SYNTHESIS OF FOLDED, PIPELINED ARCHITECTURES FOR MULTI-DIMENSIONAL MULTIRATE SYSTEMS*

Vijay Sundararajan *Keshab K. Parhi*
Dept. of ECE, University of Minnesota
E-mail: vijay@ee.umn.edu, parhi@ee.umn.edu

ABSTRACT

Motivated by the need for designing efficient architectures for two-dimensional discrete wavelet transforms (DWTs), this paper presents a novel multi-dimensional (MD) folding transformation technique which can be used to synthesize control circuits for pipelined architectures for a specific class of multirate MD digital signal processing (DSP) algorithms. Although a multirate MD DSP algorithm contains decimaters and expanders which change the effective sample rate of a MD discrete time signal, MD folding time-multiplexes the algorithm to hardware in such a manner that the resulting synchronous architecture requires only a single clock signal for the clocking of the datapath. Feasibility constraints are derived for folding a 2-D data-flow graph (DFG) onto a given set of hardware functional units according to a specified schedule. Area/power efficient architectures are derived for 1-4 level 2-D discrete wavelet transforms (DWT) with 18.5-23.3% savings in storage area.

1. INTRODUCTION

The folding transformation [1] maps a behavioral description of an algorithm in the form of a data-flow graph (DFG) [2] to a set of hardware functional units according to a specified static schedule [1]. The goal of this mapping is to achieve a solution that optimizes a given objective function defined in a 3-D design space consisting of throughput, power and area. While folding of a DFG has been completely formulated for 1-D systems [1], to the best of our knowledge it remains unsolved for multi-dimensional (MD) multirate systems. Folding in 1-D systems has resulted in extremely efficient architectures, see, [1]. An extension of folding to higher dimensions is needed for design of efficient manual/automated design of hardware for multi-dimensional DSP algorithms, described by a multi-dimensional data-flow graph (MDFG), such as algorithms for image processing. This research was motivated by design of efficient architectures for 2-D discrete wavelet transform which is a multirate algorithm.

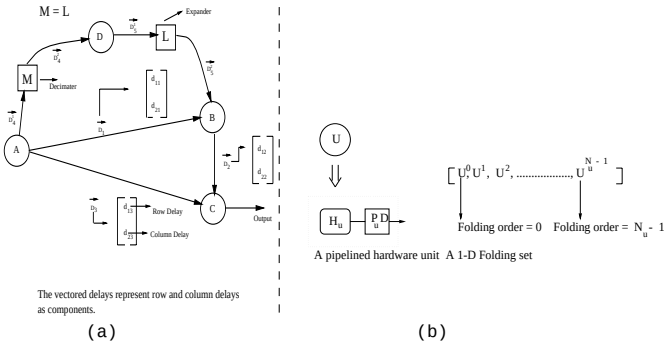

Figure 1. (a)A 2-D DFG, (b)DFG nodes mapped to a hardware unit H_u.

2. FOLDING

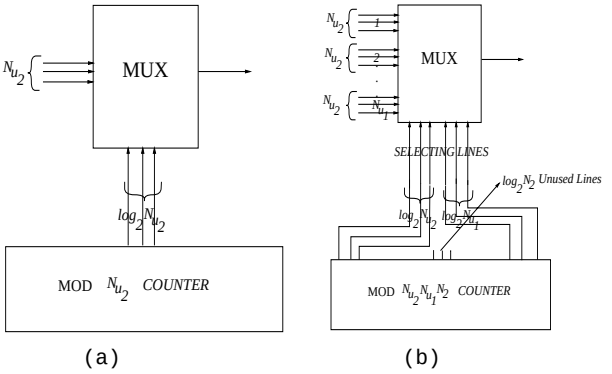

Figure 2. (a) A 1-D Folding control structure, (b) A 2-D Folding control structure.

We consider two paradigms for folding. The first one performs a so called 1-D folding. The control structure for this is shown in Fig. 2(a). Let every input line to the multiplexor in Fig. 2(a) be fed by the output of the corresponding node in the folding set shown in Fig. 1(b). This mechanism can be used for periodic multiplexing of the operations in the folding set in Fig. 1(b) according to a static periodic schedule given by the position of the DFG nodes in the folding set. Unfortunately such a control structure is not useful for folding of two and higher dimensional multirate data-flow graphs, see Fig. 1(a) for an example of a 2-D DFG. This fact can be shown quite easily and will not be discussed further.

