Mapping Tensor Products Onto VLSI Networks with Reduced I/O

A. Elnaggar       H. M. Alnuweiri       M. R. Ito

Department of Electrical Engineering, University of British Columbia, Vancouver, B.C., Canada, V6T1Z4

Abstract

This paper presents a methodology for designing folded VLSI networks for implementing tensor-product forms. Using tensor-products leads to very efficient expressions for a large number of computations in digital signal processing and matrix arithmetic. The resulting networks can trade-off total time delay with I/O bandwidth and chip area. The main goal is to parametrize the VLSI architecture so that it can be implemented under various packaging constraints including the available number of I/O pins, available chip-area, and certain restrictions on maximum wire length. Our methods result in folded VLSI networks with optimal $\mathcal{A}T^2$ trade-off for digital filtering and multidimensional transforms, where $A$ is the total area of the VLSI circuit (or chip) and $T$ is its total time delay.

1: Introduction

Highly parallel fine-grain computations can be naturally divided into alternating stages of data processing (computation) and communication. Processing is carried out by arithmetic/logic units or processing elements, and communication is realized by one or more stages of permutation networks. This strategy has been particularly useful in deriving very fast parallel architectures for multidimensional signal and image processing, arithmetic circuits, sorting and other problems [1,2,3,8,15,17]. However, the implementation of applications with a large number of inputs can be restricted by several physical attributes of the implementation medium. Such attributes include the limited number of I/O terminals per package (a VLSI chip, a multichip module, or a printed circuit board). Such limitations will continue to increase as the device features size continues to decrease. The main consequences of such limitations is that the particular design of parallel networks (particularly, interconnection networks) is shaped by specific packaging parameters (e.g. number of I/O pins and available layout area) and other physical specifications (e.g. wire length). In many cases, different parameters mandate significant changes in the original design or replacing the original network entirely by another design that better fits the specifications. To avoid the problem of redesigning parallel architectures whenever certain packaging parameters change, one must seeks a methodology that allows the design process to be parameterized so that a parallel network of a given size can be laid out in packages of various sizes by simple remapping.

Towards achieving this goal, this paper proposes a number of network folding techniques for designing VLSI circuits which have an optimal $\mathcal{A}T^2$ trade-off. The $\mathcal{A}T^2$ trade-off is the most useful complexity measure for VLSI networks, since it is based on information-transfer and wire-area arguments [17]. Network folding refers to the process of mapping a network with $N$ I/O terminal into a folded network with $N/Q$ inputs, where $Q$ is the I/O reduction (or scaling) parameter. Our methodology is based on a two-step design process. The first achieves optimality for logarithmic-depth VLSI networks by reformulating computations in tensor-product decompositions. The second step derives folded versions of these networks by separately folding the interconnection and parallel-arithmetic components of the original network. Our main goal is to provide a flexible mapping scheme for mapping a tensor expression of a given size onto a spectrum of VLSI networks that vary in a number of attributes such as I/O bandwidth, chip area, total time delay and maximum wire length. The mapping methodology results in VLSI networks whose various attributes are parametrized in the problem size $N$ (generally, $N$ is the number of I/O elements). The scaling parameter, $Q$, enables mapping an algorithm for a problem of a given size onto a wide range of architectures subject to packaging constraints and other technological specifications, without having to modify the algorithm to match each case.

Tensor-products (Kronecker products) have proven
to be useful in providing a unified decomposable matrix formulations for multidimensional transforms, convolutions, matrix multiplication, and several other fundamental computations. Recently, Granata et al [6] and Regalia et al [13] have shown that when coupled with permutation matrices, tensor-products provide a unifying framework for describing and programming a wide range of fast recursive algorithms for various transforms. They have also described techniques for manipulating tensor-product formulations of multidimensional transforms into forms suited to parallel processing machines and vector processing machines.

