On Unlimited Parallelism of DSP Arithmetic Computations

Miodrag Potkonjak
C&C Research Laboratories, NEC USA, Princeton

Jan Rabaey
Dept. of EECS, University of California, Berkeley

ABSTRACT

We present a conceptually simple and practical technique which identifies necessary and sufficient conditions for achieving unlimited improvement in the throughput of a general DSP arithmetic computation (DAC). For the cases when necessary conditions are fulfilled, this technique is a basis for an approach which transforms an arbitrary DAC so that unlimited parallelism is realized. An important special case, linear computations, is addressed using a specially tuned version of the general algorithm so that the introduced latency and hardware overhead are minimized. The effectiveness of the new approach is demonstrated on several real life examples.

1. MOTIVATION AND PRIOR ART

We address the computational concurrency of DSP arithmetic computations (DACs), computations which are performed on a semi-infinite stream of input data using additions, subtractions, multiplications, and divisions. Because of the iteration bound present in recursive DACs, the throughput for this class of computations is often limited by technology and processing constraints. Although several special classes of recursive DACs have been successfully transformed so that arbitrarily high sampling rates can be achieved, a consistent framework for determining when the application of transformations leads to unlimited parallelism has not been addressed.

Rapid progress in semiconductor technology, IC design methodology and CAD tools, removed many of the initial constraints which imposed area as the primary constraint, and elevated throughput as the main criterion for design comparison. Since the mid sixties [Hoc65, Sto73, Kog73], there have been numerous successful attempts at optimizing several classes of recursive computations for asymptotically unlimited throughput. While the first researchers concentrated their effort in the field of numerical solutions of differential equations, and later on for several important classes of DSP computations (e.g. some classes of adaptive linear filters and dynamic programming like computation such as Viterbi decoders) it has been shown that arbitrarily high sampling rates can be achieved by restructuring the computations using special sets of transformations [Gol68, Cha69, Vol70, Bur71, Men87, Mes88, Par89, Fet90, Lin91]. Recently, Lin and Messerschmitt [Lin91] achieved important theoretical results, showing that an arbitrary finite state machine has unlimited concurrency. However, their method is only applicable to systems with a moderate number of binary states, which practically rules out its application on even very small numerical computations. This leaves open the important question: "which DSP arithmetic computations have structures suitable for the exploration of unlimited parallelism?"

The major result of this paper is a technique which identifies when a DAC can be transformed so that arbitrarily high speed-up is possible. For instances of a DAC where necessary conditions are satisfied, a procedure which speeds it up is presented. The procedure is built by combining the concept of temporal loop unrolling (known in the DSP and design literature under a number of different formulations such as look-ahead, K-slow, recursive doubling and block filtering techniques) with common subexpression replication and a well organized set of algebraic transformations. A more detailed description of the work presented here can be found in [Pot93].

2. MAIN IDEA

The main idea behind this new procedure for the unlimited improvement of throughput is conceptually simple, yet effective. The approach is to build on the top of a combination of work done in theoretical computer science in the area of parallel arithmetic computations [Bre74, Mul76, Val83, Mil87, Mil88, May90] and work which addresses the use of computational transformations in the compiler and high level synthesis domain [Pot92]. These results are combined with loop unfolding techniques, techniques widely used in the numerical analysis and VLSI DSP communities. For the general case, we use the results of Valiant et. al [Val83] and Miller, Ramachandran and Kaltofen [Mil88, Kal86]. In some cases, the algorithms presented by Brent [Bre74] and Muller and Preparata [Mul76] are sufficient and introduce lower hardware overhead. Finally, for the important case of linear computations we rely on the recently introduced maximally fast implementation algorithm [Pot92]. We will concentrate in the rest of the paper on the restricted case when the DAC is division free. However, by using either [Kal86] or [Str73], DACs with divisions can be handled in the same manner. The key result on which we are basing the new approach is that the computation of an arbitrarily organized computation of a polynomial of formal degree d (see next section) which

0-7803-0946-4/93 $3.00

uses n operators, can be computed in time O (log n (log n + log d)).

From now on, we will refer to those procedures as algebraic speed-up procedures. There are two major obstacles which limit the performance of the algebraic speed-up procedures when applied to DSP arithmetic computations. First, DSP computations are normally represented as directed acyclic graphs (DAGs) with multiple outputs (often including the states). Second, the algebraic speed-up procedures will only achieve major improvements for large values of n, or, in other words, it is important that the DACs are as large as possible. Both considerations can be elegantly and efficiently addressed using computational transformations.