2.1. 1-D to 2-D Folding

2.1.1. Control Structures for Folding

In the control structure shown in Fig. 2(b), the input lines to the multiplexor which are $N_{u_1} \times N_{u_2}$ in number have been divided into N_{u_1} sets of N_{u_2} lines each. Due to the $log_2(N_2)$ [1] unused lines of the counter driving the MUX select lines the schedule for input lines connected to the output does not have a simple periodicity. Here, N_2 refers to the number of columns that an input to a node folded using this control structure will have. [1] The overall

*This work was supported by the Army Research Office under grant number DA/DAAH-94-G-0405.

[1] Restriction of N_{u_1}, N_{u_2}, N_2 to powers of 2 is not required but is used for ease of illustration.

periodicity of this structure is $N_{u_1} \times N_{u_2} \times N_2$. Assuming that the counter starts at count 0, for the first $N_{u_2} \times N_2$ time units the structure schedules the first N_{u_2} set of input lines periodically, i.e., it behaves exactly like a control structure for 1-D folding. From time unit $N_{u_2} \times N_2$ to $2 \times N_{u_2} \times N_2$ the next set of N_{u_2} input lines are periodically scheduled. Similarly, from $(i-1) \times N_{u_2} \times N_2$ to the $i \times N_{u_2} \times N_2$ time unit the i^{th} set of N_{u_2} input lines is scheduled. Such a control structure forms a 2-D folding control structure. The list of inputs to such a 2-D folding control structure can be arranged in the form of a $N_{u_1} \times N_{u_2}$ matrix with successive rows representing successive sets of N_{u_2} nodes mapped to the input lines of the multiplexor. This matrix constitutes a 2-D folding set. The folding order of a node will be a 2-D vector whose components are the row and column number of that node in the 2-D folding set. Now we make a very important claim which the reader can verify with a little exercise; the $(n_1, n_2)^{th}$ iteration of a node with folding order (u_1, u_2) will be executed at time-unit t_{n_1,n_2} given by:

$$t_{n_1,n_2} = N_{u_1} N_{u_2} N_2 n_1 + N_{u_2} N_2 u_1 + N_{u_2} n_2 + u_2. \quad (1)$$

Assume that a hardware unit to which this node is mapped has P_u pipelining stages and each stage requires 1 clock cycle for execution. The output from the MUX drives either a circulating buffer or a (SRAM/DRAM) module which has a READ access time P_{READ} and a WRITE access time P_{WRITE}. The result of the $(n_1, n_2)^{th}$ iteration of such a node is, therefore, available for use at time unit $N_{u_1} N_{u_2} N_2 n_1 + N_{u_2} N_2 u_1 + N_{u_2} n_2 + u_2 + P_u^{'}$, where $P_u^{'} = P_u + P_{READ} + P_{WRITE}$.

It is possible to devise three and higher dimensional folding control structures by a straightforward extension of these ideas.

2.1.2. 2-D Folding

The primary objective of folding is to ensure that the scheduling does not violate any dependencies that the MDFG might have. Folding of 2-D DFG arcs with diagonal decimaters and diagonal expanders are illustrated. The single rate DFG arc case is just a special case of these two cases with either decimation or expansion matrix set equal to the identity matrix. Throughout this section we assume that a source node of a DFG arc has folding order (u_1, u_2) and folding set of dimension $N_{u_1} \times N_{u_2}$ and a destination node has folding order (v_1, v_2) and folding set of dimension $N_{v_1} \times N_{v_2}$. The source node is assumed to be mapped to hardware unit H_u and the destination node to H_v.

