–10]. Similarly, when the number of rows (number of outputs) decreases to 1, it becomes a sum of products (SOP) computation. Finally, when both the number of rows and columns decrease to one, it becomes a single constant multiplication problem (SCM) [11–13]. As the SCM problem can be solved optimally using pre-computation, we will not consider that anymore here.

✉ Narges Mohammadi Sarband
 narges.mohammadi.sarband@liu.se

Oscar Gustafsson
 oscar.gustafsson@liu.se

Mario Garrido
 mario.garrido.galvez@liu.se

¹ Department of Electrical Engineering, Linköping University, Linköping, Sweden

Figure 1 Average required time for the RPAG algorithm [7] to create (a) an MCM or SOP adder graph and (b) a CMM adder graph for matrices with dimension $M \times N$.

Instead, we will primarily consider the CMM problem as it is a generalization of the MCM and the SOP problems.

The general problem of finding a minimum adder cost solution is an NP hard problem [14]. Although optimal approaches have been suggested [14–16], the majority of the algorithms suggested are heuristics, although there are both rules on when the solutions are optimal as well as lower bounds that can be used to prove optimality [1]. There have been two major classes of algorithms suggested to solve this type of problems: adder graph algorithms [3, 7, 8, 12, 17] and sub-expression sharing algorithms [4, 5, 9, 10]. In general, adder graph algorithms provide a lower adder cost since they are not limited by a specific number representation. Instead, they focus on building up a search space of possible values to realize from the one already realized. This works well for MCM problems, where it is rather likely that the different coefficients can be computed as a simple shift-and-add of other coefficients. In fact, since the CMM problem is, from an adder graph perspective, a question of finding intermediate results that are not needed as an output, the more outputs, i.e. coefficients there is, the easier the problem becomes. The intermediate results not needed as an output as often referred to as non-output fundamentals.

However, adder graph algorithms do also take a significantly longer time when solving hard problems, i.e. problems where many intermediate results must be determined, such in the case of an SOP computation. Hence, although the sub-expression sharing algorithms may not be able to provide an optimal solution, the adder graph algorithms may in some cases not be able to provide a solution at all within a reasonable time limit. Therefore, adder graph algorithms are better at providing solutions when they work well, for example for MCM problems, but as the complexity grows fast with the number of inputs, they will become impractical for problems with many inputs. This is the main observation motivating the current work: by transposing the problem matrix, solving using an adder graph algorithm, and transposing the resulting solution back, we can solve certain types of problems using more efficient algorithms.

To support this observation we have provided the time required to solve the different variants of the problems using the algorithm from [7]. In Fig. 1a, the time to solve the SOP and the corresponding MCM problem, i.e., the CMM

problem for an $N \times 1$ and a $1 \times N$ matrix, respectively, is shown. It can be seen that already at six input SOP problem takes about 1000 seconds on average, while the transposed MCM problem takes less than a second, although the absolute values are not really the relevant aspect here. Clearly, the gap increases with the number of columns of the SOP problem. Similarly, in Fig. 1b, the time for solving a CMM problem using the same algorithm is shown. We see a small increase with the number of rows (although when the number of rows increases further, we may eventually expect a decrease), but more importantly, there is an increase of one to two orders of magnitude in the number of columns. Hence, we can conclude that for certain problem sizes, there will be a prohibiting¹ solution time, although the transposed version of the problem can be readily solved.

In addition to the adder cost, the number of cascaded adders, the adder depth, is also of interest. Partly because of the operating frequency of the resulting circuit, but primarily because of the increased power consumption caused by an increased amount of glitches when more adders are cascaded. It can be shown to be a relation between the adder cost and adder depth, and, more specifically, the only way to reduce the depth of a given solution is to either replace the non-output fundamentals or add new ones.

Although, as will be illustrated later, we have yet not been able to find a relation between the adder depth of the original and the transposed solution, we still apply the transposition in such a way that the depth is minimize subject to the input solution. It should be noted that it is well known that transposition can be used to obtain e.g. an SOP solution from an MCM solution. However, to the best knowledge of the authors, there has been no earlier work that shows how to perform the transposition in practice, less considers the adder depth while doing it. A preliminary version of this work was introduced in [18] for the SOP problem only.

In this work, we consider two-input adders only, although a similar approach should be applicable to e.g. carry-save adders [19–21] or ternary adders [22, 23].

¹If the solution time is prohibiting clearly depends on the situation. For example, taking a week may not be an issue if it is a one time optimization. However, if it takes several months to solve the problem, in most cases it will be prohibiting. In addition, adding another column will increase it with about an order of magnitude.

2 Adder Graphs and Transposed Adder Graphs

Adder graph algorithms use a directed acyclic graph (DAG), where each node (or vertex), except for the input nodes, represents an addition, and the output of the addition is called the fundamental of that node. Each edge has a weight that represents a shift and possibly a negation. Formally, the DAG is represented by the two sets V and E , for vertices (nodes) and edges, respectively. In a standard adder graph, all nodes except for the input nodes have exactly two incoming edges.