In this paper we extend the application of the techniques presented in [6,13,16,18], by describing several methodologies for mapping tensor-product formulations onto efficient VLSI architectures. The mapping method is based on establishing direct correspondence between tensor-product forms and VLSI layouts. Specific network topologies are determined from number of factors including the size of matrices, type of tensor operations, and type of permutation matrices. It will be shown that expressions for the number of processing units, chip area, time delay, and I/O bandwidth, can be derived directly from the tensor equations and the selected architectural class.

2: Preliminaries

2.1: Tensor-Product Decompositions

The tensor-product is a binary matrix operator which provides a mechanism for multiplying two matrices to form a single larger matrix. Once the tensor-product factorization of a large matrix has been derived, the symmetries identified in the factorization can be used to derive an efficient algorithm for a large number of matrix operations such as matrix multiplication, matrix inversion, etc. Since matrix operations form the core of any digital signal processing system, tensor-product formulations proved to be a powerful tool for designing and implementing DSP algorithms.

Let $A_{R,S}$ and $B_{M,N}$ be two arbitrary matrices of dimension $R \times S$ and $M \times N$, respectively.

The tensor-product of $A$ and $B$, $C = A \otimes B$ which is a matrix of dimension $RM \times SN$ defined by

$$ C = \begin{bmatrix} a_{0,0}B & a_{0,1}B & \cdots & a_{0,S-1}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{R-1,0}B & \cdots & \cdots & a_{R-1,S-1}B \end{bmatrix} $$

(1)

It is possible to think of the tensor-product as a matrix decomposition. Given a large matrix $C$, it could be expressed in terms of two smaller matrices $A$ and $B$, called tensor matrices in the form

$$ C = A \otimes B $$

(2)

Two particular forms of tensor-products have been identified in [6,13,16] for use with parallel and pipelined (vector) machines. The first of these is the parallel form described by the following tensor-product,

$$ C_n = (I_s \otimes B_r) $$

$$ = \begin{bmatrix} B_r & 0 \\ B_r & B_r \\ 0 & \ddots & \ddots & \ddots \\ & 0 & \ddots & \ddots & \ddots & B_r \end{bmatrix} $$

(3)

where $I_s$ is the identity matrix of order $s$, and $B_r$ is an $r \times r$ matrix. The resulting matrix $C$ is a block-diagonal of order $n = rs$. In many DSP computations, the above decomposition is applied to an input vector $X_n$ of dimension $n = rs$. If the vector $X_n$ is split into $s$ sub-vectors $x_i$, $1 \leq i \leq s$ each of dimension $r$, then we can perform the matrix vector product $B_r x_i$ in parallel on all $s$ sub-vectors which described by,

$$ Y_n = C_n X_n = (I_s \otimes B_r) X_n $$

$$ = \begin{bmatrix} B_r x_1 \\ B_r x_2 \\ \vdots \\ 0 \end{bmatrix} $$

(4)

The second special tensor-product form is the vector form described as follows

$$ Y_n = C_n X_n = (A_r \otimes I_s) X_n $$

$$ = \begin{bmatrix} a_{0,0}I_s & a_{0,1}I_s & \cdots & a_{0,r-1}I_s \\ \vdots & \vdots & \vdots & \vdots \\ a_{r-1,0}I_s & a_{r-1,1}I_s & \cdots & a_{r-1,r-1}I_s \end{bmatrix} X_n $$

(5)

where $A_r$ is an $r \times r$ matrix, $Y_n$ and $X_n$ are two vectors of dimension $n$. The matrix vector multiplication in (5) has another form which is useful for VLSI implementation. This form is derived by replacing the matrix vector multiplication by two matrices multiplications as follows

$$ \begin{bmatrix} y_0 & y_1 & \cdots & y_{s-1} \\ y_s & y_{s+1} & \cdots & y_{2s-1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{rs-2} & y_{rs-1} & \cdots & y_{rs-1} \end{bmatrix} = $$

$$ \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,r-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,r-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r-1,0} & a_{r-1,1} & \cdots & a_{r-1,r-1} \end{bmatrix} \begin{bmatrix} x_0 & x_1 & \cdots & x_{s-1} \\ x_s & x_{s+1} & \cdots & x_{2s-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{rs-2} & x_{rs-1} & \cdots & x_{rs-1} \end{bmatrix} $$

i.e.

$$ Y_{s,r}^T = A_r X_{s,r}^T $$

or

$$ Y_{s,r} = X_{s,r} A_r^T $$

(6)

Where $X_{s,r}$ is a rearrangement of vector $X_n$ into matrix of dimension $s \times r$. Similarly $Y_{s,r}$ is a rearrangement of vector $Y_n$ into matrix of dimension $s \times r$.  

