Throughput Optimization of General Non-Linear Computations

Inki Hong, Miodrag Potkonjak† and Lisa M. Guerra‡
Synopsys Inc., Mountain View, CA
†Computer Science Department, University of California, Los Angeles, CA
‡Rockwell Semiconductor, Newport Beach, CA

Abstract

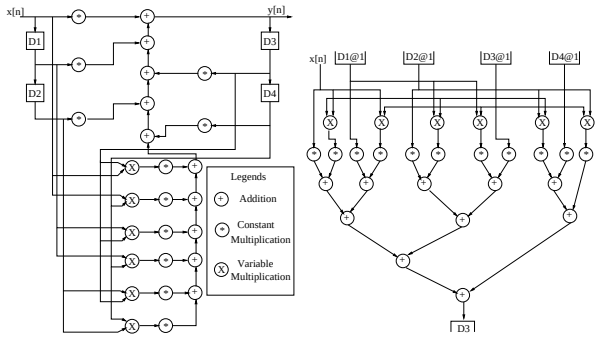
This paper addresses an optimal technique for throughput optimization of general non-linear data flow computations using a set of transformations. Throughput is widely recognized as the most important design metric of the modern DSP and communication applications. Numerous approaches have been proposed for throughput optimization, but most was restricted to limited classes of computations. They have limited effectiveness when applied to large complex non-linear DSP and communication computations. The new technique is used as an optimization engine in a divide-and-conquer global approach for throughput optimization. We demonstrate the effectiveness of the new technique on numerous real-life non-linear designs.

1 Introduction

Throughput is widely recognized as the most important design metric of modern DSP and communication applications [7]. Techniques for optimizing throughput are also widely used during the optimization of other design metrics such as area and power. For example, one of the most effective techniques to optimize power is voltage scaling which is enabled by throughput optimization [1]. Iteration bound, and control and data dependencies impose fundamental limits on achievable performance. Transformations are one of the most effective ways to overcome these limitations [10, 1]. Transformations alter the structure of a computation in such a way that the user specified input/output relationship is maintained. Numerous approaches have been proposed for throughput optimization using transformations, but most was restricted to limited classes of computations such as linear computations and even more restricted instances of linear computations [8, 9]. They are inapplicable or ineffective when applied to large complex non-linear DSP and communication computations. Guerra et al. [2] recently proposed a divide-and-conquer approach for general non-linear computations which leverages upon existing techniques for limited computation types by enabling more effective and coordinated use of the techniques. The approach logically divides the whole computation into subparts which may fall into special computation types that can be optimized effectively by existing techniques. However, the approach is ineffective when subparts are classified as general non-linear computation for which there are few existing effective techniques for throughput optimization.

To overcome these limits, this paper addresses an optimal technique for throughput optimization of general non-linear computations using a set of transformations (distributivity, associativity, inverse and zero element law, common subexpression replication and elimination, constant propagation, pipelining and unfolding). The target applications are large complex non-linear DSP and communication data-flow intensive computations.

To illustrate the key ideas behind the new approach and show its effectiveness, consider the control data flow graph (CDFG) of 2nd order Volterra filter, shown in Figure 1(a). The 2nd order Volterra filter is non-linear. The critical path of the computation is 12 clock cycles, assuming each operator takes one clock cycle. The critical path occurs when the next state of the delay D_3 is computed. A number of known techniques can be used to improve the throughput, but only by limited amounts. For example, pipelining may reduce the critical path to at best 9 clock cycles. Using the technique proposed in this paper, the critical path can be reduced to 6, as shown in Figure 1(b).



(a) Original CDFG (b) Optimized D_3 cone

Figure 1: Motivational example: 2nd order Volterra filter

2 Related Work

Numerous techniques for throughput optimization have been proposed. The first set of techniques is based on transformations [6, 8, 9]. For control-flow intensive computations, Lakshminarayana and Jha [4] proposed a set of techniques for throughput and power optimization. Most was restricted to limited classes of computations such as linear computations and even more restricted instances of linear computations. Guerra et al. [2] proposed a divide-and-conquer approach for general non-linear computations which logically divides the computation into subparts and applies sets of individual techniques for limited computation types to subparts. This technique has limited effectiveness when subparts are classified as general non-linear computation for which there are few existing effective optimization techniques. In this research, we develop an optimal technique for throughput optimization of general non-linear computations using a set of transformations. In general, arbitrary speed-up is not known to be achievable for some classes of non-linear computation [3].