Common subexpression replication makes it possible to transform an arbitrary DAG into a tree, where each output depends only on a partial subset of the inputs and where fanout to multiple outputs has been eliminated. Common subexpression replication acts as an enabling transformation for the application of the speed-up procedures. To create large graphs (as is beneficial for the algebraic speed-up procedures), the time loop is unrolled a number of times. In this way, arbitrary large graphs and hence speed-ups can be obtained. This is at the expense of extra hardware. It will be proven however that the cost increase is controlled by well defined bounds. The interesting feature of this procedure is that it can be applied on general DACs, and there it is a generalization of a previously published technique [Pot92], which only handles linear computations. For a special case of linear computation, an additional use of grouping and pipelining of all primary inputs and computing them using the Horner scheme results in smaller hardware overhead and better trade-off between latency and throughput [Pot93].

The presented optimization procedure is illustrated for the 2nd order IIR direct form II filter (Figure 1), as shown in Figures 2-4. Figure 2 shows the initial structure of the filter, Figure 3 the 2 times unfolded filter and Figure 4 the structure after the application of common subexpression replication, followed by the algebraic speed-up procedure. The critical path of the final implementation is only 3, although two iterations are performed. Since the initial critical path equaled 4, the throughput is improved by a factor of 2.67.

3. MAJOR RESULT

3.1. When does a DAC have an unlimited parallelism implementation?

An algebraic straight-line program consists of a sequence of assignment statements executed in a specified order. The expressions on the right-hand side are expressions which involve arithmetic operators (+ , - ,* , /) that are applied either on an input variable or variable to which value is assigned in an earlier assignment. Straight-line programs are often represented as DAGs with a single output node. Since both representations, when only (+, - , *) are used (when division is also used see [Str73]), compute some polynomial, the formal degree of a computation represented as a straight-line program or a DAG is defined as the degree of the computed polynomial [May90]. Similarly, for each node of the corresponding DAG, the formal degree can be defined. Note that when a node is + or -, its degree is equal to the maximum degree of its inputs, when a node is *, the degree is the product of the degrees of its input degrees. Theorem 1 states a simple and computationally efficient test to detect whether a given DAC can be implemented at arbitrarily high speed [Pot93].

Theorem 1: The throughput of a recursive DAC can be unlimitedly improved if and only if the formal degree of the N-times unrolled initial iteration is growing at a sublinear rate compared to N.

The required testing of the rate of growth of the formal degree can be easily achieved by analyzing feedback dependencies of all multiplications [Pot93].

3.2 Algorithms for achieving unlimited parallelism

The Algorithm for the arbitrary speed-up of an arithmetic DSP computation can be described using the following pseudo-code:

1. *Unfold the basic iteration N times;*
2. *Organize the arithmetic expression for each output as a tree using common subexpression replication;*
3. *Using the algebraic-speed-up procedure, reduce the critical path of the unfolded expressions for all outputs;*
4. *Pipeline all computation outside recursive loops;*
5. *Minimize the resulting graph using common subexpression elimination.*

The power of the algorithm is expressed by the following theorems. The detailed proofs of all theorems can be found in a technical report [Pot93].

Theorem 2: The throughput of a recursive DAC which satisfies the conditions from Theorem 1 can be increased arbitrarily.

Theorem 3: The ratio of the transformed to the initial AT^2 product grows only as the square of logarithm.

Theorem 4: The proposed algorithm produces the design which has asymptotically the minimum critical path among all designs with the same latency.

3.3 Linear Computation

For the important and widely used special case of linear

Before Unfolding:

$$x_1 = In_1 + a^* x_0 + b^* y_0$$
$$y_1 = x_0 \qquad \text{(a)}$$
$$Out_1 = x_1 + c^* x_0 + d^* y_0$$

After Unfolding:

$$x_1 = In_1 + a^* x_0 + b^* y_0 \qquad x_2 = In_2 + a^* x_1 + b^* y_1$$
$$y_1 = x_0 \qquad y_2 = x_1 \qquad \text{(b)}$$
$$Out_1 = x_1 + c^* x_0 + d^* y_0 \qquad Out_2 = x_2 + c^* x_1 + d^* y_1$$

