FAST IMPLEMENTATION OF RECURSIVE PROGRAMS USING TRANSFORMATIONS

Miodrag Potkonjak

C&C Research Laboratories, NEC, 4 Independence Way, Princeton, NJ 08540

Jan Rabaey

Dept. of EECS, University of California at Berkeley, Berkeley, CA 94720

ABSTRACT

An automatic transformational approach to reduce the iteration bound of recursive DSP algorithms is presented. The proposed approach combines delay retiming, algebraic transformations and loop unrolling in a well defined order. The effectiveness of the approach is demonstrated using examples.

1. INTRODUCTION

While in a number of application specific designs an efficient hardware resource utilization is of primary interest, another common situation in signal processing is that throughput requirements are at the edge of what can be achieved using available technology. In these cases, reaching the throughput requirements is the single most important goal, even if it results in lower resource utilization rates. A general intrinsic property of many signal processing applications is that the increased latency can to some extent be tolerated. This is in contrast with some other areas, such as robotics, where the time between the acceptance of the input and the issuing of the output is most important. Therefore, during throughput optimizations, constraints on the latency should be taken into account.

2. PROBLEM FORMULATION

When throughput rate is of primary importance and the CDFG does not have feedback edges, pipelining provides a straightforward solution. It is sufficient to introduce as many pipeline stages as are allowed by latency (e.g., add as many delays on all input, or all output edges, but not on both) and then to retime the resulting graph using the Leiserson-Saxe retiming algorithm [1]. It is important to notice that when CDFG has feedforward edges, an introduction of a dummy transfer operation on those edges is necessary [2]. Of course, the Leiserson-Saxe algorithm will do this automatically.

However, most of the signal processing algorithms have internal recursion. Examples of such algorithms include both relatively simple cases, such as infinite response and adaptive filters, and more complex ones, such as algorithms for solving systems of non-linear equations and adaptive compression algorithms. Graphs involving recursions display an upper bound on the computation rate, called the pipeline stage bound (or iteration bound) [3]. This pipeline stage bound is given by $T_{iter}=\max(T_i/(ND_i))$. The maximum is taken over all loops, T_i is the sum of the computation times of all the nodes in loop i, and ND_i is the number of delay elements in loop i [3].

Several researchers addressed some special program instances (e.g., IIR filters, Viterbi processor, quantizer loops) and achieved a significant progress in reducing the pipeline stage bound [3,4,5,6,7]. Our goal is to find an approach that will automatically transform arbitrary programs (including, of course, recursive programs), into a form where the pipeline stage bound is reduced to a minimum for a given latency. This can be achieved in a dual way: reducing T_{iter} by applying algebraic transformations (associativity, commutativity and distributivity) and by moving delays (retiming). In order to provide more possibilities for both approaches, it is often necessary to do partial unrolling of the time loop.

3. SMALL EXAMPLE

Very often the best way to introduce a new idea is to explain its steps using a simple example. This section has the goal of illustrating and explaining how basic transformations can be used to reduce significantly the critical path of the pipeline stage. Neither the example nor the idea of reducing the pipeline stage is new. What is new is that minimization of the critical path in the pipeline stage is achieved by an explicit and systematic application of basic transformations. This provides a framework for the solution of the problem of the fast implementation of arbitrary recursive algorithms.

Consider the example shown in Figure 1a, which represents first order IIR filter. This is actually the smallest possible example on which it is still possible to improve pipeline stage time, without going into suboperation level transformations. This filter contains only one loop with two operations: one addition and one multiplication. Both operations are in the loop. Figure 2a shows the corresponding pseudo-code. Assume that each operation takes one cycle. Since the loop contains two operations and one delay, the pipeline stage is 2 cycles long.

Figure 2b shows the pseudo-code after the loop is unrolled once. Figure 1b shows the flow graph, after the application of this basic transformation. The main effect is that the loop now contains two delays. However, the number of operations in the loop also increased twice, so the pipeline stage, which can be achieved using retiming, is still two. Although we did not immediately profit from the application of the loop unrolling, this transformation is creating a starting position for the other basic transformations.

