IMPLEMENTATION OF RECURSIVE DIGITAL FILTERS INTO VECTOR SIMD DSP ARCHITECTURES

J.P. Robelly, G. Cichon, H. Seidel and G. Fettweis

Vodafone Chair for Mobile Communications Systems
Dresden University of Technology
01062 Dresden Germany
e-mail: robelly@ifn.et.tu-dresden.de

ABSTRACT

Recently, digital signal processors featuring vector SIMD instructions have gained renewed attention, since they offer the potential to speed up the computation of digital signal processing algorithms. However, when implementing recursive algorithms the maximum achievable speed up factors are upper bounded. In this paper we investigate these performance limitations when pure recursive filters are implemented into parallel DSP architectures. We show that by applying algebraic transformations a block formulation of any recursive filter can be derived, which can be efficiently implemented into SIMD DSP architectures. We also show that the number of additional vector operations introduced by the transformation grows linearly with the level of parallelism and that it does not depend on the recursion order. These results enable the achievement of important speed up factors even for low order recursions. Moreover, we introduce a suitable algebraic notation of the block formulation of the recursive filter, which reveals the processor instructions required to implement the algorithm into the SIMD DSP.

1. INTRODUCTION

Parallel SIMD DSP architectures offer the potential to increase the data transfer rates between memory and computational resources, since data vectors residing on memory are accessed in parallel in order to be processed. This enables the achievement of speed up gains in the implementation of DSP algorithms into this processor architectures.

However, there are many algorithms in digital signal processing that are recursive and therefore cannot be easily mapped onto parallel DSP architectures. These algorithms present direct data dependencies, which naturally lead to a serial formulation of the algorithm that is not well suited for the parallel memory access capacity of SIMD DSP architectures. Thus, this type of algorithm becomes a bottleneck that governs the overall performance of the parallel DSP. Among the most prominent examples we can mention IIR filters, adaptive algorithms and decision feedback equalizers. In this paper we are concerned with the analysis of pure recursive filters.

The idea of applying algebraic transformations to speed up the implementation of recursive filters has been thoroughly studied in the past [1]. However, it has not been extensively applied to find suitable implementations of recursive digital filters into SIMD DSPs due to the wide spread believe that the overhead introduced by such transformations hinders the achievement of important speed up factors. This paper presents a mathematical framework for finding efficient realizations of recursive digital filters into parallel DSP architectures based on algebraic transformations.

2. PARALLEL DSP ARCHITECTURES

In this section we introduce a generic parallel DSP concept. In figure 1 a block diagram of a parallel DSP architecture is illustrated. We will assume throughout this paper a scalable number of data paths for this architecture. Furthermore, we assume that the following features are supported:

1. Each memory access requires one clock cycle to be completed.
2. Fetch of data vectors and and data scalars is possible.
3. Modulo address arithmetic is available.
4. Broadcast data transfers from memory to the processor register file are supported and require one clock cycle to be executed.
5. Data vector shift (Zurich-Zip [1] data transfer) is available and is executed in one clock cycle.

This work was supported in part by the German Science Foundation (Deutsche Forschungsgemeinschaft, DFG) under grant SFB358.

[1] Also known as the IBM Zurich-Zip data transfer

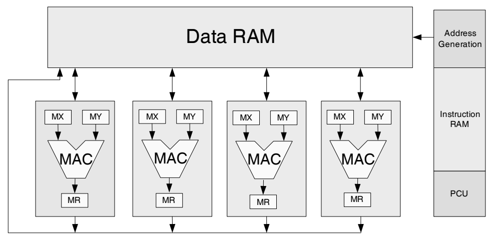

Fig. 1. Block Diagram of a Parallel Processor Architecture

6. MAC operations are executed in one clock cycle.

We also assume that one iteration of the recursive filter is completed once the resulting data has been written into memory. This model will be used later to compare the achievable speed up factors in realizations of recursive filters using a scalable number of parallel data paths with respect to the realization into a serial DSP with only one data path. In order to make a fair comparison, all the assumptions mentioned above hold for the serial DSP case. Of course, vector data manipulation features like data vector fetching, data broadcast and Zurich-Zip, are not supported by the DSP with one MAC unit.