There are some modifications to the standard adder graph that is worth mentioning. First, in [12], the concept of vertex reduced adder graphs was introduced. The idea is that if a node has a fan-out of one, i.e., only one outgoing edge, the order of that node and the node connected at the end of the outgoing edge can change place arbitrarily. Hence, both can be merged to a single node, and the exact order determined later. This will be used here as part of the transposition process. Second, most algorithms for solving MCM and CMM problems normalize the node values, both with respect to shifts and signs, so that there will be only one node with a fundamental of, say, 3, despite that the MCM problem states a multiplication by 3, -3 , and 6. As we do not want to miss the information about the shifts and sign differences once transposing, we here introduce the concept of a complete adder graph. A complete adder graph have explicit output nodes, where each output node has only one incoming edge, corresponding to the shift and possible negation from the normalized fundamental.

In this work, we denote input nodes I_i and output nodes O_j and draw them with a rectangle. Adder nodes are denoted A_k and drawn with a circle. Figure 2 illustrates this concept.

To transpose an adder graph, the transposition theory [25] is used. Transposition of a signal flow graph reverses the direction of all signals. In addition, inputs are interchanged with outputs and adders are interchanged with branches [24–26]. This is illustrated in Fig. 3 for one adder node and its incoming and outgoing edges. To further clarify it, we have illustrated the branch explicitly, while in the rest of the paper this is just shown as multiple outgoing edges.

Although, the transposition determines which branches are added to each other, it does not provide any information about the order in which those additions are carried out when more than two branches converge in the same adder. Therefore, a new graph resulting from transposing an adder graph would be a directed graph but not necessarily an adder graph with two-input adders, because it may have some nodes that have more than two incoming edges which represent multiple-input adders. Therefore, it is similar to the vertex reduced adder graph in [12].

From a signal processing perspective, the transposed network will compute the transposed version of the matrix-vector

Figure 2 The value of one internal node, which is corresponding to an adder node, is calculated based on two other vertices value, which are connected to that adder node through two weighted edges. The value of an output is, weight times of its connected vertex value.

multiplication of the original network. Hence, an MCM solution can be transposed to an SOP solution, etc. Assuming that an $M \times N$ matrix C has an adder cost Adders_C , the adder cost for the transposed $N \times M$ matrix C^T is [27]

$$\text{Adders}_{C^T} = \text{Adders}_C + N - M. \quad (2)$$

Hence, the difference in adder cost is the same as the difference in number of inputs and outputs. This leads to the conclusion that if we can determine a solution with low adder cost for C , it can be used to obtain a low adder cost solution for C^T and vice versa.

3 Proposed Approach

The basic idea behind the proposed algorithm is that adder graph algorithms are more efficient for narrow matrices, i.e., when the matrix has more rows than columns, but are not so efficient for wide matrices, i.e., when the matrix has more columns than rows as earlier discussed and illustrated in Fig. 1. In the latter case, the proposed approach provides an alternative based on transposing the matrix C with dimension $M \times N$ to obtain the matrix C^T with dimension $N \times M$. If $M < N$, solving the CMM problem for C^T

Figure 3 Transposition of an adder node. Inputs to the adder node are converted into branches and outputs of the adder node are converted into an adder.

Figure 4 Transposing one-input- M -outputs MCM adder graph, the solution of $M \times 1$ constant matrix $\mathbf{C}$ yields a N -inputs-one-output SOP solution for matrix $\mathbf{C}^T = [C_{1,1} \ C_{1,2} \ \dots \ C_{1,N}]$, with $N = M$.

will be much faster than for $\mathbf{C}$. The obtained realization for $\mathbf{C}^T$ is then transposed to obtain a solution for $\mathbf{C}$. The relation of matrices and realizations is shown in Fig. 4 for the SOP case. Figure 5 shows the steps of the proposed algorithm and the intermediate results after each step. The steps are explained in detail in Sections 3.1, 3.2 and 3.3. Throughout this section, an example solving the constant matrix $\mathbf{C} = [7 \ 14 \ 8 \ -8 \ 4]$, i.e., a sum of products, is used to illustrate the approach.

3.1 Generating a Complete Adder Graph

As shown in the flowchart, given a constant matrix $\mathbf{C}$ that defines a wide CMM or an SOP problem, can be solved by transposing the input matrix. This transposition results a transposed matrix $\mathbf{C}^T$. Next, an adder graph algorithm is applied on the transposed matrix $\mathbf{C}^T$, therefore, the CMM or MCM algorithm produces a primary adder graph for $\mathbf{C}^T$. As described earlier, CMM/MCM adder graph algorithms produce only normalized fundamental nodes and since we do not want to miss any inputs in the transposing process, the proposed algorithm includes the step, generate complete adder graph. For example, to produce an SOP adder graph, all constant matrix members must be considered, so that they are multiplied by their corresponding inputs and if the MCM adder graph does not produce all outputs, the result of SOP would be incorrect, due to the absence of some inputs in the transposing step, therefore, it is necessary to add a step that complete the graph. To generate a complete adder. To generate a complete adder graph, the proposed algorithm checks the primary adder graph and if there are outputs that are not declared in the primary adder graph, this step adds new nodes (outputs) to the primary adder graph to cover all outputs and make it complete. New added outputs can be obtained by shifting or changing the sign of other normalized fundamental nodes. This process can also be performed for repeated outputs to cover different outputs with the same value. In the proposed example with

Figure 5 Proposed steps for producing an adder graph for constant matrix $\mathbf{C}$, by solving the corresponding CMM/MCM problems for transposed matrix $\mathbf{C}^T$, transposition and minimum depth expansion algorithm.

constant matrix $\mathbf{C} = [7 \ 14 \ 8 \ -8 \ 4]$, the MCM algorithm first produces the primary adder graph, which is shown in Fig. 6a. Then, the complete adder graph in Fig. 6b is obtained from the primary adder graph by adding all of the outputs not included in the Primary one.