We can see that the associativity cannot be applied to the resulting flow graph. However, we can apply distributivity. The resulting pseudo-code is shown in Figure 2c, while the new flow graph is in Figure 1c. Again, the critical path is not reduced; actually, the required amount of the hardware has increased. However, associativity can now be applied to both additions and multiplications to remove one addition and one multiplication from the loop. The effect is shown in the pseudo-code format in Figure 2d, and in the flow graph format in Figure 1d. As the consequence of the application of associativity, we now have only two operations in the loop. Since the loop also has two delays, the pipeline stage is reduced to one. This can be achieved by using pipelining as shown in Figure 1e. In this way we achieve the maximum throughput increase without going to suboperational transformations.

4. FAST IMPLEMENTATION OF RECURSIVE PROGRAMS USING TRANSFORMATIONS: PROCEDURE

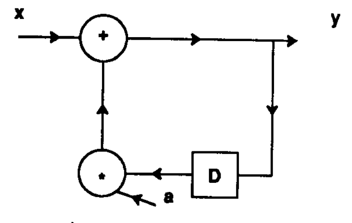
We proved by using polynomial reduction from equal subset-sum problem, that the combination of both retiming and algebraic laws to transform programs into a maximally fast recursive implementation is NP-complete even for a restricted formulation of the problem, where the program contains only adders and delays[8].

A simple, yet efficient procedure for transforming an arbitrary computation graph so that it can be implemented with the very short pipeline stage, can be given using the following six steps, described by the following pseudo-code:

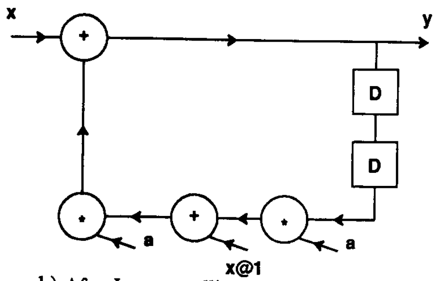
```
1 Using distributivity and associativity, move as many
  operations as possible out of the cycles;
2 Unroll the time loop. The number of unrollings is
  bounded either by hardware or timing constraints;
3 Repeat step 1;
4 Retime the graph so that the structure of the resulting
  graph is such that algebraic transformations will have
  a maximal effect;
5 Reduce the critical path of the resulting graph using
  either the Valiant [9] or Miller [10] algorithm (which
  optimally applies associativity and distributivity);
6 Introduce sufficient pipeline stages (using a revised
  Leiserson's retiming algorithm);
```

Steps 1, 3, 5 are reducing the iteration bound using algebraic transformations. Step 4 is a crucial step, which provides valuable pre-processing for the final iteration bound reduction in step 5. Step 6 is the final auxiliary step.

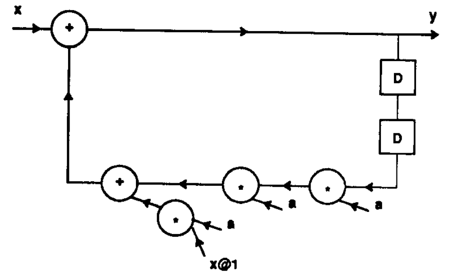
One of the main problems to be addressed when implementing the retiming step is the selection of the objective function. A simple and easy to compute function would be the number of nodes