2.1.3. Folding for a Decimating DFG Arc

A Decimating DFG arc is shown in Fig. 3(a). The defining relation for such an arc is:

$$y(n_1, n_2) = x(M_1(n_1 - d_{12}) - d_{11}, M_2(n_2 - d_{22}) - d_{21}). \quad (2)$$

For ease of implementability of control structures the folded delays must be iteration independent. This constraint leads to $N_{v_1} = M_1 N_{u_1}$ and $N_{v_2} = M_2 N_{u_2}$. The number of delays in the folded arc from U to V can be derived to be:

$$\begin{aligned} D_F^D &= (M_1 d_{12} + d_{11}) N_{u_1} N_{u_2} N_2 + (v_1 - u_1) N_{u_2} N_2 + \\ &\quad (M_2 d_{22} + d_{21}) N_{u_2} + v_2 - u_2 - P_u^{'}. \end{aligned} \quad (3)$$

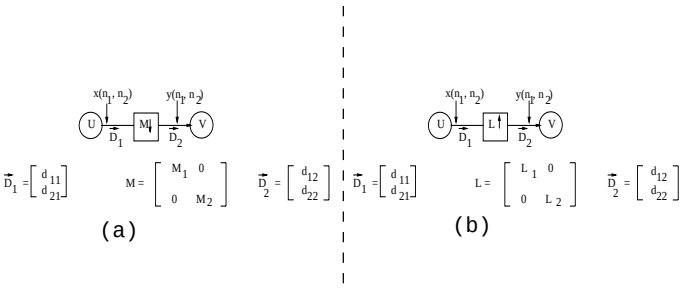

Figure 3. A multirate arc of a 2-D DFG with (a) A diagonal decimater, (b)A diagonal expander.

2.1.4. Folding for an Expanding DFG Arc

The defining relation for such an arc, see Fig. 3(b) is:

$$x(n_1, n_2) = y(L_1(n_1 + d_{11}) + d_{12}, L_2(n_2 + d_{21}) + d_{22}). \quad (4)$$

Like before for iteration independence of D_F^E we require $L_1 N_{v_1} = N_{u_1}$ and $L_2 N_{v_2} = N_{u_2}$. The number of delays in the folded arc can be derived to be:

$$\begin{aligned} D_F^E = (L_1 d_{11} + d_{12}) N_{v_1} N_{v_2} (L_2 N_2) + v_1 N_{v_2} (L_2 N_2) - u_1 N_{u_2} N_2 + \\ n_2 (L_2 N_{v_2} - N_{u_2}) + (L_2 d_{21} + d_{22}) N_{v_2} + v_2 - u_2 - P_u^{'}. \end{aligned} \quad (5)$$

2.2. Retiming for folding

Retiming for folding is the process of retiming a DFG so that the folded delays in any folded arc is nonnegative. For 1-D case see [1]. For an excellent review of 2-D retiming the reader is referred to [3] and [4]. Throughout this section we assume that we are folding an arc from node U to node V and that node U has a retiming vector $(r(u_1), r(u_2))$ and node V has a retiming vector $(r(v_1), r(v_2))$. Once again the single-rate case is considered as a special case of the multirate cases.

2.2.1. Arcs with decimaters

The retiming constraint for folding a decimating arc can be easily derived and is given by:

$$r(u_1) N_{u_1} N_2 + r(u_2) - M_1 r(v_1) N_{u_1} N_2 - M_2 r(v_2) \leq \left\lfloor \frac{D_F^D}{N_{u_2}} \right\rfloor. \quad (6)$$

2.2.2. Arcs with expanders

The retiming constraint for folding an expanding arc can be easily derived and is given by:

$$L_1 L_2 r(u_1) N_{u_1} N_2 + L_2 r(u_2) - L_2 r(v_1) N_{v_1} N_2 - r(v_2) \leq \left\lfloor \frac{D_F^E}{N_{v_2}} \right\rfloor. \quad (7)$$

2.2.3. Single Rate Arcs

The retiming inequalities for single rate arcs have the same form as (7) with $L_1 = L_2 = 1$ and $D_F^E = D_F^S$ with $d_{12} = d_{22} = 0$.

2.3. Retiming for storage reduction

Another objective of retiming is to minimize the storage requirements for implementing the folded architecture. A very good approximate linear modeling is made possible by using an artifice developed in the classic paper on retiming by Leiserson *et.al.* [5]. This problem can be easily verified to be a hard ILP problem [6]. We therefore resort to separable retiming detailed next.