3.2 Transposing the Complete MCM Adder Graph

Once the complete adder graph for $\mathbf{C}^T$ has been produced, the next is to apply transposition on it, in the way that

Figure 6 (a) Primary adder graph and (b) complete adder graph for the constant matrix $\mathbf{C} = [7 \ 14 \ 8 \ -8 \ 4]$.

Figure 7 Vertex reduced adder graph created by transposing the complete adder graph in Fig. 6b, as discussed in Section 3.2.

is explained in Section 2. The result is a vertex reduced adder graph for **C** which contains all relation of shift-and-add structure of adder graph for **C**. However, this vertex reduced CMM adder graph for **C**, may contain multiple-input adders, because it may have more than two incoming edges to some vertices as previously described. To convert multiple-input adders of a vertex reduced adder graph into two-input adders, a minimum depth expansion algorithm is used. This algorithm is described next in Section 3.3. In the proposed algorithm if the MCM adder graph algorithm is used, the transposing step will produce the SOP network. Because applying transposition on a single-input- N -output MCM network produces an N -inputs-single-output SOP network. Transposing the example in Fig. 6b, creates the vertex reduced adder graph shown in Fig. 7.

3.3 Minimum Depth Expansion

Algorithm 1 describes the proposed minimum depth expansion algorithm used to transform a vertex reduced adder graph into an adder graph with two-input adders and minimum depth. The algorithm changes a K -input adder in a vertex reduced adder graph to $K - 1$ two-input adders. This conversion uses a method that produces the minimum depth of the resulting adder graph for **C**, subject to the CMM/MCM solution given for **C**^T. Transposition theory represents, which branches, or inputs of a complete adder graph for **C**^T should be added together to result an adder graph for **C**. Therefore, depth of the resulted adder graph for **C** is not dependent on transposition theory and it is related to the method which is used in changing, multiple-input adders of a vertex reduced adder graph to two-input adders which produces the final adder graph for **C**. When all K branches

that must be added are ready at the same time, the minimum depth for adding them is equal to $\lceil \log_2 K \rceil$. In a vertex reduced adder graph, for all adder nodes, there are K connected incoming edges which should be added together, but their availability depth is different. The only available vertices from beginning are input vertices and the rest of connected incoming edges to an adder node would be ready in different time, after some calculation. Therefore, to achieve the minimum depth for addition of some edges together with different depth, the minimum depth expansion algorithm is offered which is described in the Algorithm 1.

Algorithm 1 Minimum depth expansion algorithm.

Input: VMG = (V , E)

1: $V = \{I_1, I_2, \dots, I_N, A_1, A_2, \dots, A_k, O_1, O_2, \dots, O_M\}$
 $E = \{(a_1, b_1, w_1), (a_2, b_2, w_2), \dots, (a_d, b_d, w_d)\}$
 $(a_j, b_j) \subset (V \times V)$
 w_j : shift between a_j and b_j

Output: CMM adder graph = (V' , E')

Initialisation

2: $V' \leftarrow V$
3: $D \leftarrow \{(I_1, 0), \dots, (I_N, 0), (O_1, -1), \dots, (O_M, -1), (A_1, -1), \dots, (A_k, -1)\}$
4: $B \leftarrow \{A_1, \dots, A_k, O_1, \dots, O_n\}$ $\triangleright$ adders to be converted
5: $z \leftarrow 0$

Converting multiple-input adders to two-input adders

6: **while** $|B| \neq \emptyset$ **do** {adders remaining}
7: Find $b_i \in B \mid \exists S \leftarrow \{\forall l(a_l, b_i, w_{li})\} \Rightarrow \forall l, \exists(a_l, D(a_l)) \in D \vee D(a_l) \neq -1$ $\triangleright$ Find an adder (node) where its inputs (edges) are ready
8: **while** $|S| \neq \emptyset$ **do** {inputs remaining}
9: Sort S according to depth
10: $S \leftarrow \{(a_l, b_i, w_{li}), (a_{l+1}, b_i, w_{l+1,i}), \dots\}$
 $\Rightarrow \forall l D(a_l) \leq D(a_{l+1})$
11: $S \leftarrow S \setminus \{s_1, s_2\}$ $\triangleright$ remove s_1, s_2 from S
12: **if** ($|S| = \emptyset$) **then** { s_1, s_2 are last two edges }
13: $V' \leftarrow V' \cup \{a_1, a_2\}$
14: $E' = E' \cup \{s_1, s_2\}$
15: $D(b_i) = \max(D(a_1), D(a_2)) + 1$
16: $D = D \cup \{(b_i, D(b_i))\}$
17: **else**
18: $z \leftarrow z + 1$
19: $V' \leftarrow V' \cup \{A_{k+z}\}$ $\triangleright$ create new node
20: $E' \leftarrow E' \cup \{(a_1, A_{k+z}, w_1), (a_2, A_{k+z}, w_2)\}$
21: $S \leftarrow S \cup \{(A_{k+z}, b_i, 0)\}$
22: $D(A_{k+z}) \leftarrow \max(D(a_1), D(a_2)) + 1$
23: $D \leftarrow D \cup \{(A_{k+z}, D(A_{k+z}))\}$
24: **end if**
25: **end while**
26: $B \leftarrow B \setminus \{b_i\}$
27: **end while**