3. BLOCK FORMULATION OF RECURSIVE FILTERS

The serial formulation of a pure recursive filter of order p can be described by the following expression

$$y(k) = u(k) + \sum_{i=1}^{p} a_i y(k-i). \tag{1}$$

A state space representation of this algorithm can be described by the following equations:

$$\begin{aligned} \underline{x}(k+1) &= A\underline{x}(k) + Bu(k) \\ y(k) &= C\underline{x}(k) + Du(k), \end{aligned}$$

where $\underline{x}(k) = [x_1(k) \quad x_2(k) \quad \ldots \quad x_p(k)]^T$ is the state vector at time k. It is important to note that in recursive filters the state vector contains past computed results. Thus,

$$x_i(k) = y(k-i) \qquad \text{for} \quad i = 1, 2, \ldots, p \tag{2}$$

For the system matrices we have

$$A = \begin{bmatrix} a_1 & a_2 & \ldots & a_p \\ 1 & 0 & \ldots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \ldots & 1 & 0 \end{bmatrix}, \tag{3}$$

$$B = [\ 1 \quad 0 \quad \ldots \quad 0\ ]^T, \tag{4}$$

$$C = [a_1 \quad a_2 \quad \ldots \quad a_p], \tag{5}$$

$$D = 1. \tag{6}$$

Our aim is to raise the serial algorithm in order to find a block formulation that deals with input vectors to produce output vector. Thus, we define the input and output vectors of the raised system as

$$\begin{aligned} \underline{u}(k) &= [u(Nk) \quad u(Nk+1) \quad \ldots \quad u(Nk+N-1)]^T \\ \underline{y}(k) &= [y(Nk) \quad y(Nk+1) \quad \ldots \quad y(Nk+N-1)]^T, \end{aligned}$$

where N is the raising factor. The resulting block formulation of the recursive filter can be described by the following raised state-space equations

$$\underline{x}^{[R]}(k+1) = A^{[R]}\underline{x}^{[R]}(k) + B^{[R]}\underline{u}(k) \tag{7}$$

$$\underline{y}(k) = C^{[R]}\underline{x}^{[R]}(k) + D^{[R]}\underline{u}(k). \tag{8}$$

The system matrices of the resulting raised system can be derived from the system matrices of the original serial system as follows [3]

$$A^{[R]} = A^N, \tag{9}$$

$$B^{[R]} = [A^{N-1}B \quad A^{N-2}B \quad \ldots \quad AB \quad B], \tag{10}$$

$$C^{[R]} = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{N-1} \end{bmatrix}, \tag{11}$$

$$D^{[R]} = \begin{bmatrix} D & 0 & \ldots & 0 \\ CB & D & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{N-2}B & CA^{N-3}B & \ldots & D \end{bmatrix}. \tag{12}$$

The state vector of the raised system is defined as follows

$$\underline{x}^{[R]}(k) = \underline{x}(Nk). \tag{13}$$

Thus, according to (2) we can write for the state vector of the raised system

$$\underline{x}^{[R]}(k) = [y(Nk-1) \quad \ldots \quad y(Nk-p)]^T. \tag{14}$$

Thus, taking equations (7)-(14) and defining the system matrices of the original serial system as in equations (3)-(6), we obtain a block formulation for recursive filters that is known as incremental block processing [1]. We obtained this representation using a transformation known as the lifting isomorphism or raising procedure. It is to remark that the matrix $D^{[R]}$ is the overhead matrix introduced by the transformation.

3.1. Numerical Example

Consider a recursive filter of second order $p = 2$ with coefficients $a_1 = \frac{5}{4}$ and $a_2 = -\frac{1}{4}$. Applying the lifting isomorphism with a raising factor $N = 3$, we obtain from equation (8) the following block representation of the second order recursive filter.