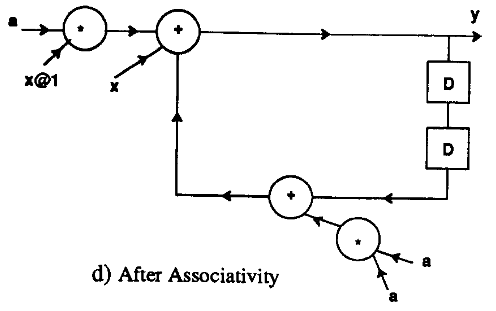
a) Original



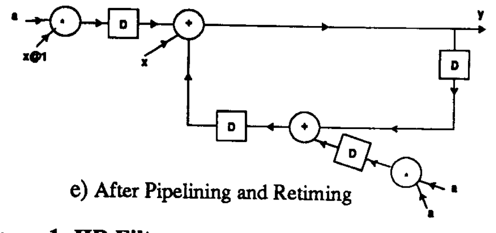
b) After Loop unrolling



c) After Distributivity



d) After Associativity



e) After Pipelining and Retiming

Figure 1: IIR Filter

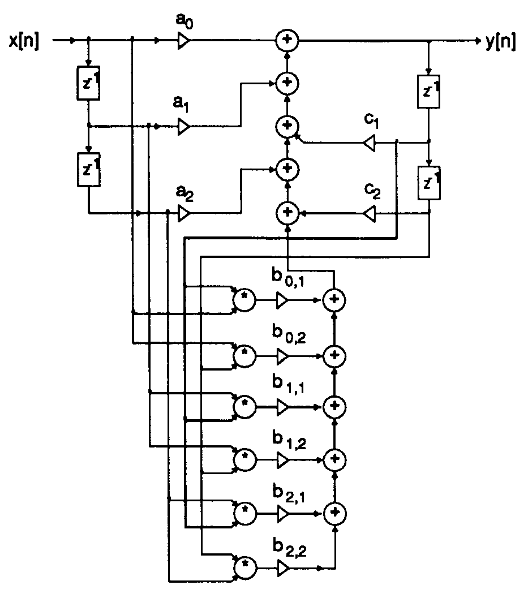
a) y = x + a*y@1

b) y = x + a * (x@1 + a*y@2)

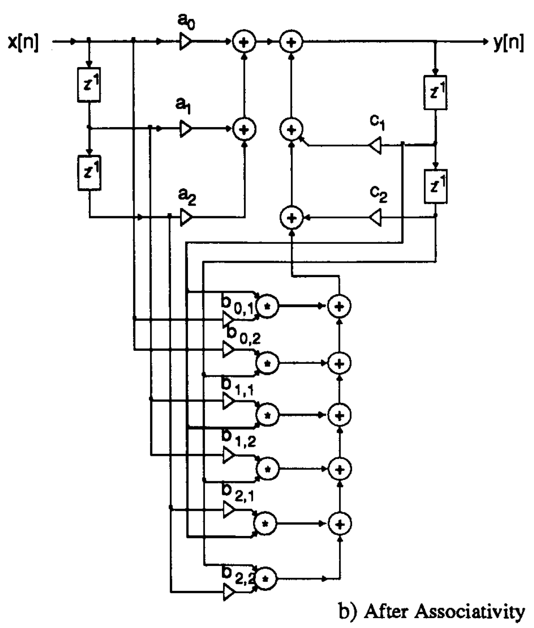
c) y = x + a*x@1 + a*(a*y@2)

d) y = x + a*x@1 + (a*a)*y@2

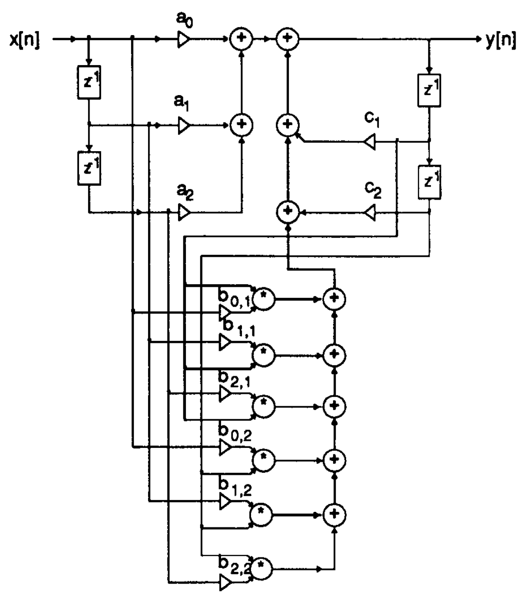
Figure 2: Effect of transformations, pseudo code format



