Towards an Optimised VLSI Design Algorithm for the Constant Matrix Multiplication Problem

Andrew Kinane, Valentin Muresan and Noel O'Connor
Centre for Digital Video Processing, Dublin City University, Dublin 9, IRELAND
Email: kinanea@eeng.dcu.ie

Abstract— **The efficient design of multiplierless implementations of constant matrix multipliers is challenged by the huge solution search spaces even for small scale problems. Previous approaches tend to use hill-climbing algorithms risking sub-optimal results. The proposed algorithm avoids this by exploring parallel solutions. The computational complexity is tackled by modelling the problem in a format amenable to genetic programming and hardware acceleration. Results show an improvement on state of the art algorithms with future potential for even greater savings.**

I. INTRODUCTION

Applications involving the multiplication of variable data by constant values are prevalent throughout signal processing. Some common tasks that involve these operations are Finite Impulse Response filters (FIRs), the Discrete Fourier Transform (DFT) and the Discrete Cosine Transform (DCT). Optimisation of these kind of constant multiplications will significantly impact the performance of such tasks and the global system that uses them. The examples listed are instances of a more generalised problem – that of a linear transform involving a constant matrix multiplication (CMM). The problem is summarised as follows: substitute all multiplications by constants with a minimum number of shifts and additions/subtractions (we refer to both as "additions") [1]. The optimisation criterion may be extended beyond adder count only and include factors like routability, glitching etc. but is restricted to adder count in this paper.

II. PROBLEM STATEMENT

A CMM equation $\mathbf{y} = \mathbf{A}\mathbf{x}$ (where $\mathbf{y}, \mathbf{x}$ are N-point 1D data vectors and $\mathbf{A}$ is an $N \times N$ matrix of M-bit fixed-point constants) may be thought of as a collection of N dot products with each dot product y_i expressed as follows:

$$y_i = \sum_{j=0}^{N-1} a_{ij} x_j, \quad i = 0, \ldots, N-1 \tag{1}$$

Each constant may be represented in signed digit (SD) form:

$$a_{ij} = \sum_{k=0}^{M-1} b_{ijk} 2^k, \quad b_{ijk} \in \left\{\bar{1}, 0, 1\right\}, \quad \bar{1} \equiv -1 \tag{2}$$

Combining (1) and (2) yield a multiplierless dot product implementation requiring only adders and shifters:

$$y_i = \sum_{j=0}^{N-1} \sum_{k=0}^{M-1} b_{ijk} 2^k x_j, \quad i = 0, \ldots, N-1 \tag{3}$$

The goal is to find the optimal sub-expressions across all N dot products in (3) that lead to the fewest adder resources needed. Three properties aid the classification of approaches: SD permutation, pattern search strategy and problem subdivision.

1) SD Permutation: Consider that each of the $N \times N$ M-bit fixed point constants a_{ij} have a finite set of possible SD representations. For example with $M = 4$ the constant $(-3)_{10}$ can be represented as either $(00\bar{1}\bar{1})_2, (0\bar{1}01)_2, (\bar{1}101)_2, (0\bar{1}1\bar{1})_2, (\bar{1}11\bar{1})_2$. For the optimal number of adders to be found, all SD representations of the a_{ij} should be considered since for a CMM problem Canonic Signed Digit (CSD) representation is not guaranteed to be optimal (as shown in section V). The difficulty is that the solution space is very large, hence SD permutation has only thus far been applied to simpler problems [2]. Potkonjak et. al. acknowledge the potential of SD permutation but choose a single SD representation for each a_{ij} using a greedy heuristic. Neither of the recent CMM-specific algorithms apply SD permutation [3], [4].

