Towards an Algorithm for Matrix Multiplier Blocks

A G Dempster[1], O Gustafsson[2], J O Coleman[3]

[1]University of Westminster, 115 New Cavendish St, London W1W 6UW, UK, dempsta@wmin.ac.uk

[2] Dept of Elect Engineering, Linköping University, SE-581 83 Linköping, SWEDEN oscarg@isy.liu.se

[3]Naval Research Laboratory, Washington DC, jeffc@alum.mit.edu

***Abstract* -.**
The basic elements of an algorithm for designing multiplier blocks for matrices are presented. The new algorithm often produces results superior to the best of the older algorithms applied only to columns.

1. INTRODUCTION

1.1. Multiplier Blocks: Background

Multiplier blocks reduce the chip area and power consumption of shift-and-add multiplication circuits. After Bull and Horrocks [1], we developed algorithms that use fewest adders for fixed-point constant integer multiplication [2, 3, 4]. We also developed the most efficient (i.e. fewest adders) algorithms for several products of a single multiplicand [5], a process well suited to FIR filters. Although the application of this approach to the "several products of several multiplicands" case (i.e. matrix multiplication) had been shown to be useful some time ago [6], very little progress had been made in this area.

Only two areas have been examined involving extra "dimensionality" in relation to multiplier blocks: subexpression sharing and filter banks.

1.2. Subexpression Sharing

Subexpression sharing, introduced by Hartley [8], is a technique with the same aim as the multiplier block algorithms discussed above: to reduce the number of adders required for fixed-point multiplication. Because it is restricted in its operation by the coefficient representation it chooses initially, it tends to be less versatile and hence less efficient than multiplier blocks. However, subexpression sharing can exploit the knowledge that in an FIR filter each sample will be delayed and used again. An example in [9] shows an FIR filter with coefficients (240, 150, 15) can be executed with three adders and two delays, as in Figure 1a). A RAG-n or BHM [5] design would produce a multiplier block which must integrate with the standard transposed form structure, as in Figure 1b).

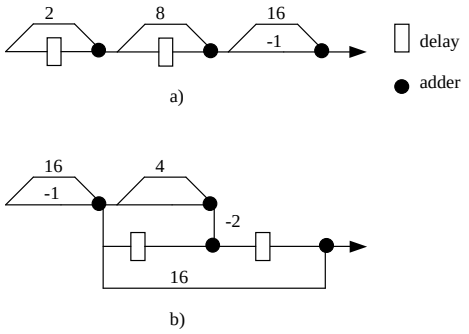

Figure 1. Multiplier block designs for coefficient set (240, 150, 15): a) subexpression sharing (3 adders), b) RAG-n or BHM (4 adders).

Although, in general, the multiplier block algorithms produce better results than subexpression sharing, here we see a benefit because not only are the edges of the graphs scalable by shifts (i.e. x 2^n) but also by delays (i.e. x z^{-m}). So the first adder in the subexpression sharing graph produces an output of (2 + z^{-1}) X(z). Effectively, both scaling and delay are *dimensions* of versatility in this problem.

1.3. Filter Banks

An FIR filter can be considered as an inner product between a series of input samples $x(n)$ and the filter coefficients $h(n)$

$$y(n) = \sum_{k=0}^{N} h(k)x(n-k) \qquad (1)$$

A filter bank consisting of M, say, parallel FIR filters of equal length each of which would have the form:

$$y_i(n) = \sum_{k=0}^{N} h_i(k)x(n-k)$$

or alternatively could be written in matrix form:

$$\mathbf{y}(n) = \mathbf{H}\mathbf{x}(n) \qquad (2)$$

where

$$\mathbf{y}(n)=\begin{bmatrix} y_1(n) \\ y_2(n) \\ \vdots \\ y_M(n) \end{bmatrix} \quad \mathbf{x}(n)=\begin{bmatrix} x(n) \\ x(n-1) \\ \vdots \\ x(n-N) \end{bmatrix}$$

$$\mathbf{H}=\begin{bmatrix} h_1(0) & h_1(1) & \cdots & h_1(N) \\ h_2(0) & h_2(1) & & h_2(N) \\ \vdots & & \ddots & \\ h_M(0) & h_M(1) & & h_M(N) \end{bmatrix}$$

Using rules of transposition we have shown that **H** can be represented by a single multiplier block [10]. In other words, all coefficients of all the sub-filters can be combined, yielding a highly efficient circuit. This solution, however, does not apply to the general matrix multiplication case. (This is because once the multiplier block is designed, figure 3b of [10] must be retransposed. When this occurs, link A can no longer be rebroken in general. This boils down to a similar scenario to that discussed above for subexpression sharing – by exploiting the knowledge that we are designing FIR filters, we achieve greater efficiency.)