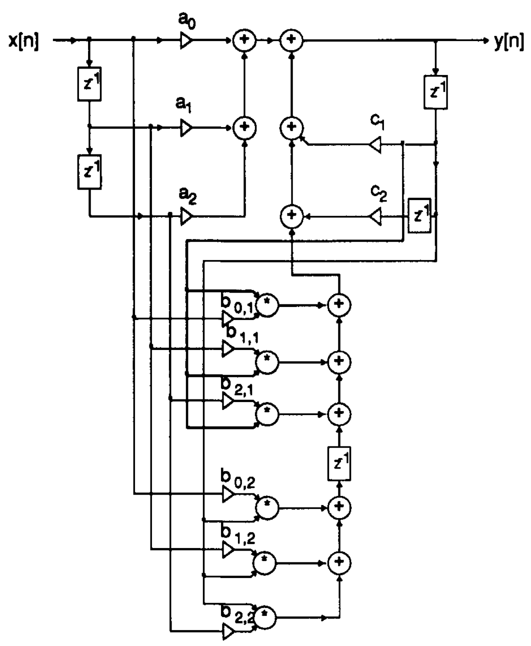
a) Original



b) After Associativity



c) After Additional Associativity



d) After Pipelining and Retiming

e) After transforming addition chain in Figure 3d to tree structure the critical path is 4

Figure 3: Volterra filter, bold line denotes critical path

contained in the largest pipeline stage. It is often possible to reduce the critical path of a pipeline stage close to the optimal $\log_2 n$, where n is the number of nodes in that stage. A more accurate (but computationally more expensive) function can be used for the important class of applications which use only multiplications, shifts, additions and subtractions. Linear as well as adaptive filters are part of this class. In this case, the smallest possible critical path of the graph equals (log C) (log C + log d), where C is the number of operations in the pipeline stage and d the degree of polynomial represented by pipeline stage. The same formula can also be used when the computation only contains addition as well as min and max operations. Examples of such applications can be found in the areas of neural networks, Markov modeling, dynamic programming and fuzzy logic. The same also holds for computations containing only logic "and" and "or" operations (logic synthesis). This isomorphism was first observed by Miller [10]. The retiming transformation itself can be implemented using a statistical approach, identical to the technique described in [11].

5. EXAMPLE: VOLTERRA SECOND ORDER POLYNOMIAL FILTER

The effect of the procedure is illustrated using a second order Volterra filter (Figure 3a). It is a polynomial non-linear filter [12], and previously has not been discussed in the context of fast implementation of recursive programs. We assume that each operation takes one cycle.

The critical path of the initial CDFG is 12, and it is denoted by the bold lines. The application of pipelining cannot reduce the critical path. However, the application of the just described procedure results in a reduction of the critical path to only 4 control cycles, even when unrolling is not applied. An application of unrolling will result in an additional reduction, but for the sake of clarity we will only discuss the case where unrolling is not used.

Application of step #1 reduces the critical path to 9, as shown in Figure 3b. Note that both multiplications by constants and all additions which can be pipelined are removed from the loop which is denoted by the bold line. Since we are not using unrolling, we will skip steps 2 and 3.

After the application of step 4, we obtain the graph shown in Figure 3c. Using associativity we have "deconvoluted loops" and using retiming we have separated the two feedback loops. Now the critical path is 7. Finally, by using the associativity applied on the adder tree (the chain of additions connected by the bold lines) in Figure 3d, we can get the CDFG with the critical path 4. Since all CDFG parts connected to the input can be easily pipelined, and other loops have a shorter critical path, we achieved a threefold reduction of the critical path.

6. PROCEDURE PROPERTIES

Several observations can be made about the proposed procedure for transforming arbitrary CDFG for fast implementation.

During retiming, it is necessary to use both associativity and distributivity in order to get the best results. The rationale is very similar to the one discussed during retiming for resource utilization, i.e., sometimes it is necessary to change the structure of a control data flow graph in order to apply retiming more optimally. Therefore, for this step we can use the same algorithm as for the retiming for resource utilization. The only modification is a different objective function.