After Transformations:

$$x_2 = In_2 + a^* In_1 + (a^* a + b)\, x_0 + a^* b^* y_0$$
$$y_2 = In_1 + a^* x_0 + b^* y_0 \qquad \text{(c)}$$
$$Out_2 = In_2 + (a + c)\, In_1 + (a^* a + b + c^* a + d)\, x_0 + (a^* b + c^* b)\, y_0$$

FIGURE 1. Functional dependences for 2nd order IIR filter before and after transformations

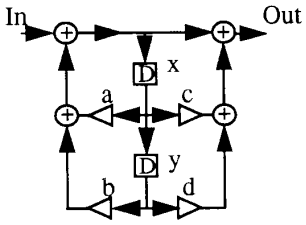


FIGURE 2. Second order IIR filter

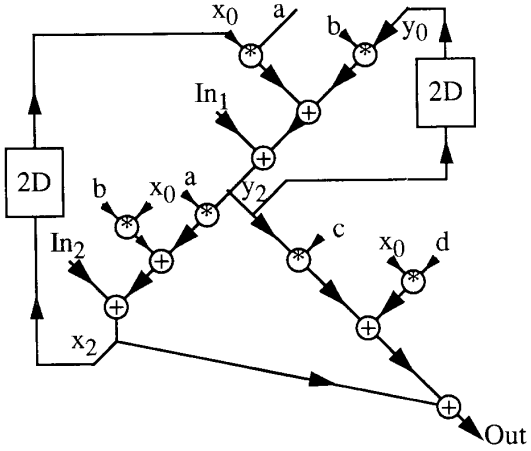
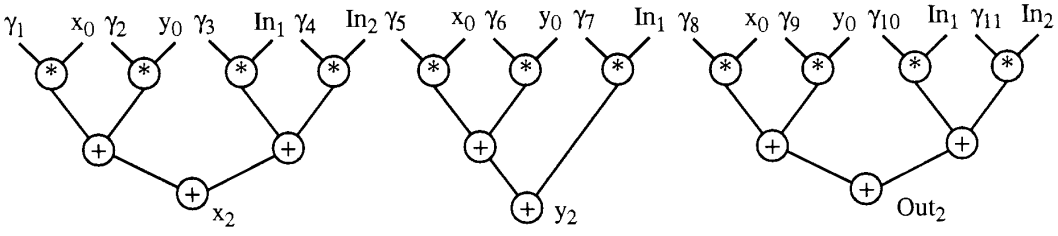


FIGURE 3. Flow graph of the unfolded IIR filter



γ_1=a*a + b; γ_3=a; γ_5=a; γ_7=1; γ_9=a*b + c*b; γ_{11}=1;
γ_2=a*b; γ_4=1; γ_6=b; γ_8=a*a + b + c*a + d; γ_{10}=a + c;

FIGURE 4. Flow graph for restructured IIR filter.

computation, by using the maximally fast implementation of linear programs in the third step of the new algorithm, a significant improvement over the previously published result can be achieved, as illustrated with the following two theorems [Pot93]:

Theorem 5: The throughput of a recursive linear DAC can be increased arbitrarily so that the latency increases at the same rate as the throughput.

Theorem 6: The ratio of the initial and the transformed AT for a linear DAC is constant.

3.4. Experimental Results

The effectiveness of the procedure is illustrated in Table 1 for two linear examples (51st order FIR filter and 8th order Avenhaus IIR filter) and two non-linear examples (a Volterra filter and an Adaptive FIR filter).

Unfolding Factor / Example	Initial	2	5	10	50	100
51 FIR	1	14.6	36.5	73	365	730
8 Avenhaus	1	3.6	9.0	18	90	180
Volterra	1	1.25	2.9	4.4	18	33
Adaptive FIR	1	3.9	8.1	15	61	110

Table 1: The throughput improvement for four examples

4. CONCLUSION

Necessary and sufficient conditions are derived for identifying when an arbitrary DAC can be transformed using unfolding so that the effective throughput is arbitrarily high. An efficient algorithm has been presented, which transforms an arbitrary DSP arithmetic computation which satisfies the mentioned criteria, so that the throughput of the transformed implementation is arbitrarily high. From a theoretical point of view, the importance of the new approach is in the re-establishment of loop unfolding as a powerful and general optimization technique. It also opens new directions for the exploration of loop unfolding by combining it with other transformations. From the practical side, since the hardware overhead is bounded and of an acceptable nature, the approach enables the speed-up of many real-life DSP numerical computations. For the important special case of linear computation, the new procedure results in solutions where the AT product is constant, regardless of speed-up and latency is traded at a linear rate for throughput.