2. NEW ALGORITHM DESIGN ISSUES

2.1. Graphical Representation

Whereas multiplier blocks have usually had integer labels on each vertex identifying the partial product to that point ("fundamental" [5]), in the matrix case these vertices must be labelled with the vector producing the partial dot product to that point. Note that each vertex still represents a 2-input adder and the output at each vertex is still a number (not a vector). Whereas the input to a multiplier block could have been labelled "1", for a matrix multiplier block, the inputs could thus be labelled [1 0 … 0], [0 1 0 …0] etc. The example of [6] is used to illustrate this method. The aim is to create a circuit that implements the matrix multiplication

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}=\begin{bmatrix} 21 & 39 \\ 11 & 5 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad (3)$$

A circuit to do this is shown in Figure 2a): "free" shifts and a single adder each are used to produce products $3x_1$ and $5x_2$. A third adder produces $(3x_1 + 5x_2)$, indicated as [3 5]. The output vertices of the graph are labelled with the corresponding row of the matrix.

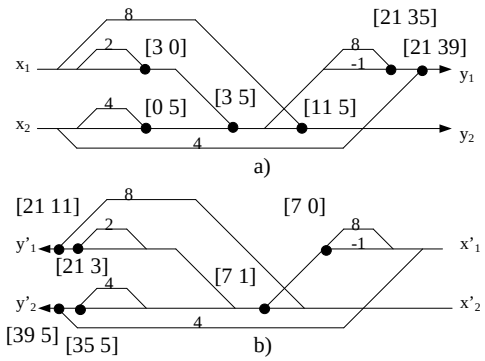

Figure 2 Matrix multiplier blocks for a) the matrix in (3) and b) its transpose (inputs at right)

2.2. Simplifications

Simplifications available to the earlier algorithms (MAG, RAG-n, BHM) are not available here:

i) They dealt only with odd numbers; here a row of a matrix can only be divided by 2 until one element is odd.

ii) They eliminate repetitions of coefficients; here a row can be ignored only if it is a multiple of another row, else it must be considered.

iii) They only considered positive integers, with negatively weighted edges representing use of subtractors; now positive and negative integers are considered as both may appear in a matrix row.

2.3. Algorithm Structure

The algorithm presented here for designing blocks for matrix multiplication is most similar to BHM [5], based on Bull's original [1]. It is similar to BHM in that it iteratively approaches the required multipliers, but has one of the advantages of RAG-n [5] (producing products in any order).

The algorithm works as follows:

i) The initial "sums" [1 0 … 0], [0 1 0 …0] etc are stored in a fundamental table.

ii) If the maximum coefficient in the matrix is M, then all products of all the fundamentals in the table by $\pm 2^n$ are generated such that no product exceeds 2M. (These are all the possible scaled inputs to new adders). For example if a 2x2 matrix has maximum coefficient 3 then the vectors [-4 0] [-2 0] [-1 0] [1 0] [2 0] [4 0] [0 –4] [0 –2] [0 –1] [0 1] [0 2] [0 4] are generated from the initial fundamentals.

iii) All possible sums between these vectors are generated. The sum that comes closest (what "closest" means we discuss below) to <u>any</u> row of the required

matrix is kept and added to the fundamental table. (Therefore the fundamental table lists the input vertices plus any outputs of adders that have been built into the graph)

iv) If a new adder exactly produces a row of the matrix then that row is eliminated from further consideration

v) Go to step ii) until all rows are eliminated.