2) Pattern Search: The pattern search goal is to find the sub-expressions in the 3D bit matrix b_{ijk} resulting in fewest adders. Usually b_{ijk} is divided into N 2D slices along the i plane (i.e. taking each CMM dot product in isolation). Patterns are searched for in the 2D slices independently before combining the results for 3D. An example 2D slice is shown in (4), a 4-point dot product with random 12-bit SD constants.

$$y_i = \begin{bmatrix} 2^{-11} \\ 2^{-10} \\ \vdots \\ 2^{-1} \\ 2^{0} \end{bmatrix}^T \underbrace{\begin{bmatrix} 0 & 0 & 1 & \bar{1} \\ 0 & \bar{1} & 0 & 0 \\ \bar{1} & 1 & 0 & 0 \\ 0 & 0 & \bar{1} & 1 \\ \bar{1} & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & \bar{1} \\ 0 & 0 & 0 & 0 \\ 0 & \bar{1} & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\text{2D slice of } b_{ijk}} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{4}$$

Algorithms may search for horizontal/vertical patterns (P1D) or diagonal patterns (P2D) in the 2D slice. The P1D strategy infers a two-layer architecture of a network of adders (with no shifting of addends) to generate distributed weights for each row followed by a fast partial product summation tree (PPST)

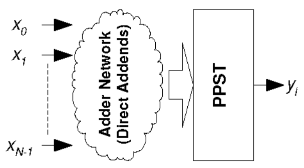


Fig. 1. P1D Architecture.

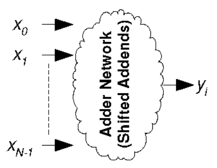


Fig. 2. P2D Architecture.

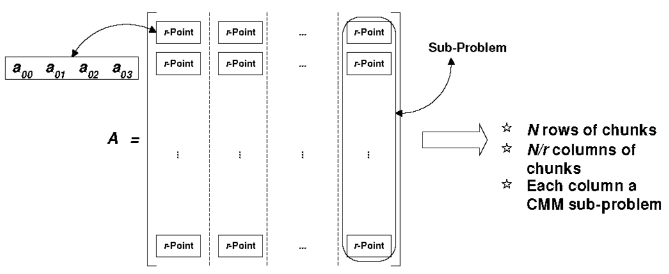
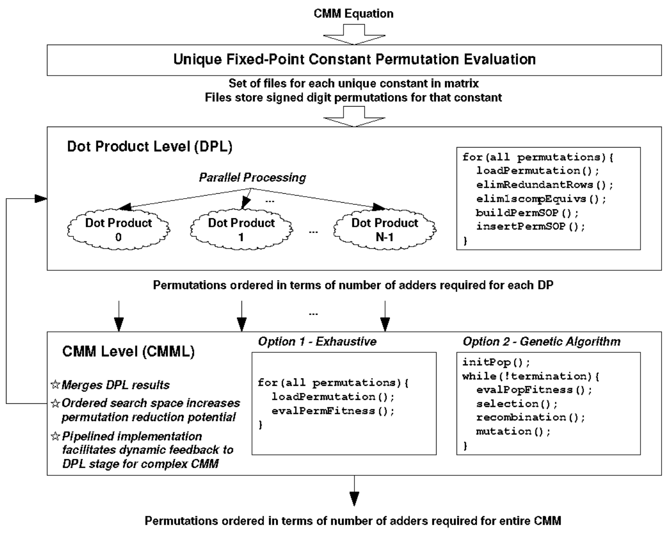

a_{00} a_{01} a_{02} a_{03}
A =
r-Point
Sub-Problem
☆ N rows of chunks
☆ N/r columns of chunks
☆ Each column a CMM sub-problem

Fig. 3. CMML Divide and Conquer.


CMM Equation
Unique Fixed-Point Constant Permutation Evaluation
Set of files for each unique constant in matrix
Files store signed digit permutations for that constant
Dot Product Level (DPL)
Parallel Processing
Dot Product 0
Dot Product 1
Dot Product N-1
Permutations ordered in terms of number of adders required for each DP
CMM Level (CMML)
☆Merges DPL results
☆Ordered search space increases permutation reduction potential
☆Pipelined implementation facilitates dynamic feedback to DPL stage for complex CMM
Option 1 - Exhaustive
Option 2 - Genetic Algorithm
Permutations ordered in terms of number of adders required for entire CMM

```
for(all permutations){
  loadPermutation();
  elimRedundantRows();
  elim1scompEquivs();
  buildPermSOP();
  insertPermSOP();
}
```

```
for(all permutations){
  loadPermutation();
  evalPermFitness();
}
```

```
initPop();
while(!termination){
  evalPopFitness();
  selection();
  recombination();
  mutation();
}
```