3. SEPARABLE RETIMING FORMULATION

Note first that the following pairs of inequalities are sufficient to guarantee feasibility of folding for any DFG arc A.

$$\begin{aligned} (Row)C_u r(u_1) - c_v r(v_1) &\leq D_A^F, \\ C_u = N_{u_1}N_{u_2}N_2, &\quad C_v = N_{v_1}N_{v_2}N_2, \\ (Column)N_{u_2} r(u_2) - N_{v_2} r(v_2) &\leq D_D^F - C_u r(u_1) + c_v r(v_1). \end{aligned}$$

The above inequalities are in generic form, if A = S we get the single-rate arc case, for A = D we get the decimating arc case and for A = E we get the expanding arc case. Doing this for every DFG arc we get two sets of equations, namely, the row constraints and the column constraints. Note that the row constraints are solved initially to get a value for the row retiming variables; this fixes the right hand side for the column constraints which are solved for subsequently to get a value for the column retiming variables.

Next we discuss the memory modeling strategy for the problem at hand. The cost of any arc, under row and column retiming, without fanout is shown in Table 1.
For nodes with fanout, at any instance of time, we would like to provide storage for only as many *live signal instances* as are present in the output arc with the maximum number of *live signal instances*. The memory modeling required for such cases is well known and the reader is referred to [5] for further details.

4. STORAGE COMPUTATIONS

Life-time analysis [7] is used to compute the minimum amount of storage required to implement the architecture obtained as a result of the folding scheme described in this paper. The technique employed to do this will be presented next.

4.1. Single Rate Arcs

At any time, R, there are, $\stackrel{DEF}{=} r_U(R)$, live samples produced by the node U. The maximum of $r_U(R)$ over all values of R gives us the minimum number of delays required to fold all arcs out of node U. Any instance, R, can be expressed in the following way:

$$\begin{aligned} R &= K_1 N_{u_1} N_{u_2} N_2 + K_2 N_{u_2} N_2 \\ &\quad + K_3 N_{u_2} + K_4 + P_{u_0} \\ P_{u_0} &= u_1 N_{u_2} N_2 + u_2 + P_u^{'}, \\ 0 \leq K_2 &\leq N_{u_1} - 1, \\ 0 \leq K_3 &\leq N_2 - 1. \end{aligned} \quad (8)$$

A comparison of (8) with the expression for the time when the $(n_1, n_2)^{th}$ iteration of node U is scheduled shows that K_1 gives us the minimum number of rows of output produced by node U. If $K_2 > 0$, this implies that the row of nodes consisting of node U is not being scheduled at instant R, then the total number of rows output by node U actually is $K_1 + 1$ otherwise, it is K_1. In case $K_2 = 0$, implying that the row of nodes consisting of node U is being scheduled at time instant R, then the total number of instances of output produced by node U is K_1 Rows + K_3 signal instances. Since $0 \leq K_2 \leq N_{u_1} - 1$, using the last observation, we can concisely express the total output from node U, out_U as:

$$out_U = N_2^u (K_1 + \left\lceil \frac{K_2}{N_{u_1}} \right\rceil) + (1 - \left\lceil \frac{K_2}{N_{u_1}} \right\rceil) K_3. \quad (9)$$

K_1, K_2 and K_3 can be easily computed. The number of signal instances consumed by sink node V, con_V, can be similarly calculated by observing that any instance, R, could be alternately expressed as:

$$\begin{aligned} R &= J_1 N_{v_1} N_{v_2} N_2 + J_2 N_{v_2} N_2 + J_3 N_{v_2} + J_4 + P_{v_1} \\ 0 \leq J_2 &\leq N_{v_1} - 1, \\ 0 \leq J_3 &\leq N_2 - 1, \end{aligned} \quad (10)$$

where P_{v_1} is the time instant when the first iteration of node V is scheduled. The number of signal instances consumed by node V can be calculated as before and we get:

$$con_V = N_2 (J1 + \left\lceil \frac{J2}{N_{v_1}} \right\rceil) + (1 - \left\lceil \frac{J_2}{N_{v_1}} \right\rceil) J_3. \quad (11)$$

Also, note that $P_{v_1} = D_S^F + P_{u_0}$. The number of *live signal instances* produced by node U into the given arc is then given by $out_U - con_V$. By maximizing this over an interval $N_{u_1}N_{u_2}N_2$ (the expression for *live signal instances* is periodic) we get the exact storage requirements for folding the given single rate arc. The case of decimating and expanding arcs can be treated similarly and will not be discussed here. The total memory required to fold the entire DFG is then found by maximizing the sum over all nodes of the memory required to fold that outgoing arc which has a maximum of live signal samples at any instance.