The minimum depth expansion algorithm guarantees that, it would create an adder graph from the related vertex reduced adder graph with the minimum depth. To get the minimum depth, this algorithm uses the method which manages the availability depth of the all nodes. Furthermore, the algorithm uses “As-Soon-As-Possible” (ASAP) addition algorithm, in which the addition would be done whenever both edges connected to an adder are ready. This is the last step of proposed algorithm which results an adder graph for the input matrix C . The algorithm indicates 0 for the depth of inputs, meaning that the value of them are defined and it gives -1 to other nodes’ depth, which means that the value of them are not calculated. Next, the minimum depth expansion algorithm looks for an adder inside the vertex reduced adder graph, where the depth of all its inputs are not -1 . It sorts all connected inputs based on their depth in an increasing format and starts to add the two first nodes which yields a new adder node and then compute the new adder node’s depth which is one more than the maximum depth of its inputs. This process continues to add all incoming nodes to each other to result the original node with the known depth. Next, the algorithm replaces the new calculated depth of mentioned adder node with the previous unknown depth, -1 , in the vertex reduced adder graph. Then, the algorithm continues to find the next adder node with available inputs and do the same process. Finally, it converts a vertex reduced adder graph to an adder graph which contains just two-input adders. The resulted adder graph has the minimum depth, subject to the vertex reduced adder graph as its input. To continue with the example, the final step of the algorithm is to apply the minimum depth expansion algorithm on the created vertex reduced adder graph in Fig. 7, the resulting adder graph in Fig. 8 is obtained.

4 Results

While it should be clear when solving the transposed problem instead of the original problem is beneficial from a solution time perspective, other aspects are not so obvious. Clearly, the effectiveness of the proposed approach depends on the quality of the solution of the transposed problem. Hence, the primary objective is to illustrate the properties of the approach and that it is potentially useful.

4.1 Adder Cost and Adder Depth

In the following results, the average adder costs and average maximum adder depths are compared to show the viability of the proposed method. To solve MCM (SOP) problems, the algorithm from [8] is used, while for CMM the algorithm in [7] is used. It should here be noted that

Figure 8 The resulting adder graph for the constant matrix $C = [7 \ 14 \ 8 \ -8 \ 4]$, after applying the minimum depth expansion algorithm in Section 3.3 to the graph in Fig. 7.

the algorithm in [8] only tries to minimize the adder cost, while the algorithm in [7] tries to minimize the adder cost, but at a minimal adder depth. However, the adder depth minimization is done for the transposed problem, and, hence, the adder depth of the original problem is not minimized. Still, [7] is one of the better published CMM algorithms.

For comparison, an implementation based on common two-term subexpression sharing is used. The algorithm is similar to that in [4] and uses CSD representation of the coefficients. However, two different strategies of selecting the subexpression to share was used. The first one, denoted SES in the following, picks the most common subexpression, as most algorithms do, with the purpose to minimize the adder cost and is therefore close to [4]. The second, denoted SES2, gives priority to minimum adder depth in the selection, therefore guaranteeing a minimum adder depth for the complete solution, while still sharing sub-expressions. For an explanation of subexpression sharing algorithms we refer the reader to [2].

In Fig. 9, the average adder costs for the different word lengths and different number of coefficients are shown for SOP solutions with random coefficients. As expected, the proposed solution provides the lowest number of adders as the underlying MCM algorithm is based on adder graphs and therefore not representation dependent. As can be seen, the benefit of the proposed approach increases with the word length. It is also clear that the SES approach results in fewer

Figure 9 Average adder cost for SOPs ($1 \times N$ matrices) solutions obtained from proposed approach, SES, and SES2, with different word lengths: (a) $W = 10$, (b) $W = 12$, (c) $W = 14$, and (d) $W = 16$.

adders compared to the minimum depth SES2 approach. The additional depth constraint comes at a price.