Fig. 4. Summary of the CMM Optimisation Algorithm.

to carry out the shift accumulate (Fig. 1). The P2D strategy infers a one-layer architecture (Fig. 2) of a network of adders that in general may have shifted addends (essentially merging the two layers of the P1D strategy). Potkonjak et. al. use the P1D strategy and search for horizontal patterns while others use the P2D strategy [3], [4]. However, these proposals select sub-expressions iteratively based on some heuristic criteria that may preclude an optimal realisation of the global problem. This is because the order of sub-expression elimination affects the results [5].

3) Synthesis: As in any hardware optimisation problem, synthesis issues should be considered when choosing sub-expressions for an N-point dot product (a 2D slice). If N is large (e.g. 1024-point FFT) then poor layout regularity may result from complex wiring of sub-expressions from taps large distances apart in the data vector. Indeed a recent paper has shown that choosing such sub-expressions can result in a speed reduction and greater power consumption [6]. It is therefore sensible to divide each N-point dot product into N/r r-point chunks and optimise each sub dot product independently. The CMM problem hence becomes N/r independent sub problems, each with N dot products of length r (Fig. 3). The optimal choice of r is problem dependent, but the proposed algorithm currently uses $r = 4$ for reasons outlined subsequently.

III. Proposed Efficient Modelling Solution

The CMM problem is a difficult discrete combinatorial problem and currently requires a shift to a higher class of algorithms for more robust near-optimal solutions. This is because the current approaches are greedy hill-climbing algorithms and the associated results are very problem dependent [5]. The challenge is in the modelling of the problem to make it amenable to efficient computation. The algorithm proposed here models the problem in such a way as to make it amenable to so-called near-optimal algorithms (genetic algorithms (GAs), simulated annealing, tabu-search) and also hardware acceleration. The proposed approach incorporates SD permutation of the matrix constants and avoids hill-climbing by evaluating parallel solutions for each permutation. Such an approach is computationally demanding but the algorithm has been modelled with this in mind and incorporates innovative fast search techniques to reduce this burden.

The proposed algorithm permutes the SD representations of the constants in $\mathbf{A}$. For each permutation, parallel solution options are built based on different sub-expression choices. These parallel implementations are expressed as a sum of products (SOP), where each product term in the SOP represents a particular solution (with an associated adder count). The SD permutation is done on each CMM dot product in isolation (Section IV-A), and the results are subsequently combined (Section IV-B). The algorithm searches for the combined SOP that represents the overall best (in terms of adder count) sub-expression configuration to implement the CMM equation. Previous approaches derive one implementation option (akin to a single term SOP) whereas the proposed approach derives parallel implementations (a multi-term SOP). It is this multi-term SOP approach and its manipulation (Section IV) that make the algorithm suitable for GAs and hardware acceleration.

The proposed algorithm currently uses the P1D strategy, so it searches for horizontal sub-expression patterns of $\{\pm 1\}$ digits in a 2D slice. The proposed SOP modelling idea can be extended to cover the P2D strategy by simply extending the digit set from $\{\pm 1\}$ to $\left\{\pm 1, \pm 2, \pm\frac{1}{2}, \pm 4, \pm\frac{1}{4}, \ldots\right\}$.

IV. The Proposed CMM Optimisation Algorithm

The proposed approach is a three stage algorithm as summarised in Fig. 4. Firstly all SD representations of the M-bit fixed point constants are evaluated using an M-bit radix-2 SD counter. Then, each dot product in the CMM are processed independently by the dot product level (DPL) algorithm. Finally the DPL results are merged by the CMM level (CMML) algorithm. The three steps may execute in a pipelined manner with dynamic feedback between stages. This offers search space reduction potential as outlined subsequently.