$$\underbrace{\begin{bmatrix} y(3k) \\ y(3k+1) \\ y(3k+2) \end{bmatrix}}_{\underline{y}(k)} = \underbrace{\begin{bmatrix} 5/4 & -1/4 \\ 21/16 & -5/16 \\ 85/64 & -21/64 \end{bmatrix}}_{C^{[R]}} \underbrace{\begin{bmatrix} y(3k-1) \\ y(3k-2) \end{bmatrix}}_{\underline{x}^{[R]}(k)}$$

$$+ \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 5/4 & 1 & 0 \\ 21/16 & 5/4 & 1 \end{bmatrix}}_{D^{[R]}} \underbrace{\begin{bmatrix} u(3k) \\ u(3k+1) \\ u(3k+2) \end{bmatrix}}_{\underline{u}(k)}$$

3.2. Tensor Product Representation of Recursive Filters

Tensor product factors have a direct interpretation in vector SIMD processors [4]. Thus, tensor products provide a mathematically correct algebraic syntax that can be directly translated into processor instructions. We adopt in this paper the tensor product notation in order to reveal the necessary instructions to implement the algorithm. For the pure recursive filter, equation (8) can be written in the following way

$$\underline{y}(k) = \underbrace{\sum_{i=1}^{p} \big(y(Nk-i) \otimes I_N\big)\underline{c}_i}_{\text{feedback}} +$$

$$\underbrace{(D \otimes I_N)\,\underline{u}(k) + \sum_{q=1}^{N-1} \big(CA^{q-1}B \otimes I_N\big)\, Z_N^q \underline{u}(k),}_{\text{feedforward}}$$

where $\underline{c}_i$ are vectors formed by the columns of the $C^{[R]}$ matrix and the operator $\otimes$ is defined as the *Kronecker* product. In the tensor product representation of the pure recursive filter we have defined I_N as the $N \times N$ identity matrix and the $N \times N$ shift matrix Z_N as follows.

$$Z_N = \begin{bmatrix} 0 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 1 & 0 \end{bmatrix}$$

3.3. Discussion

From the tensor product formulation of the algorithm it becomes obvious that the computation of the overhead matrix $D^{[R]}$ requires N additional vector operations, whereas the complexity of the feedback part of the algorithm grows linearly with the filter order p. Therefore, the overhead complexity remains constant regardless of the filter order for a fixed raising factor . In fact, if $p \gg N$, the complexity of the overhead is negligible in comparison to the complexity of the feedback part.

The tensor product representation of the algorithm also reveals the necessary processor instructions to implement the algorithm. In equation (15) we can observe the computation of the feedback part of the recursive filter with the interpretation of each algebraic structure in a SIMD instruction. As we can observe the computation consists of vector MAC operations and broadcast data transfers. The broadcast operation is represented as the Kronecker product between a scalar and an identity matrix. This is nothing else than the distribution of the scalar over the different MAC units of the processor. The computation of the feedforward part is shown in (16). We can observe that it requires an additional instruction, namely the computation of a vector shift.

Our intention with the adoption of the tensor product notation is to use a unified language that is sufficiently general to describe algorithms and derive the necessary SIMD instructions from this algebraic description. Although one might find different ways to express mathematically the necessary instructions, we think that tensor products together with the definition of some additional operators like the shift matrix of our example is general enough to deal with algorithms of different complexity.

$$\overbrace{\sum_{i=0}^{p-1} \underbrace{\big(y(Nk-i-1) \otimes I_N\big)}_{\text{Broadcast } y(Nk-i-1)} c_i}^{\text{Vector MAC}} \tag{15}$$

$$\overbrace{\underbrace{(D \otimes I_N)}_{\text{Broadcast } D}\,\underline{u}(k)}^{\text{Component-wise Multiplication}} +$$

$$\overbrace{\sum_{q=1}^{N-1} \underbrace{\big(CA^{q-1}B \otimes I_N\big)}_{\text{Broadcast } CA^{q-1}B} \quad \underbrace{Z^q \underline{u}(k)}_{\text{Vector Shift } q \text{ Positions}}}^{\text{Vector MAC}} \tag{16}$$

4. SCALABILITY ANALYSIS

In this section we analyze the achievable speed up factors when a recursive filter of order p formulated as in section 3 is implemented into a parallel DSP architecture with a parallel number of data paths N as discussed in section 2. The speed up factor is the relation between the number of necessary cycles to compute L filter results in the serial DSP and the number of necessary cycles to compute L filter results in the parallel DSP. In table 1 we can observe the number