The resulting average adder depths are shown in Fig. 10. The minimum depth SES2 algorithm has step-like behavior. The minimum adder depth of a general CMM computation is

$$\max_i \left\{ \left\lceil \log_2 \left(\sum_{j=1}^N Z(C_{i,j}) - 1 \right) \right\rceil \right\}, \quad (3)$$

where $Z(C_{i,j})$ is the number of non-zero digits of coefficient $C_{i,j}$ [1]. Hence, with random coefficients, it is expected that the average value will follow the expectation values of the equation closely. When comparing the methods to obtain the SOPs, it is seen that the average adder depth of the proposed method will go below that of SES given enough coefficients. It also indicates that the average adder depth of SOP settles at a limit determined by the number of coefficient bits, given that it does not violate the lower bound. It should be noted that the use of a different MCM algorithm may change this behavior.

The CMM problem provides a significantly larger space for presenting results, combined with an increased solution time. Hence, we only provide limited results based on random coefficients here and instead provide some synthesis results later in this section and two real-world examples in the next section. Figure 11a shows the average required adder cost for solving constant $2 \times N$ matrices $\mathbf{C}$ that $N \in [2, 8]$ with random 12-bit coefficients. As expected, the proposed algorithm provides the lowest adder

cost as it uses an adder graph algorithm which solve the problem without depending on the number representation. Again, it is clear that the SES creates a smaller number of adders in comparison with SES2. Figure 11b presents the average adder depth and we can see a similar trend as for the SOP case.

4.2 Relation of Adder Depth in Original and Transposed Solution

To see the relation between the adder depths for the MCM and SOP solutions,² 1000 random $1 \times N$ matrices were considered for $N = 20$ and $N = 50$ and 12 or 16 bits matrices members. The results are shown in Fig. 12. First, it can be observed that the depth of the SOP solution is never smaller than the depth of the MCM solution. This is not surprising considering the underlying graph structure: the number of intermediate nodes between the input and one output will never decrease by transposing the graph. In addition, although (3) shows the lower bound on the depth, it is clear from it that the lower bound can never be smaller for an SOP compared to an MCM of the same coefficients. It is also possible to see that the adder depth of the MCM solution does not have that big impact on the adder depth of the SOP solution. For example, considering Fig. 12a the adder depth of most MCM solutions are between 2 and 14, while the adder depth of the SOP solution is between 8 and 15, more or less without any obvious relation (except for

²One would expect the conceptual behavior to be similar for CMM.

Figure 10 Average adder depth for SOPs ($1 \times N$ matrices) solutions obtained from proposed approach, SES, and SES2, with different word lengths: (a) $W = 10$, (b) $W = 12$, (c) $W = 14$, and (d) $W = 16$.

the bound observed initially. Hence, an MCM solution with adder depth 2 may give an SOP solution with adder depth 13, just as an MCM solution with an adder depth of 9 may give an SOP solution with an adder depth of 10.

To further illustrate this, consider the SOP computation of the 1×3 matrix $\mathbf{C} = [3 \ 7 \ 21]$. Its MCM form can be optimally realized using an adder cost 3 and adder depth 2 as shown in Fig. 13a. The SOP form shown in Fig. 13b results in an adder depth 4 and, as expected from (2), an adder cost 5. Introducing an additional adder, resulting in the MCM form shown in Fig. 13c still results in adder depth 2. However, when transposing this graph to obtain the SOP form, a solution with adder depth 3 is obtained (and expected adder cost 6). This further establishes that it is not (yet) clear what the preferred properties of the MCM solution is to obtain a low depth SOP solution.

4.2.1 Synthesis Results

As earlier discussed, not only the adder cost, but also the adder depth contributes to the power consumption. Hence, the power consumption is a combination of Figs. 9 and 10, where the importance of each is not obvious. It is possible to state that for many cases the proposed approach results in a lower power consumption than the subexpression sharing based solutions. Especially this should hold for many coefficients, where the savings in adder cost is large. However, it should be noted that the results are average values. Hence, one may expect individual cases where this relation is even more clear, and, naturally, also the opposite. The required area for implementing sum of product using adders and shift depends on the number of adders inside the implementation,. Since the proposed method provides the

Figure 11 (a) Average required adder cost, (b) Average required adder depth, of solutions provided with the proposed algorithm, SES and SES2 to solve $2 \times N$ matrices $\mathbf{C}$ that $N \in [2, 8]$.

Figure 12 Relation between the depth of an MCM solution in solving $N \times 1$ matrices $\mathbf{C}^T$ and the SOP solution for $1 \times N$ matrix $\mathbf{C}$ provided with the proposed approach for: (a) $W = 12$, $N = 20$, (b) $W = 12$, $N = 50$, (c) $W = 16$, $N = 20$, and (d) $W = 16$, $N = 50$.

smallest amount of adders, it should produce the least area for implementation. However, this will also depend on the timing constraints.

To show the power consumption and required area, VHDL was generated for each case. The adders are described using `numeric_std` and addition and subtraction operations (+ and -). The word length is adapted to every computation to be able to hold the complete result, so no quantization is performed and there will be no overflows. This was then synthesized using Design Compiler to a 65 nm CMOS standard cell library with varying timing constraints. The switching activity was obtained from gate level simulations with timing models in ModelSim. Finally, the switching activity was imported into Design Compiler before reporting the power consumption.