3 Preliminaries

We assume that an arbitrary non-linear computation is given. We do not impose any restrictions on considered non-linear computation. We use as computational model homogeneous synchronous

0-7803-5832-5/99/ $10.00 © 1999 IEEE

data flow model [5], which is widely used in many application domains such as DSP, video and image processing, communications, control, and information theory applications. Under this model, the operations consume a single sample from each input and produce a single sample on each output, on every execution. Operation delays are given as an integral number of clock cycles. An important ramification of the semantics of the selected computation model is that it lends itself to efficient static scheduling. Furthermore, under this model, throughput is the relevant metric of performance, where throughput is the maximum rate at which a design can accept and process incoming samples. The inverse of the throughput is the critical path length, which is the maximum length of all paths which start at any of the primary inputs or states and finish at any of the primary outputs or states.

Non-recursive computations, either linear or non-linear, can be maximally sped up by pipelining. The critical path length of the computations can be reduced to the maximum delay of the longest operator in the computation. Let D_{max} denote the maximum delay of the longest operator in the computation. Combined with time-loop unfolding, this type of computations can be arbitrarily sped up since the effective critical path length is reduced to $D_{max}/(k+1)$ when the unfolding factor is k. Based on these results, the parts of the general non-linear computation which do not belong to feedback cycles can be optimized separately from those which belong to feedback cycles. Thus, we concentrate on the parts which belong to feedback cycles in our study to develop a technique for throughput optimization of general non-linear computation.

In linear computations, the next state (delay) and the outputs are linear functions of the previous states and inputs. A system exhibits linearity when using as computational elements only addition, subtraction and multiplication with constants. A system can also exhibit linearity over either *min* or *max* in place of the addition, and addition in place of the multiplication with constants. Given arbitrary linear computations, the maximally fast technique [9], which combines several algebraic and redundancy manipulation transformations, reduces the critical path length to $C = \lceil log_2(N)+1 \rceil + 1$, where N is the number of states. Combined with time-loop unfolding, the technique arbitrarily speeds up the linear computations such that the effective critical path length is reduced to $C/(k+1)$, where k is the unfolding factor.

The class of feedback linear computations is a special class of general non-linear computations such that there are no multiplications or divisions between variables in feedback cycles. These computations may have multiplications and divisions between primary inputs, between primary inputs and algorithmic delays in feedback cycles, or between algorithmic delays, not in feedback cycles. The maximally fast approach for linear computation can be applied to this class of computations to result in arbitrary speed-up of throughput [2, 9].

A limited subset of general non-linear computations can be translated into a form in which all non-linear operations are moved outside of feedback cycles. A computation containing only variable multiplications is such a computation [2].

4 Global Approach

Our strategy for optimizing throughput of general non-linear computations is based on the divide-and-conquer paradigm [2]. The divide-and-conquer approach is based on an observation that while a given computation may overall be difficult to optimize, some of its subparts can be optimized more easily. The divide-and-conquer approach divides a computation into subparts, which are separately optimized and thus enables more effective use of existing techniques for special classes of computations. This approach has limited effectiveness when subparts do not fall into one of the special classes of computations for which effective techniques exist. We develop an optimal technique to optimize throughput of general non-linear computation using a set of transformations. This new technique is used as an optimization engine when subparts are classified as general non-linear computation.