A. Dot Product Level (DPL) Stage

The DPL algorithm iteratively builds a SOP, and the final SOP terms are the unique sub-expression selection options after considering all SD permutations of the dot product constants in question. The final SOP terms are listed in increasing order of the number of adders required by the underlying sub-expressions. The DPL algorithm executes the following steps for each SD permutation.

Step1: Load the next SD permutation of the dot product constants. This corresponds to a 2D slice of b_{ijk}, e.g. in (4).

Step2: Using the P1D strategy, slice rows with ≤ 1 non-zeros need not be considered for sub-expression sharing and are eliminated.

Step3: It may be the case that there are occurrences where a certain pattern is on one row and it's 1's complement occurs on another row. Such pairs infer the same unique set of additions, albeit the final output is the $\pm$ of the other. Thus one of these rows can be eliminated since only additional inverters are required and the '1' added to the LSB can be subsumed by the PPST.

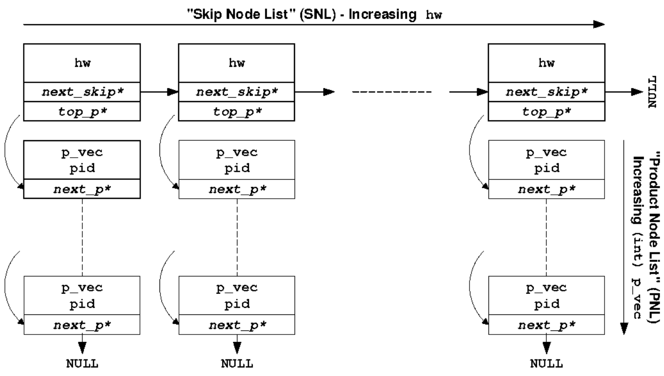
Step4: Steps 2 and 3 reduce the 2D slice in (4) to the left matrix in (5). The DPL algorithm considers each row in turn and builds an implementation SOP for that row, as in (5).

$$\begin{bmatrix} 0 & 0 & 1 & \bar{1} \\ \bar{1} & 1 & 0 & 0 \\ 0 & 1 & 1 & \bar{1} \\ 0 & \bar{1} & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} (p_{11}) \text{ AND} \\ (p_6) \text{ AND} \\ ((p_3)(p_{51}) \text{ OR } (p_{10})(p_{52}) \text{ OR } (p_{11})(p_{53})) \text{ AND} \\ (p_{10}) \text{ AND} \\ (p_0) \end{bmatrix} \quad (5)$$

Each SOP term is represented internally as a data structure with elements p_vec (a bit vector where each set bit represents a specific adder to be resource allocated) and hw (the Hamming weight of p_vec that records the total adder requirement). The number of possible two input additions is equivalent to the combinatorial problem of leaf-labelled complete rooted binary trees [7]. With $r = 4$, the number of possibilities is 180 (proof omitted to save space) and the general series in r increases quickly for $r > 4$. We are currently researching an automated method for configuring the DPL algorithm for any r. So each p_vec is a 180-bit vector with a hw equal to the number of required adders. The SOPs for each row are logically ORed together to form a permutation SOP that is an exhaustive set of sub-expressions options that implement the entire permutation. The permutation SOP for (4) is given by (6) where p_v means bit v is set in the 180-bit p_vec for that SOP term.

$$\begin{aligned} &((p_{11})(p_6)(p_3)(p_{51})(p_{10})(p_0)) \text{ OR} \\ &((p_{11})(p_6)(p_{10})(p_{52})(p_0)) \text{ OR} \\ &((p_{11})(p_6)(p_{53})(p_{10})(p_0)) \end{aligned} \quad (6)$$

The first term in (6) has hw = 6 so it requires 6 unique additions (+PPST) to implement (4) whereas the latter two options only require 5 unique additions (+PPST). Obviously one of the latter two options is more efficient if implementing this dot product in isolation. However when targeting a CMM problem one must consider the CMM level, and it may be that permuting the first option at CMML gives in a better overall result since it may overlap better with requirements for the other dot products. Hence it is necessary to store the entire SOP for each permutation at DPL and then permute these at CMML to get the guaranteed optimal.