Here, we consider three instances of sum of product with a random $1 \times N$ matrix $\mathbf{C}$. In Fig. 14, area and power consumption for $N = 10$ and 8-bit matrix coefficients are shown. The adder costs are 18, 22, 23 and adder depths are 7, 6, 5 for the proposed approach, SES, and SES2, respectively.

As a second case, $N = 14$ and 10-bit matrix coefficients are considered. Here, the adder costs are 29, 35, 39 and adder depths are 7, 7, 6 for the proposed approach, SES, and SES2, respectively. The area and power consumption results are shown in Fig. 15. As the adder depths are about the same

for all solutions, they can all reach a critical path of 3 ns, and thanks to the lower adder count, both area and power are minimal using the proposed approach.

Finally, the last considered instance is $N = 30$ and random 12-bit matrix coefficients, with the results shown in Fig. 16. The adder costs are 62, 85, 89 and adder depths are 13, 8, 8 in proposed, SES, and SES2, respectively. While there is a significant reduction in area using the proposed approach, it is clear that the critical path cannot reach 4 ns caused by the higher adder depth. The power consumption is about the same, independent of algorithm here. For lower speeds, the proposed method has slightly lower power consumption, while for higher speeds, the proposed method leads to slightly increased power consumption. This is caused both by the synthesis tool introducing more circuit area to meet the timing requirements and the increased number of glitches.

From these three figures it is clear that it should always be beneficial from an area perspective to solve the transposed problem and transpose the solution as better algorithms can be used in reasonable time for solving the MCM problem compared to directly solving the SOP problem. However, as there is limited control of the resulting adder depth, this sometimes becomes an issue from a power consumption and timing perspective. It should be stressed that the proposed

Figure 13 Realization of the 1×3 matrix $\mathbf{C} = [3 \ 7 \ 21]$. **(a)** Optimal MCM solution for matrix $\mathbf{C}^T$, **(b)** SOP solution for matrix $\mathbf{C}$ from transposing **(a)** with adder depth 4, **(c)** MCM solution for matrix $\mathbf{C}^T$ with extra adder, and **(d)** SOP solution for $\mathbf{C}$ from transposing **(c)** with adder depth 3.

Figure 14 Area in μm^2 and power consumption of solving a sum of product ($1 \times N$ matrix $\mathbf{C}$) with $N = 10$ and $W = 8$ using the proposed approach, SES, and SES2.

Figure 15 Area in μm^2 and power consumption of solving a sum of product ($1 \times N$ matrix $\mathbf{C}$) with $N = 14$ and $W = 10$ using the proposed approach, SES, and SES2.

Figure 16 Area in μm^2 and power consumption of solving a sum of product ($1 \times N$ matrix $\mathbf{C}$) with $N = 30$ and $W = 12$ using proposed SOP, SES and SES2.

Table 1 Filter specifications for considered interpolation filters.

Interpolation factor	Filter order	Fractional bits	Passband edge	Stopband edge	Passband ripple	Stopband ripple
2	78	13	0.4π	0.5π	0.001	0.001
3	59	7	0.3π	0.35π	0.1	0.01

Table 2 Adder cost and adder depth for the filters using the proposed methods and sub-expression sharing methods.

Interpolation factor	Adder cost			Adder depth		
	Proposed	SES	SES2	Proposed	SES	SES2
2	92	112	115	9	8	7
3	46	60	65	8	6	6

approach will provide the minimum adder depth subject to the solution of the transposed problem, although the relation between the adder depth of the transposed and original problems still is unclear.

5 Design Examples

To illustrate the potential benefit of the proposed approach for a real-world application, two interpolation filters are considered. As discussed in [27], it is possible to compute the polyphase branches of an interpolation filter using a CMM approach. The filters are design using an approach similar to [28] to minimize the number of non-zero terms of the filter coefficients as this should give a low-complexity realization. The specifications of the filters are shown in Table 1, with resulting filter orders and number of fractional bits, i.e., bits to the right of the binary point, of the designed coefficients. The filter order is selected slightly higher than the theoretical minimum as this allows reducing the word length which gives a lower total complexity.

The CMM problems that should be solved here are of sizes 2×40 and 3×20 . Clearly, solving these problems using an adder graph algorithm may be very time consuming. Extrapolating the results from Fig. 1b very conservatively with one order of magnitude per two additional columns, the estimated time required is about 10^{20} s $\approx 3 \times 10^{12}$ years and 10^{10} s ≈ 300 years, respectively. Instead, solving the trans-

posed version is much more feasible obtaining the solution in a matter of seconds. Here, a version of the algorithm in [8] is used to obtain as low adder cost as possible. For comparison, SES and SES2 were used to solve the original problem directly. Again, as the complexity does not grow as rapidly for sub-expression sharing, these algorithms also finished within seconds. The resulting adder costs and adder depths for the algorithms are shown in Table 2.

The expected pattern repeats itself. The proposed method results in a clearly lower adder cost and the expense of a slightly increased adder depth. To see the effect of area and power consumption, a VHDL description was generated including the delay elements required from a filtering perspective. The input word length was 12 and 8 bits for interpolation by 2 and 3, respectively, and all bits of the results were kept. The results were then obtained as in the previous section and are shown in Figs. 17 and 18 for interpolation by 2 and 3, respectively. As can be seen, the power is about the same for all three approaches, but the area is significantly lower for the proposed approach. Hence, it would from all aspects be beneficial to use the proposed approach in this case.