The divide step partitions the computation by identifying mutually exclusive subparts. This involves identifying operations inside feedback cycles, and those outside of feedback cycles. Since all operations outside of feedback cycles can be pipelined to any requested level, we only need to consider the subparts of the computation inside feedback cycles for throughput optimization. The identification of operations inside feedback cycles is done by identifying the computation's strongly connected components(SCCs), using the standard depth-first search-based algorithm [11]. For any pair of operations A and B within a SCC, there exist both a path from A to B, and one from B to A. All operations in non-trivial SCCs (those with more than one operation) are part of a feedback cycle. The non-trivial SCCs are isolated from each other and from parts outside of feedback cycles using pipeline delays, which allows independent optimization. During subpart optimization, the inserted pipeline delays are treated like a subpart primary input or primary output, depending on whether the subpart reads from or writes to the delay. The non-trivial SCCs are classified as being either linear or non-linear. Non-linear computations are further classified as being either feedback linear, being transformable to a form in which all non-linearities are moved outside of the SCC, or being neither.

Depending on the class in which an SCC belongs to, different optimal techniques are used to optimize throughput. The global throughput of the whole computation is the minimum of the throughputs for all SCCs. For each class of computations except the last class, the general non-linear computation, there exist optimal techniques for throughput optimization. We develop an optimal technique for throughput optimization of general non-linear computations, which is described in Section 5.

5 Throughput Optimization of General Non-Linear Computations

In this Section, we describe a new technique to optimize throughput of general non-linear computations.

5.1 Enabling Steps

Unfolding the computation by a factor of k results in the simultaneous computation of $k+1$ iterations of the computation. For general non-linear computations, the throughput will continue to improve with larger factors of unfolding up to some point and then level off with larger factors. The search to find an optimal unfolding factor starts with an unfolding factor of 1. As long as the throughput continues to improve, the factor is doubled and the throughput optimization is performed. Once no improvement is achieved, a binary search between the current factor k and previously tried factor, $k/2$, is performed to find the smallest optimal factor for which improvement will cease.

For the computation, unfolded by any factor, we apply the same technique to optimize throughput. First, we group and pipeline all primary inputs so that each next state depends on only one new pipeline delay. This step is an enabling transformation which greatly reduces the influence of the primary inputs. This step is most effective when a computation has a large number of primary inputs and relatively few states in feedback cycles. Next, we identify dependencies between delays (both algorithmic and pipeline delays) and non-linear operations. The dependencies can be represented in a directed acyclic graph. Finally, using the dependency information obtained in the last step, we arrange all output cones for next states using the fastest tree structure as described in the next subsections. The global throughput is the minimum of the throughputs for all output cones.

5.2 General Output Cone

Given a general output cone, the dependencies between delays and non-linear operations can be represented in a tree. If there are

only linear operations between delays, between nonlinear operations, and between delays and non-linear operations, we connect a directed edge between them. At the top level, an output may be dependent on some independent non-linear operations and some delays. Each of these independent non-linear operations may depend on some other independent non-linear operations and some delays. In the base of the tree, non-linear operations depend on only some delays.

We note that the output of the parts can be of the following two types: *linear output* - the output does not depend on any non-linear operations, *non-linear output* - the output depends on some non-linear operations. We can further classify the non-linear output into one with only one non-linear operation and one with more than one non-linear operations. The first case is interesting to consider because it gives us some insights on the effect of the non-linear operations on the critical path length (CPL) of the output cone.

5.2.1 Linear Output

If an output does not depend on any non-linear operations, the output can be represented as linear combination of the delays which it depends on. The throughput of the output cone can be optimized using the maximally-fast approach in [9].

5.2.2 Output with One Non-Linear Operation

We consider an output cone which includes only one non-linear operation and a number of linear operations. This special case is interesting to consider because it gives us some insights on the effect of the non-linear operations on the critical path length (CPL) of the computation. Suppose that the CPL of the output cone by the maximally fast technique is $l + 1$, assuming all linear operations in the output cone. Let n be the number of delays for the output cone. It follows that $2^{l-1} < n \leq 2^l$. Let x be the maximum number of delays that the inputs of the non-linear operation depend on. The fastest way to compute the output of the non-linear operation is that all the inputs are computed by the maximally fast technique and then the non-linear operation is performed. Thus, the CPL for the non-linear part is $\lceil logx + 2 \rceil$. We treat this output as an additional input to the maximally fast tree for the whole output cone. Of course, the available time of this input depends on the CPL for the non-linear part. There are four cases to consider in terms of the CPL of the non-linear output cone comparing with that of the corresponding linear output cone. The worst case arises when x is the same as n. In this case, the CPL for the non-linear output cone increases by three from that of the corresponding linear output cone.