151
2.2: Permutation Matrices

As mentioned earlier, permutation matrices play a central role when combined with the tensor-products [13,16,18]. The permutation matrix, P, is a special square matrix whose entries are zeroes and ones. Such that each row or column of P has a single 1 entry. If n=r*s is the dimension of an input vector X_n, then the effect of the permutation matrix P_{n,s} on X_n is to shuffle X by grouping all the r elements separated by distance s together. The first r element will be x_{1s}, x_{2s}, x_{3s}, \ldots, x_{(r-1)s}. The next r elements are thus x_{1+qs}, x_{1+2qs}, \ldots, x_{1+(r-1)qs} and so on.

Let A_r and B_s be any square matrices of order r and s, respectively, and n=rs. In general, the tensor-product of two matrices is not commutative. However, the order of the product can be reversed using stride permutations [6,13,16,18] as in the following equation,

\[(A_r \otimes B_s) = P_{n,s}(B_s \otimes A_r)P_{n,r}\]  

(8)

This is an important tool for converting tensor parallel-forms into vector-forms and vice versa, as follows,

\[(I_r \otimes B_s) = P_{n,r}(B_s \otimes I_r)P_{n,s}\]  

(9)

\[(B_s \otimes I_r) = P_{n,s}(I_r \otimes B_s)P_{n,r}\]  

(10)

For example, the general form

\[Y_n = (C_n)X_n = (A_r \otimes B_s)X_n\]  

(11)

can be expressed in a purely parallel-form as follows

\[Y_n = (C_n)X_n = [P_{n,r}(I_s \otimes A_r)P_{n,s}](I_r \otimes B_s)X_n\]  

(12)

or in a purely vector-form as follows

\[Y_n = (C_n)X_n = (A_r \otimes I_s)[P_{n,r}(B_s \otimes I_r)P_{n,s}]X_n\]  

(13)

3: Mapping Multi-Dimensional Decompositions onto VLSI

For DSP applications, tensor-product formulations can be generalized for multi-dimensional processing [16]. These of three forms are very well suited mapping parallel VLSI architectures and will be discussed here.

3.1: Networks for the Multi-Dimensional Parallel Form (MDPF)

Let \(N = N_1N_2\cdots N_k\) where \(N_1, N_2, \ldots, N_k\) be positive integers, the MDPF is

\[A_{N_1} \otimes A_{N_2} \otimes \cdots \otimes A_{N_k} = \prod_{i=1}^{k} P_{N_iN_i}(I_{N_iN_i} \otimes A_{N_i})\]  

(14)

where \(A_{N_i}\) is an \(N_i \times N_i\). This form can be realized by a cascade of \(k\) network stages, in which stage \(i\) performs

\[P_{N_iN_i}(I_{N_iN_i} \otimes A_{N_i})X_N\]  

(15)

where \(X_N\) is either the input vector of dimension \(N\) or the output vector from the previous stage. This network stage is shown in Figure 1, where each of the \(N/N_i\) blocks at the input is an Orthogonal-Trees Networks (OTN) of size \(N_i \times N_i\) [9,11]. The area of each such OTN is \(O(N_i^2\log^2 N_i)\), and each is capable of performing the matrix-vector product \(A_{N_i}X_i\) (i.e. an \(N_i \times N_i\) matrix times an \(N_i\)-dimension vector) in \(O(\log N_i)\) time. The \(N/N_i\)-shuffle permutation realizes the permutation \(P_{N_iN_i}\). It can be shown that, if \(k\) is a constant, then the total area of the \(k\) stages is \(O(N^2)\) and the total delay is \(O(\log N)\) [5].