For the general case, the effectiveness depends on the resulting depth of the transposed adder graph. However, these examples clearly show that there are cases where using the proposed approach on the transposed matrix is preferred over solving the original problem using a less efficient algorithm.

Figure 17 Area in μm^2 and power consumption of filter for interpolation by 2.

Figure 18 Area in μm^2 and power consumption of filter for interpolation by 3.

6 Conclusion

In this work a method to systematically obtain a transposed shift-and-add network with minimum depth subject to the original solution is presented. The main motivation for this is to be able to solve hard constant matrix multiplication problems, by solving the transposed problem and then transpose the solution. It is in general beneficial when the matrix is wider than it is tall, i.e., the number of columns is higher than the number of rows, as the computation time grows rapidly with the width/number of columns of the matrix. A simple example of this is obtaining a sum of products by instead solving a multiple constant multiplication problem.

While CMM problems can be readily solved using sub-expression sharing algorithms, the potential benefit in adder cost of using adder graph algorithms can be utilized with a much lower computation time compared to solving the original problem. It has been shown by examples that the connection between the adder depth, important for power consumption and to some extent timing, of the original and transposed solution is not straightforward. Sometimes a low depth solution leads to a significantly higher adder depth for the transposed solution, sometimes the difference is marginal. Despite this, it was shown that there were clearly examples of cases where this uncertainty was not a problem and the lower complexity from the adder graph solution lead to a lower power consumption for the original problem, despite not knowing what the resulting adder depth will be. However, as the proposed method will give the minimum adder depth subject to the solution of the transposed problem, it is of interest to further study what the solution of the transposed problem should look like to result in a low adder depth for the original problem.

Acknowledgments Open access funding provided by Linköping University.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Gustafsson, O. (2007). Lower bounds for constant multiplication problems. *IEEE Transactions on Circuits and Systems II*, 54(11), 974–978.
2. Meher, P.K., Chang, C.H., Gustafsson, O., Vinod, A.P., Faust, M. (2017). Shift-add circuits for constant multiplications. In Meher, P.K., & Stouraitis, T. (Eds.) *Arithmetic circuits for DSP applications* (pp. 33–76). Wiley.
3. Dempster, A.G., Gustafsson, O., Coleman, J.O. (2003). Towards an algorithm for matrix multiplier blocks. In *Proceedings of european conference circuit theory design* (pp. 1–4).
4. Macleod, M., & Dempster, A. (2004). Common subexpression elimination algorithm for low-cost multiplierless implementation of matrix multipliers. *Electronics Letters*, 40(11), 651–652.
5. Boullis, N., & Tisserand, A. (2005). Some optimizations of hardware multiplication by constant matrices. *IEEE Transactions on Computers*, 54(10), 1271–1282.
6. Gustafsson, O., Khursheed, K., Imran, M., Wanhammar, L. (2010). Generalized overlapping digit patterns for multi-dimensional sub-expression sharing. In *IEEE International conference green circuits systems* (pp. 65–68).
7. Kumm, M., Hardieck, M., Zipf, P. (2017). Optimization of constant matrix multiplication with low power and high throughput. *IEEE Transactions on Computers*, 66(12), 2072–2080.
8. Gustafsson, O. (2007). A difference based adder graph heuristic for multiple constant multiplication problems. In *Proceedings of IEEE international symposium on circuits and systems* (pp. 1097–1100): IEEE.