	Number of Cycles
Memory Access	$(2p+2)L$
MAC	pL
Total Number of Cycles	$(3p+2)L$

Table 1. Number of cycles for filter of order p and L input samples in a serial DSP

	Number of Cycles
Memory Access	$(2p+N+2)(L/N)$
MAC	$(N+p)(L/N)$
Zurich Zip	$(N-1)(L/N)$
MAC init	(L/N)
Total Number of Cycles	$(3N+3p+2)(L/N)$

Table 2. Number of cycles for filter of order p and L input samples in an N parallel DSP

of necessary cycles to compute L input samples into a serial DSP architecture. In table 2 we observe the total number of cycles required to compute L filter results realized in a parallel DSP with N data paths.

Taking the total number of cycles of table 1 and 2 we have the following speed up factor

$$\text{speedup} = \frac{n(3p+2)}{3n+3p+2}. \qquad (17)$$

In figure 2 we have plotted equation (17) for different values of N for a recursive filter of order p. From this diagram we can observe that the achievable speed up factor is upper bounded. Once this upper bound has been reached, the speed up factor remains constant even if the number of parallel data paths and thus, the available processing power is incremented. It is also important to remark that this upper bound is about the filter order.

Another way to look at the speed up factors is offered in figure 3. Using this figure one can determine the necessary number of data paths in order to achieve some speed up factor for a recursive filter of a certain order. From this figure we can observe that the increment on performance is also limited by the available processing power.

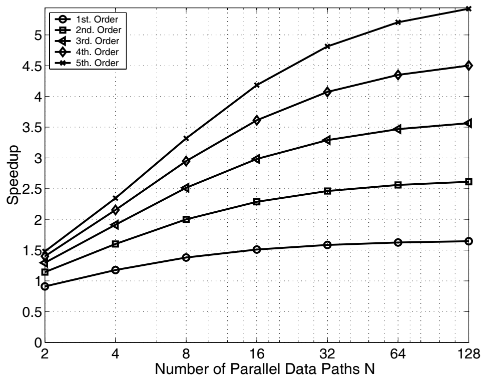

Fig. 2. Speedup vs. Level of Parallelism for recursive digital filters of different orders.

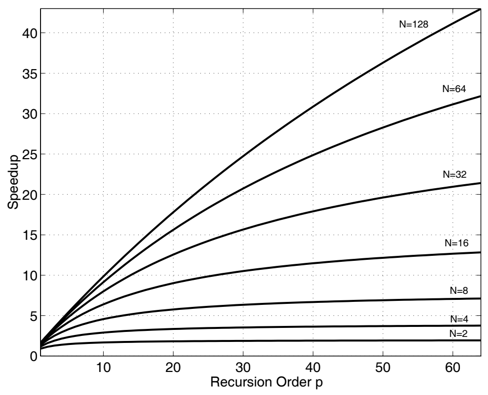

Fig. 3. Speedup vs. Filter Order p for different number of data paths N.

5. CONCLUSION

Parallel DSP architectures can speed up the computation of digital signal processing algorithms. In this paper we showed that by means of algebraic transformations, it is possible to find block formulations of pure serial algorithms. We have shown that the increment on the achievable speed up for the implementation of recursive filters into vector SIMD architectures is upper bounded to a factor that corresponds the order of the filter. We have also introduced a suitable algebraic notation that reveals the required SIMD instructions to implement the algorithm. This algebraic notation can be used to automate code generation.

6. REFERENCES

[1] K. K. Parhi and D. Messerschmitt, “Pipeline interleaving and parallelism in recursive digital filters, part ii: Pipelined incremental block filtering,” *IEEE Trans. Acoust.,Speech Signal Processing*, vol. ASSP-37, no. 7, pp. 1118–1135, July 1989.

[2] D. I. Moldovan, *Parallel Processing: From Applications to Systems*. San Mateo California: Morgan Kaufmann, 1993.

[3] A. Feuer and G. C. Goodwin, *Sampling in Digital Signal Processing and Control*. Boston,Basel,Berlin: Birkhaeuser, 1996.

[4] R. Tolimieri, M. An, and C. Lu, *Algorithms for discrete Fourier transform and convolution*. New York: Springer Verlag, 1997.