2.4. Distance Measure

Whereas BHM chose the result closest to its target coefficient as simply being that with the smallest scalar difference, here we have a number of ways of measuring "distance". The two we considered were the Manhattan ("city-block") distance and the adder cost distance. The Manhattan distance between vectors v_1 and v_2 each of N element is:

$$\mathrm{MD} = \sum_{i=1}^{N} \mathrm{abs}\left(v_1(i) - v_2(i)\right) \tag{4}$$

The adder cost distance is:

$$\mathrm{AD} = \gamma\left(v_1 - v_2\right) + \sum_{i=1}^{N} \mathrm{MAG}\left[\mathrm{abs}\left(v_1(i) - v_2(i)\right)\right] \tag{5}$$

where MAG is the lookup table of single-coefficient adder costs produced by the MAG algorithm [2] and $\gamma(v)$ is an operator that counts the number of non-zero elements of v. This cost is thus the number of adders required to be further included in the graph such that the "required" vector could be produced from our current vector of interest by simply adding a vector that was synthesised completely by single-coefficient multipliers.

2.5. The "Kick-Start"

It was found experimentally that the use of adder cost distance (AD) led to designs that were constructed row by row, coefficient by coefficient, with little use of the redundancy we want to exploit. A "kick-start" was used, where the row with the lowest initial AD was synthesised not according to the algorithm, but as separate single adders.

2.6. Transposition

A multiplier block that produces several products of a single multiplicand, with the structural adders and delays of an FIR filter, can be transposed, producing a filter with identical coefficients, with the structural adders inside the transposed multiplier block. If transposition applies only to the multiplier block (i.e. ignoring the filter structure), it becomes a "many-to-one" structure rather than "one-to-many". In inner product terms, the row vector becomes a column vector (it is "transposed").

In the case of matrix multiplication, transposing the network, as in Figure 2b), produces a circuit that multiplies by the transposed matrix, i.e.

$$\begin{bmatrix} y'_1 \\ y'_2 \end{bmatrix} = \begin{bmatrix} 21 & 11 \\ 39 & 5 \end{bmatrix} \begin{bmatrix} x'_1 \\ x'_2 \end{bmatrix} = \begin{bmatrix} 21 & 39 \\ 11 & 5 \end{bmatrix}^T \begin{bmatrix} x'_1 \\ x'_2 \end{bmatrix} \tag{6}$$

This shouldn't be surprising, as the original transposition theorem [11] used "transpose" in this context. What this means for us though is that we can freely operate on either rows or columns as we build up the multiplier block, and transpose the result if columns were used.

3. RESULTS

The results presented here are preliminary, because exhaustive testing has not yet been attempted and the algorithms are still under development. However, two experiments have produced some interesting results. For two scenarios, the costs produced by the new algorithm were compared to the costs of applying RAG-n to each column and then using structural adders to combine the results at the end:

Scenario 1. Matrices of size 2 x 2 were compared for wordlengths w =4 to 20 bits.

Scenario 2. Matrices of size n x n were compared for wordlength 8.

The implementation of the algorithm is quite slow, and only small sample sizes have typically been tested, so our results are qualitative. However, the following observations were made:

i) Using MD in scenario 1, the new algorithm tends to give superior results on average for $w < 8$ bits. For $4 < w < 12$ bits, at least 20% of matrices are produced using fewer adders than under RAG-n. For $12 < w < 20$, this figure is 5-10%.

ii) Using MD in scenario 2 results in 20% or more of matrices having fewer adders than for RAG-n for 2x2 and 3x3 matrices. As matrices grow larger, the new algorithm designs fewer better.

iii) Using AD (as discussed in section 2.5), the algorithm works on one coefficient producing long thin graphs. The kick-start reduced the adder cost in some cases by 30% and made the new algorithm much more competitive for larger wordlengths.

iv) Using AD (with and without kick-start) in scenario 1, the new algorithm was better in more cases than RAG-n for w <16.

v) Using AD in scenario 2, for 2x2 and 3x3 matrices, the new algorithm was best more often than RAG-n but the reverse was true for larger matrices. Average costs are shown in Figure 3.

4. CONCLUSION

We present for the first time a method of producing multiplier blocks specifically designed for matrix multiplication. Results are encouraging from this early work. Designing a small matrix, the best approach is to use the new algorithm presented here and RAG-n applied to columns, then choose the better design. Often the new algorithm's design will be best. Larger matrices seem to be best designed by RAG-n.