3.2: Reduced I/O and Network Folding for the Parallel Form

To cope with practical limitations on the area and I/O bandwidth of logarithmic-depth VLSI networks that implement the parallel form, provisions must be provided for trading off time with I/O and/or area. In this section we described network folding techniques which map an MDPF logarithmic-depth VLSI network with \(N\) inputs, into a folded networks which are characterized by two parameters \(N\) (problem size) and \(Q\) (I/O reduction parameter). The folded network has \(N/Q\) I/O pins, area \(A = O(A_N/Q^2)\), and time delay \(T = QT_N\), where \(A_N\) and \(T_N\) are, respectively, the area and time delay of the \(N\)-input logarithmic depth VLSI circuit. As an example, a folded MDPF VLSI network will have area \(O(N^2/Q^2)\) and the time delay \(O(Q\log N)\) [5]. The parameter \(Q\) can be varied over the entire range \([1, N]\). However, limited fan-in and area lower bounds restricts the range of \(Q\) to \([1, \sqrt{N}/\log N]\) for a large classes of problems [4,8].
3.3: Network Folding

We start by reviewing an index mapping scheme that results in simple folded networks for stride permutations. The main goal is to map a stride permutations network of size $N = 2^n$ (i.e. with $N$ I/O terminals) into an equivalent network with $N/Q$ I/O terminals. This mapping will be called folding, and $Q$ will be called the I/O reduction factor. A folded stride permutation network has $N/Q$ I/O terminals. Thus, bits from at most $N/Q$ elements can be input or output at a time through the folded network. The index-mapping procedure views the $N$-element input or output vectors as a matrix $Q$ columns with $N/Q$ elements per column. Specifically, let $M = N/Q$, then an $N$-element vector $V_i(N) \in F^N$ can be arranged into an $M \times Q$ matrix $V_{M \times Q} \in F^{M \times Q}$ as follows:

$$V_{M \times Q} = \begin{bmatrix}
v_0 & v_1 & \cdots & v_{N-M} \\
v_1 & v_{M+1} & \cdots & v_{N-M+1} \\
\vdots & \vdots & \ddots & \vdots \\
v_{M-1} & v_{2M-1} & \cdots & v_{N-1}
\end{bmatrix}$$ (16)

In other words, column $i$, $0 \leq i \leq Q - 1$, of the above matrix contains elements with indices $i_1, i_2 \in \{0, 1, \cdots, (i+1)N/Q - 1\}$. The mapping (permutation) of elements from the input vector to the output vector can now be expressed as a mapping of elements from the input matrix to the output matrix. For stride permutation, each index in the output matrix is obtained by performing the specified bit-permute operations on the bit representation of the corresponding index (i.e. the index at the same position) in the input matrix. An example of an $S_{6}(64,4)$ network derived by our method is shown in Figure 2, where $S_{N/M}(N, Q)$ denotes a folded $N/M$-Shuffle network of stage $i$ (with reduction factor $Q$). Note that, the folded network consists of $N/Q^2$ sub-networks, each of which is a $Q \times Q$ transpose circuit. Figure 3 shows bit-serial and 2-bit-at-a-time (2-BAAT) versions of the transpose circuit with a word length $w = 6$. Details of derivation can be found in the full version of this paper [5].

Reducing the area of the OTN networks stage (shown in Figure 1) can be done using the following procedure. First, the notation $V_i(N, Q)$ will be used to denote the network realizing (14), where $Q$ is the I/O reduction factor. Also $V_i(N, Q)$ will be used to realize (15). Folding $V_i(N, 1)$ (i.e. stage $i$ of the network $V(N, 1)$ which realizes form (14)) can be done by using $N/Q$ copies of $V_i(N_1, 1)$ and using a folded $S_{N/M}(N, Q)$ permutation network. A complete example $V(N, 1)$, $N = 16$, factored as $N = N_1N_2N_3 = 4 \times 2 \times 2$ and the corresponding folded version, $V(N, 2)$ are shown in Figure 4. Note that, $V_i(N_1, 1)$ is identical structure to $V_i(N, 1)$ except that it has $N_i$ inputs [3].