"Skip Node List" (SNL) - Increasing hw
hw
next_skip*
top_p*
p_vec
pid
next_p*
NULL
"Product Node List" (PNL) Increasing (int) p_vec

Fig. 5. DPL Skiplist Arrangement.

Step5: The algorithm checks each term in the current SOP produced by step 4 to see if it has already been found with a previous permutation. If so it is discarded – only unique implementations are added to the global list. This global list is implemented using a 2D skip list to minimise the overhead of searching it with a new term from the current permutation SOP (Fig. 5). In the horizontal direction there are "skip nodes" ordered from left to right in order of increasing hw in the skip node list (SNL). In the vertical direction there are "product nodes" and each skip node points to a product node list (PNL) of ordered product nodes where each product node in the PNL has the same number of bits set (i.e. hw) in its p_vec bit vector. Therefore if a new node is presented with hw = hw_{new} and p_vec = p_vec_{new}, it makes sense to only search the subset of nodes in the global list that have the same value hw_{new} (i.e. only search one particular PNL). The PNLs are ordered in increasing order of their p_vec, if p_vec is considered as a 180-bit integer. Therefore when iterating over a PNL at a given skip node and a node is found that has a p_vec (p_vec_{cur}) such that $\text{p\_vec}_{cur} > \text{p\_vec}_{new}$ it is guaranteed that p_vec_{new} is not already in the list and can be inserted at this point. When inserting into the list a unique permutation ID (pid) is added to the node along with p_vec so that the SD permutation that generated it can be reconstructed. If the condition $\text{p\_vec}_{cur} = \text{p\_vec}_{new}$ is true, then the SOP term already exists in the PNL so the new node is discarded and the search terminates immediately for this SOP term.

The DPL algorithm is dominated by low level operations such as comparisons, Boolean logic and bit counting. Indeed profiling shows that on average 60% of the computation time is consumed by bit counting (50%) and bitwise OR (10%). Such tasks can readily be accelerated in hardware by mapping SOPs to a FIFO structure and the logic OR operations to OR gates.

B. Constant Matrix Multiplication Level (CMML) Stage

Once the DPL algorithm has run for each of the dot products in the CMM, there will be N 2D skip lists – one for each of the N dot products examined. The task now is to find the best overlapping product nodes for all of the CMM dot products. It is expected (though not guaranteed) that since the skiplists are ordered with the lowest hw PNL first, the optimal result will be converged upon quickly saving needless searching of large areas of the permutation space. The CMM Level (CMML) algorithm searches for the optimal overlapping nodes from each of the DPL lists.

The CMML algorithm permutes the terms in each skiplist with terms from others, starting from the top of each. For each permutation the N product nodes are combined using bitwise OR and bit counting similar to the techniques used in the DPL algorithm. The value of hw of the combined node represents the number of adders necessary to implement the CMM for the current permutation. The potential exists to use lowest hw value found thus far to rule out areas of the search space. For example if an improved value of hw = 5 is found for a CMML solution, there is no point in searching DPL PNLs with hw > 5 since they are guaranteed not to overlap with other DPL PNLs and give a better result than 5. The current best value of hw at CMML level could also be fed back to the DPL algorithm to reduce the size of the skiplists generated by DPL (and hence permutation space) without compromising optimality.

Although the ordering of the search space makes it more likely for the exhaustive CMML algorithm to find the best solution relatively quickly, the huge permutation space means that the exhaustive CMML approach is not tractable. However, the proposed modelling of the CMM problem and bit vector representation of candidate solutions means that the CMML algorithm is very amenable to GAs. The bit vectors can be interpreted as chromosomes and the value of hw can be used to build an empirical fitness function (the less adders required the fitter the candidate). Details of a suitable GA can be found in [8]. The current fitness function is based upon the number of adders but may in future be extended to include parameters such as layout, and speed.