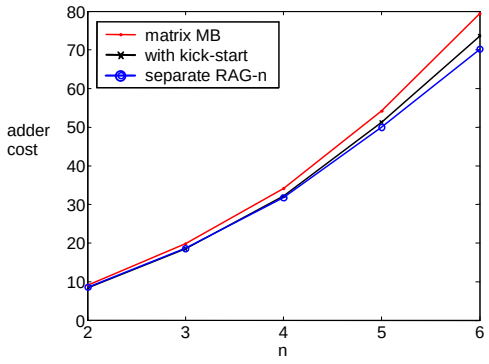

Figure 3 Adder cost versus *n* for *nxn* matrices – new algorithms versus RAG-n applied to columns. Costs are averaged over 20 matrices.

The new algorithm also gives a method of designing multiplier blocks for the transposed problem, which has not been addressed and is useful when considering the power consumption of multiplier blocks [7].

Further work that should be attempted is:
i) Continue to improve the algorithm so it will reliably outperform RAG-n. This includes examining how best to apply the "kick-start" (should it be the highest cost row; should the number of rows treated this way vary with matrix size), or inserting a large number of useful cost-1 adders *a la* the C1 algorithm [12].
ii).Examine whether the new algorithm is more efficient than BHM for vectors (FIR filters), due to its measure of distance to *any* coefficient.
iii) Examine the performance of the algorithm on non-square matrices. Performance should be better when rows outnumber columns (because this state can be achieved by transposing if there are more columns, this makes square matrices the hardest case).
iv) Examine the performance of the algorithm on large, sparse matrices.
v) Examine whether matrix decompositions can be used ahead of the algorithm.
vi) Explore the use of subexpression sharing in a new algorithm design.

5. ACKNOWLEDGEMENT

In this collaboration Coleman and Dempster were respectively supported by the Office of Naval Research (ONR), Arlington VA, USA and the ONR International Field Office in London.

References

[1] D R Bull and D H Horrocks, "Primitive operator digital filters", IEE Proceedings G, vol 138, no 3, pp401-412, Jun 1991

[2] A G Dempster and M D Macleod, "Constant integer multiplication using minimum adders", IEE Proceedings - Circuits, Devices and Systems, vol 141, no 5, pp407-413, Oct 1994

[3] A G Dempster and M D Macleod, "General algorithms for reduced-adder integer multiplier design", Electronics Letters, vol 31, no 21, pp1800-1802, Oct 1995

[4] O Gustafsson, A G Dempster and L Wanhammar, "Extended Results for Minimum-Adder Constant Integer Multipliers", Proc ISCAS 2002, ppI73-76

[5] A G Dempster and M D Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters" IEEE Trans Circuits and Systems II, vol 42, no 9, pp569-577, September 1995

[6] A G Dempster and M D Macleod, "Graphical Methods for Efficient Multiplier Design 2: Applications to Digital Filters", invited paper, Proc ECCTD99, vol 1, pp 261-264, September 1999.

[7] V A Bartlett and A G Dempster, "Using Carry-Save Adders in Low-Power Multiplier Blocks", proc ISCAS 2001, Sydney, May 6-9 2001, pp IV222-225

[8] R I Hartley, "Optimization of canonic signed digit multipliers for filter design", ISCAS91, pp1992-1995

[9] M Martinez-Peiro, E I Boemo, L Wanhammar, "Design of high-speed multiplierless filters using a nonrecursive signed common subexpression algorithm", IEEE Trans C&S II, 49 (3), pp 196 -203

[10] A G Dempster and N P Murphy, "Efficient Interpolators and Filter Banks using Multiplier Blocks", IEEE Trans Sig Proc, 48 (1), pp 257-261

[11] R E Crochiere and A V Oppenheim, "Analysis of Linear Digital Networks", Proc IEEE, vol 63, pp581-595, Apr 1975

[12] Dempster, A, S S Demirsoy and I. Kale, "Designing Multiplier Blocks with Low Logic Depth", *Proc ISCAS 2002*, ppV773-776