5. DESIGN EXAMPLES

The design of a folding set will be governed primarily by the input schedule. The exact binding of operation nodes to hardware units can take advantage of various power saving strategies like input correlation and operation locality; these techniques are vital for power savings in implementations with heavy resource sharing [8], [9]. Our first example is a 1-level 512 × 512 2-D DWT with 4-tap filters, whose folding set is designed by taking the above factors into consideration. In fact the folding set of each multiplier consists of nodes having a common input. This makes it possible to eliminate these multiply operations completely at the expense of shifts and adds and get enormous savings in power as compared to other implementations. The folding set for the adders is designed in such a way that node clusters in the DFG get mapped to physical clusters in the architecture. This can have a major impact on power savings in routing and in shared busses [9]. We assume a 2-stage pipeline for the multipliers and a 1-stage pipeline for the adders.

The 2-D DFG of a 1-level 2-D DWT along with a low-power folding set is shown in Fig. 4. The folded architecture is shown in Fig. 5. It uses two dual port (1 READ and 1 WRITE) 512 word SRAM/DRAM and two dual port 256 word SRAM/DRAM and 35 registers organized variously as circulating buffers. This represents around 23.3% savings in storage area as compared to the best known implementation in [10]. We tabulate the comparison of a 4-level 2064×2064 2-D DWT with 4-tap filters. In our method we could either use 4 different rate clocks for the four different levels and hence some extra multipliers and adders or we get a single clock solution. The comparison in Table 2 suggests that the multi-clock solution could be preferable in terms of area. Our new folding approach generates superior architectures as compared with previously published ones.

Table 1. Modeling of arc cost for computing objective function in row and column retiming.

Arc type	d_{11}^r	d_{21}^r	d_{12}^r
S-type	$r(v_1) - r(u_1) + d_{11}$	$r(v_2) - r(u_2) + d_{21}$	-
D-type	$d_{11} - M_1 \left\lfloor \frac{d_{11}}{M_1} \right\rfloor$	$d_{21} - M_2 \left\lfloor \frac{d_{21}}{M_2} \right\rfloor$	$r(v_1) - \frac{r(u_1)}{M_1} + \left\lfloor \frac{d_{11}}{M_1} \right\rfloor + d_{12}$
E-type	$\frac{r(v_1)}{L_1} - r(u_1) + \left\lfloor \frac{d_{12}}{L_1} \right\rfloor + d_{11}$	$\frac{r(v_2)}{L_2} - r(u_2) + \left\lfloor \frac{d_{22}}{L_2} \right\rfloor + d_{22}$	$d_{12} - L_1 \left\lfloor \frac{d_{12}}{L_1} \right\rfloor$
Arc type	d_{22}^r	Cost for Row-retiming	Cost for Col-retiming
S-type	-	$d_{11}^r N_2^u$	d_{21}^r
D-type	$r(v_2) - \frac{r(u_2)}{M_2} + \left\lfloor \frac{d_{21}}{M_2} \right\rfloor + d_{22}$	$d_{12}^r N_2^v$	d_{22}^r
E-type	$d_{22} - L_2 \left\lfloor \frac{d_{22}}{L_2} \right\rfloor$	$d_{11}^r N_2^u$	d_{21}^r

Table 2. Comparison of various wavelet architectures for 4-level 2064 × 2064 2-D DWT.