9. Hartley, R.I. (1996). Subexpression sharing in filters using canonic signed digit multipliers. *IEEE transactions on circuits and systems ii*, 43(10), 677–688.
10. Potkonjak, M., Srivastava, M.B., Chandrakasan, A.P. (1996). Multiple constant multiplications: Efficient and versatile framework and algorithms for exploring common subexpression elimination. *IEEE transactions on computers-aided design integrated circuits systems*, 15(2), 151–165.
11. Dempster, A.G., & Macleod, M.D. (2004). Using all signed-digit representations to design single integer multipliers using subexpression elimination. In *Proceedings of IEEE international symposium on circuits and systems*, (Vol. 3 pp. 165–168): IEEE.
12. Gustafsson, O., Dempster, A.G., Johansson, K., Macleod, M.D., Wanhammar, L. (2006). Simplified design of constant coefficient multipliers. *Circuits, Systems, and Signal Processing*, 25(2), 225–251.
13. Thong, J., & Nicolici, N. (2009). Time-efficient single constant multiplication based on overlapping digit patterns. *IEEE Transactions on Very Large Scale Integration*, 17(9), 1353–1357.
14. Gustafsson, O. (2008). Towards optimal multiple constant multiplication: a hypergraph approach. In *Proceedings of Asilomar conference on signals, systems, and computers* (pp. 1805–1809): IEEE.
15. Aksoy, L., Da Costa, E., Flores, P., Monteiro, J. (2008). Exact and approximate algorithms for the optimization of area and delay in multiple constant multiplications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(6), 1013–1026.
16. Kumm, M. (2018). Optimal constant multiplication using integer linear programming. *IEEE Transactions on Circuits and Systems II*, 65(5), 567–571.
17. Kumm, M., Zipf, P., Faust, M., Chang, C. (2012). Pipelined adder graph optimization for high speed multiple constant multiplication. In *Proceedings of IEEE international symposium on circuits and systems* (pp. 49–52).
18. Sarband, N.M., Gustafsson, O., Garrido, M. (2018). Obtaining minimum depth sum of products from multiple constant multiplication. In *Proceedings of IEEE workshop on signal processing systems* (pp. 1–6).
19. Gustafsson, O., Ohlsson, H., Wanhammar, L. (2001). Minimum-adder integer multipliers using carry-save adders. In *Proceedings of IEEE international symposium on circuits and systems*, (Vol. 2 pp. 709–712): IEEE.
20. Gustafsson, O., Dempster, A.G., Wanhammar, L. (2004). Multiplier blocks using carry-save adders. In *Proceedings of IEEE international symposium on circuits and systems*, (Vol. 2 pp. II–473): IEEE.
21. Gustafsson, O., & Wanhammar, L. (2011). Low-complexity and high-speed constant multiplications for digital filters using carry-save arithmetic. In Marquez, F.P.G. (Ed.) *Digital filters* (pp. 241–256). IntechOpen.
22. Kumm, M., Hardieck, M., Willkomm, J., Zipf, P., Meyer-Baese, U. (2013). Multiple constant multiplication with ternary adders. In *Proceedings of international conference on field-programmable logic applications* (pp. 1–8): IEEE.
23. Kumm, M., Gustafsson, O., Garrido, M., Zipf, P. (2018). Optimal single constant multiplication using ternary adders. *IEEE Transactions on Circuits and Systems II*, 65(7), 928–932.
24. Bordewijk, J.L. (1957). Inter-reciprocity applied to electrical networks. *Applied Scientific Research*, 6(1), 1–74.
25. Bhattacharyya, B., & Swamy, M. (1971). Network transposition and its application in synthesis. *IEEE Transactions on Circuit Theory*, 18(3), 394–397.
26. Schmid, H. (2002). Circuit transposition using signal-flow graphs. In *Proceedings of IEEE international symposium on circuits and systems*, (Vol. 2 pp. 25–28).
27. Gustafsson, O., & Dempster, A.G. (2004). On the use of multiple constant multiplication in polyphase FIR filters and filter banks. In *Proceedings of IEEE nordic signal processing symposium* (pp. 53–56).
28. Gustafsson, O., Johansson, H., Wanhammar, L. (2001). An MILP approach for the design of linear-phase FIR filters with minimum number of signed-power-of-two terms. In *Proceedings of european conference on circuit theory and design*.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Narges Mohammadi Sarband

received her B.Sc. degree in computer hardware engineering from Sadjad University of Technology, Mashhad, Iran in 2006 and her M.Sc. degree in computer architecture from University of Isfahan, Isfahan, Iran in 2014. She is from 2017 working towards her Ph.D. in electrical engineering with specialization in computer engineering at Linköping University (LiU), Sweden. Her research focuses on efficient design and hardware imple-

mentation of communication algorithms for beyond 5G systems in both ASIC and FPGA. She has also worked on optimizing peer to peer algorithms in computer networking.

Oscar Gustafsson received the M.Sc., Ph.D., and Docent degrees from Linköping University, Linköping, Sweden, in 1998, 2003, and 2008, respectively. He is currently an Associate Professor and Head of the Division of Computer Engineering, Department of Electrical Engineering, Linköping University. His research interests include design and implementation of DSP algorithms and arithmetic circuits in FPGA and ASIC. He has authored and

co-authored over 180 papers in international journals and conferences on these topics. Dr. Gustafsson is a member of the VLSI Systems and Applications and the Digital Signal Processing technical committees of the IEEE Circuits and Systems Society. He has served as an Associate Editor for the IEEE Transactions on Circuits and Systems Part II: Express Briefs and Integration, the VLSI Journal. Furthermore, he has served and serves in various positions for conferences such as ISCAS, PATMOS, PrimeAsia, ARITH, Asilomar, NorCAS, ECCTD, and ICECS.

Mario Garrido received the M.Sc. degree in electrical engineering and the Ph.D. degree from the Technical University of Madrid (UPM), Madrid, Spain, in 2004 and 2009, respectively. In 2010 he moved to Sweden to work as a postdoctoral researcher at the Department of Electrical Engineering at Linköping University. From 2012 to 2019 he was Associate Professor at the same department. In 2019 he moved back to UPM, where he holds a Ramón y Cajal

Research Fellowship. His research focuses on optimized hardware design for signal processing applications. This includes the design of hardware architectures for the calculation of transforms, such as the fast Fourier transform (FFT), circuits for data management, rotators such as the CORDIC algorithm, circuits to calculate statistical and mathematical operations, and neural networks. His research covers high-performance circuits for real-time computation, as well as designs for small area and low power consumption.