During step 4, the key profit often comes from the so-called "loop deconvolution" (as in the Volterra filter example). This refers to the minimization of the number of edges which belong to more than one loop using associativity. This can be by enforced using an appropriate objective function when associativity is applied.

A good objective function is the number of nodes in a pipeline stage. It is often possible to reduce the critical path to near optimal $\log_2 n$, where n is a number of nodes in a pipeline stage. A better objective function can be used in several important special cases: (i) when the computation graph has only additions, multiplication and subtractions; (ii) when it has only min, max and addition operations; and (iii) when it has only "*and*" and "*or*" operations. The first case is important in signal processing, e.g., for many filter structures [13], the second one in a large number of artificial intelligence and game theory applications as well as in fuzzy logic [14]. The third one has a significant application in logic synthesis [15].

7. CONCLUSION

An automatic transformational approach to reduce the iteration bound of recursive DSP algorithms is presented. The proposed approach combines delay retiming, algebraic transformations and loop unrolling in a well defined order.

8. REFERENCES

[1] C.E. Leiserson, F.M. Rose, J.B. Saxe: "Optimizing synchronous circuits by retiming", Proceedings of Third Conference on VLSI, pp. 23-36, Computer Science Press, 1983.

[2] A. Nicolau, R. Potasman: "Incremental Tree Height Reduction for High Level Synthesis", *28th ACM/IEEE DAC*, pp. 770-774, 1991.

[3] D. Messerschmitt, "Breaking The Recursive Bottleneck", in Performance Limits in Communication Theory and Practice, Kluwer Academic Publishers, 1988.

[4] K.K. Parhi, D.G. Messerschmitt: "Pipeline interleaving and parallelism in recursive digital filters - Part I & Part II" *IEEE T-ASSP*, pp. 1099-1117 & pp. 1118-1134, July 1989.

[5] H.-P. Lin, D.G. Messerschmitt: "Finite State Machine has Unlimited Concurrency", *IEEE Trans. on Circuits and Systems*, Vol. 38, No. 5, pp. 465-475, 1991.

[6] A. Fetweis, H. Meyr, L. Thiele: "Algorithm Transformations for Unlimited Parallelism", *IEEE International Symposium on Circuits and Systems*, pp. 1756-1759, New Orleans, 1990.

[7] K. Parhi, "Algorithm and architecture design for high speed digital signal processing", *Ph.D. Dissertation*, University of California, Berkeley, 1988.

[8] M. Potkonjak: "Algorithms for High Level Synthesis: Resource Utilization Based Approach", *Ph.D. Dissertation*, University of California, Berkeley, 1991.

[9] L.G. Valiant, S.Skyum, S. Berkowitz, C. Rackoff: "Fast Parallel Computation of Polynomials Using Few Processes,", *SIAM Journal on Computing*, Vol. 12, No 4, pp. 641-644, 1983.

[10] G.L. Miller, S. Teng: "Dynamic parallel complexity of computational circuits", *Proc. 19th Ann. ACM Symp. on Theory of Computing*, pp. 254-263, 1987.

[11] M. Potkonjak and J. Rabaey, "Optimizing the Resource Utilization Using Transformations", *Proc. IEEE ICCAD Conference*, Santa Clara, Nov. 1991.

[12] V.J. Mathews: "Adaptive Polynomial Filters", *IEEE Signal Processing Magazine*, Vol. 8. No. 3, pp. 10-26, July 1991.

[13] R. E. Blahut, "Fast Algorithms for Digital Signal Processing", Addison-Wesley Publishing Company, 1985.

[14] B. Kosko: "Neural networks and fuzzy systems: a dynamical systems approach to machine intelligence", Englewood Cliffs, NJ: Prentice Hall, 1991.

[15] R.K. Brayton et al.: "Logic minimization algorithms for VLSI synthesis", Boston: Kluwer Academic Publishers, 1984.