It is possible to show that the area of $V(N, Q)$ is $O(N^2/Q^2)$ and its time delay is $O(Q \log N)$ which gives optimal $AT^2 = O(N^2 \log^2 N)$ [5].

3.4: Networks for the Multi-Dimensional Vector Form (MDVF)

Another useful form for VLSI implementation is the Multi-Dimensional Vector Form (MDVF) defined as follows [16]

$$A_{N_i} \otimes A_{N_2} \otimes \cdots \otimes A_{N_k} = \prod_{i=1}^{k} (A_{N_i} \otimes I_{N_i}) P_{N_i,N_i}$$ (17)

If $X_N$ is either an $N$-dimensional input vector to the circuit or the output from the previous stage, then one stage of this multi-stage form performs the following computation,

$$\left( A_{N_i} \otimes I_{N_i} \right) P_{N_i,N_i} X_N$$ (18)

from (6), we can rewrite (18) as

$$A_{N_i} X_{i_1,i_2}$$ (19)
version of the Preparata-Vuillemin network. A detailed description of this network will appear in the full version of this paper [5]. The lower bound for multiplying two matrices in the bit-model is \( AT^2 = (n^4w^2) \) where \( w \) is the word length of the matrix elements [7]. In our model a systolic array of dimensions \( r_1 \times r_2 \) is used such that \( l_1/r_1 = l_2/r_2 \) where \( l_1 = N_1 \) and \( l_2 = N/N_1 \) to guarantee that the two matrices are square. We can show in [5] that, if the word length is \( w = \log N \), then the network achieves the optimal \( AT^2 = O(N^2\log^2 N) \).

### 3.6: Networks for the Mixed Form

A more general form of tensor-product decomposition is the mixed-form defined by the following equation,

\[
A_{N_1} \otimes A_{N_2} \otimes \cdots \otimes A_{N_k} = \prod_{i=1}^{k} I_{N(i-1)} \otimes (A_{N_i} \otimes I_{N(i)} \otimes I_{N(i)})
\]

where \( N(i) = N_1 N_2 \cdots N_i \) and \( N(0) = 1 \). The mixed form is a general form from which the parallel and the vector forms can be derived [13,16]. Thus a network realizing the above mixed-form will consist of \( k \) stages, where stage \( i \) performs

\[
I_{N(i-1)} \otimes (A_{N_i} \otimes I_{N(i)} \otimes I_{N(i)}) X
\]

This can be viewed as a parallel composition of \( N(i-1) \) vector-forms as shown in Figure 6. The complete network of the mixed form is shown in Figure 7.

### 3.5: Reduced I/O and Area-Time Trade-offs for the Vector Form

The VLSI lower-bound \( AT^2 = \Omega(n^4) \) for multiplying two \( n \times n \) matrices has been established in [14] for the word-model. To achieve this bound, Preparata and Vuillemin [12] used an \( r \times r \) array of modules, each of which is an \( (n/r) \times (n/r) \) recursive matrix-multiplier (based on one stage of multiplier and a multiplicity of addition trees). Here, we use a bit-model

---

1 Actually, the lower bound holds for matrix-squaring which is a special case of the general matrix-multiplication problem.
4: Conclusion

In this paper we presented a flexible mapping scheme for mapping a tensor expression of a given size onto a spectrum of VLSI networks that vary in I/O bandwidth, chip area, total time delay and maximum wire length. The mapping methodology results in VLSI networks whose various attributes are parametrized in the problem size N and the scaling parameter Q. This enables mapping an algorithm for a problem of a given size onto a wide range of architectures subject to packaging constraints and other technological specifications, without having to modify the algorithm to match each case. Also, expressions for the number of processing units, chip area, time delay, and I/O bandwidth, can be derived directly from the tensor equations and the selected architectural class [5].

Acknowledgment

This research was supported in part by the NSERC operating grants No. 589054 and No. 581260.

5: References