V. Experimental Results

For a fair comparison with other approaches, the number of 1-bit full adders (FAs) allocated in each optimised architecture should be used as opposed to "adder units", since the bitwidth for each unit is unspecified in other publications apart from [4]. FA count more accurately represents circuit area requirements. Using the 8-point 1D DCT ($N = 8$ with various M) as a benchmarking CMM problem, Table I compares results with other approaches based on adder units and FAs where possible. Our approach compares favourably with [4] in terms of FAs (see FA% savings in Table I), even though this gain is not reflected by the number of adder units required.

The modelling approach used means that the proposed DPL algorithm is tractable and can search the DPL SD permutation space exhaustively in the order of hours (although this increases with M). However, the CMML search space is huge and the results presented here are based on searching <1% of the possibilities (using an untuned GA). However, it is expected that the ordering of the DPL solutions means that

TABLE I
1D 8-point DCT Adder Unit / Full Adder Requirements

CMM	Initial	[1]	[5]	[4]		Proposed Approach		
	+	+	+	+	FA	+	FA	FA%
DCT 8bit	300	94	65	56	739	78	730	1.2
DCT 12bit	368	100	76	70	1202	109	1056	12.1
DCT 16bit	521	129	94	89	2009	150	1482	26.2

the most promising regions of the CMML search space are examined first. The hypothesis of achieving extra saving by permuting the SD representations is validated by the fact that the best SD permutation yielding the results in Table I are not the CSD permutation.

Although the savings achieved are incremental, there exists significant potential for improvement:

i: Investigation into the optimal value of r – that is the optimal sub division of large CMM problems into independent chunks. This can only be truly evaluated if synthesis parameters such as fanout and routability are included in the optimisation criterion as well as FA count.

ii: The integration of the P2D strategy mentioned earlier. It is likely that there exists a maximum number of rows apart in the b_{ijk} slice diagonal patterns forming useful sub-expressions will be. This is because if sub-expression addends come from rows far apart in b_{ijk}, the adders inferred have a large bitwidth.

iii: Optimal tuning of the CMML GA parameters to search the permutation space most effectively.

VI. Conclusions

The general multiplierless CMM design problem has a huge search space, especially if different SD representations of the matrix constants are considered. The proposed algorithm addresses this by organising the search space effectively and modelling the data in a fashion amenable to GAs and hardware acceleration. Preliminary results validate the approach and show an improvement on the current state of the art.

References

[1] 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. Computer-Aided Design*, vol. 15, no. 2, pp. 151–165, Feb. 1996.

[2] A. G. Dempster and M. D. Macleod, "Digital Filter Design Using Subexpression Elimination and all Signed-Digit Representations," in *Proc. IEEE International Symposium on Circuits and Systems*, vol. 3, May 2004, pp. 169–172.

[3] M. D. Macleod and A. G. Dempster, "Common subexpression elimination algorithm for low-cost multiplierless implementation of matrix multipliers," *IEE Electronics Letters*, vol. 40, no. 11, pp. 651–652, 2004.

[4] N. Boullis and A. Tisserand, "Some Optimizations of Hardware Multiplication by Constant Matrices," *IEEE Trans. Comput.*, vol. 54, no. 10, pp. 1271–1282, Oct. 2005.

[5] M. D. Macleod and A. G. Dempster, "Multiplierless FIR Filter Design Algorithms," *IEEE Signal Processing Lett.*, vol. 12, no. 3, pp. 186–189, Mar. 2005.

[6] M. Martinez-Peiro, E. I. Boemo, and L. Wanhammar, "Design of High-Speed Multiplierless Filters Using a Nonrecursive Signed Common Subexpression Algorithm," *IEEE Trans. Circuits Syst. II*, vol. 49, no. 3, pp. 196–203, Mar. 2002.

[7] S. D. Andres, "On the number of bracket structures of n-operand operations constructed by binary operations," 2005, private communication.

[8] A. Kinane, V. Muresan, and N. O'Connor, "Optimisation of Constant Matrix Multiplication Operation Hardware Using a Genetic Algorithm," in *Proc. 3rd European Workshop on Evolutionary Computation in Hardware Optimisation (EvoHOT)*, Budapest, Hungary, Apr. 10–12, 2006.