5. REFERENCES

[Bre74] R.P. Brent: "The parallel evaluation of general arithmetic expression", Journal of ACM, Vol. 21, No. 2, pp. 201-206, 1974.

[Bur71] C.S. Burrus: "Block Implementation of Digital Filters", IEEE Trans. on Circuits Theory, Vol. 18., No. 6, pp. 697-701.

[Cha69] D. Chanoux: "A method of Digital Filter Synthesis", M.S. thesis, MIT, Cambridge, MA, May 1969.

[Fet90] G. Fetwies, H. Meyr: "A 100 Mbit Viterbi Decoder Chip: Novel Architecture and its Realization", IEEE Internation Conference on Communications, Vol. 2, pp. 463-467, 1990.

[Gol68] B. Gold, K.L. Jordan: "A Note on Digital Filter Synthesis", Proc. of IEEE, pp. 1717-1718, 1968.

[Hoc65] R. Hockney: "A fast direct solution of Poisson's equation using Fourier analysis", Journal of ACM, Vol. 12, No. 1, pp. 95-113, 1965.

[Kog73] P.Koge, H. Stone: "A parallel algorithm for the efficient solution of a general class of recurrence equations", IEEE Trans. of Comp., Aug 1973.

[Lin91] H. Lin, D.Messerschmitt: "Finite State Machine has Unlimited Concurrency", IEEE Trans. on CAS, Vol. 38, No. 5, pp. 465-475, 1991.

[May90] E.W. Mayr: "Theoretical Aspects of Parallel Computation", in "VLSI and Parallel Computation", ed. by R. Suaya, G. Birtwistle, Morgan Caufman, Palo Alto, CA 1990.

[Men87] T. Meng, D.G. Messerschmitt: "Arbitrarily high sampling rate adaptive filters", IEEE Trans. ASSP, apr. 1987.

[Mes88] D.G. Messerschmitt: "Breaking the recursive bottleneck", in "Performance Limits in Communication Theory and Practice", J.K. Skwirzinsky, Ed., Kluwer, Amsterdam, 1988.

[Mil87] G.Miller, S.-H. Teng: "Dynamic parallel complexity of computational circuit", 19th ACM Symposium on Theory of Computing, pp. 254-263, 1987.

[Mil88] G. Miller, V. Ramachandran, E. Kaltofen: "Efficient parallel evaluation of straight-line code and arithmetic circuits", SIAM Journal of Computing, Vol. 17, No. 4, pp. 687-695, 1988.

[Mul76] D.E. Muller, F.P. Preparata: "Restructuring of Arithmetic Expressions For Parallel Evaluation", Journal of ACM, pp. 534-543, 1976.

[Par89] K. K. Parhi, D. Messerschmitt: "Pipeline interleaving and parallelism in recursive Filters, Parts 1 and 2", IEEE Trans. on ASSP, Vol. 37, pp. 1099-1117, 1117-1134, 1989.

[Pot92] M. Potkonjak, J. Rabaey: "Maximally Fast and Arbitrarily Fast Implementation of Linear Computations", ICCAD-92, pp. 304-308, 1992.

[Pot93] M. Potkonjak, J. Rabaey: "On Unlimited Parallelism in DSP Arithmetic Computations", Technical Report 5510-093-001, NEC USA, Princeton, 1993.

[Sto73] H. Stone: "An efficient algorithm for the solution of a tridiagonal linear system of equations", Journal of ACM, Vol. 20, No. 1, pp. 27-38, 1973.

[Str73] V. Strassen: "Vermeidung von divisionen", Journal of Reine U. Angew. Math., Vol. 264, pp. 182-202, 1973.

[Val83] L. Valiant, S. Skyum, S. Berkowitz, C. Rackof: "Fast parallel computation of polynomials using few processors", SIAM Journal of Computing, Vol. 12, No. 4, pp. 641-644, 1983.

[Vol70] H.B. Voelcker, E.E. Hartquist: "Digital Filtering via Block Recursion", IEEE Trans. on Audio Electroacous., Vol. 18. No. 2, pp. 169-176.