Technique	mult.	add.	storage	storage saved
Systolic[10]	16	12	16512	-
RAM based[10]	16	12	16512	-
Distributed[10]	16	12	16512	-
Area Efficient[10]	16	12	16512	-
Ours:(1 clock)	16	12	13452	18.5%
Ours:(4 clocks)	32	24	12680	23.3%

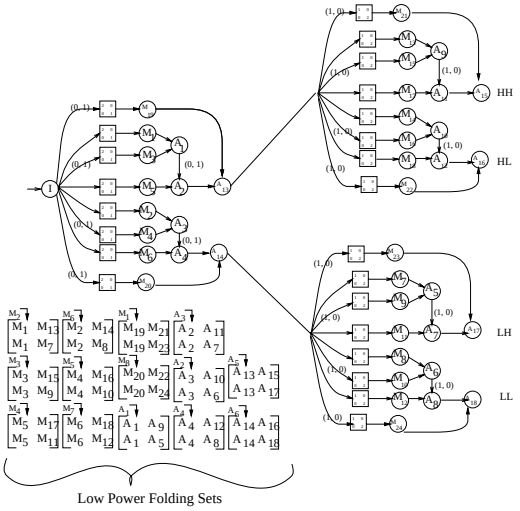

Figure 4. DFG for the 2-D DWT along with the 2-D folding set.

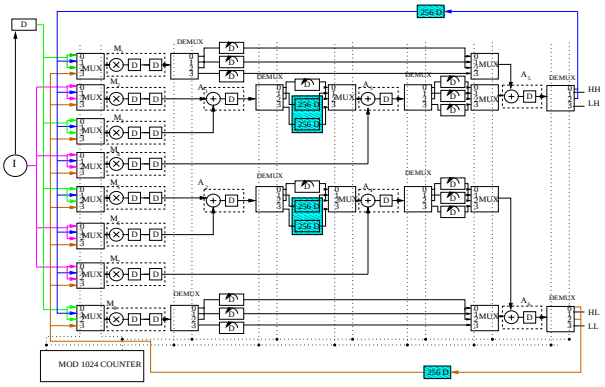

Figure 5. Folded Architecture for 2-D DWT.

6. CONCLUSIONS AND FUTURE WORK

An important new design technique was presented for high level synthesis of MD multirate DSP systems implemented with a row-by-row input scan order. An extension of these techniques to include more complex input scanning orders and decimating/expanding operators is currently under investigation.

REFERENCES

[1] K. K. Parhi, C. Y. Wang, and A. P. Brown, "Synthesis of Control Circuits in Folded Pipelined DSP architectures," *IEEE Journal of Solid State Circuits*, vol. 27, no. 1, pp. 29–43, 1992.

[2] E. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, pp. 1235–1245, 1987.

[3] T. Denk, M. Majumdar, and K. K. Parhi, "Two-Dimensional Retiming with Low Memory Requirements," *ICASSP-96*, vol. 6, pp. 3330–3333, May. 1996.

[4] N. Passos, E. H. M. Sha, and S. Bass, "Schedule-Based Multi-dimensional Retiming on Data Flow Graphs," *Proceedings of the 8th International Parallel Processing Symposium*, pp. 195–199, April. 1994.

[5] C. E. Leiserson and J. B. Saxe, "Retiming Synchronous Circuitry," *Algorithmica*, vol. 6, no. 1, pp. 5–35, 1991.

[6] V. Zivojnovic and R. Schoenen, "On Retiming of Multirate DSP Algorithms," *ICASSP-96*, vol. 6, pp. 3310–3313, May. 1996.

[7] K. K. Parhi, "Systematic synthesis of DSP data format converters using life-time analysis and forward-backward register allocation," *IEEE Trans. Circuits And Systems II Analog and Digital Signal Processing*, vol. 39, pp. 423–440, July. 1992.

[8] A. P. Chandrakasan and R. W. Brodersen, "Minimizing Power Consumption in Digital CMOS Circuits," *Proceedings of the IEEE*, vol. 83, pp. 498–523, April. 1995.

[9] R. Mehra, L. M. Guerra, and J. Rabaey, "Low Power Architectural Synthesis and the Impact of Exploiting Locality," *Journal of VLSI Signal Processing*, vol. 13, no. 2-3, pp. 239–258, 1996.

[10] R. M. Owens, M. Vishwanath, and M. J. Irwin, "A Very Efficient Storage Structure For DWT and IDWT Filters," *To appear in Journal of VLSI Signal Processing.*