1. **no increase in CPL:** This case happens when $p + n \leq 2^l$, where $p = 2^m$ and m is the smallest integer such that $2^m \geq 4x$.
2. **increase in CPL by one:** This case happens when there is no p that satisfies the case 1 and $x \leq 2^{l-2}$.
3. **increase in CPL by two:** This case happens when $2^{l-2} < x \leq 2^{l-2} + 2^{l-1}$.
4. **increase in CPL by three:** This case happens when $2^{l-2} + 2^{l-1} < x \leq n$

From this analysis, we note that the maximum number of delays that the inputs of a non-linear operation depend on should be minimized to optimize the critical path length of the computation.

5.2.3 Output with k Independent Non-Linear Operations

We assume that the output cone of each independent non-linear operation has already been recursively optimized. We also assume that the linear part consists of d delays. Our goal is to find the fastest tree structure that maintains the original input/output relationship. It is equivalent to find the fastest tree structure of all inputs which consist of both the outputs of the k non-linear operations and d delays, where the inputs may arrive at different timing based on the critical path length of the non-linear output cone of the non-linear operation.

Algorithm for Throughput Optimization of k Independent Non-Linear Operations and d Delays

1. Sort the k independent non-linear operations and d delays in the decreasing order of critical path length. Let the non-linear operations and delays $n_1, \cdots, n_k$ and $D_1, \cdots, D_d$, respectively in the order;
2. Connect the input n_1 to a multiplication with constant. The output of the multiplication is connected to an addition. CPL = height(n_1) + 2, where height(n_1) is the height of the output cone for the input n_1;
3. Next_Input = Build_Tree(n_2, CPL-1);
4. Connect the tree built to the addition;
5. While (Some inputs are left out of the tree) {
6. Connect the output of the addition to another addition a_n. CPL = CPL + 1;
7. Next_Input = Build_Tree(Next_Input, CPL-1);
8. Connect the tree built to the addition a_n;
9. }
10. Return the tree built;

Build_Tree(n, C)

1. Connect the input n to a multiplication with constant. The output of the multiplication is connected to an addition. CPL = height(n) + 2, where height(n) is the height of the output cone for the input n. Let Next_Input be the next input of the input n in the sorted list;
2. **If** (height(Next_Input) = height(n)) {
   Connect the input Next_Input to a multiplication with constant. The output of the multiplication is connected to the addition;
3. } **Else** {
4. Next_Input = Build_Tree(Next_Input, CPL-1);
5. Connect the tree built to the addition;
6. }
7. While (CPL $< C$ and some inputs are left out of the tree) {
8. Connect the output of the addition to another addition a_n. CPL = CPL + 1;
9. Next_Input = Build_Tree(Next_Input, CPL-1);
10. Connect the tree built to the addition a_n;
11. }
12. Return the tree built and the Next_Input;

Figure 2: Pseudo code for throughput optimization of output with k independent non-linear operations and d delays

We now describe an algorithm to solve the problem optimally. The pseudo code of the algorithm is shown in Figure 2. We first sort the inputs (k non-linear output cones and d delays) in the order of decreasing critical path length. We process the inputs in this order. Obviously, the d delays have their critical path lengths of 0. We solve a problem to build a tree with the smallest height for all the inputs. In the first step, the input n_1, the output of the non-linear operation n_1, is connected to a multiplication with constant and then the output of the multiplication is connected to one input of an addition a_1. The constant of the multiplication can be determined from the original computation. The critical path length has been increased by two from the critical path length of the output cone of the non-linear operation n_1. In the next step, we solve a problem where for the other input of the addition a_1, we need to include as many inputs as possible that does not increase the current critical path length. If any inputs are left out of the tree built, then the output of the addition a_1 is connected to an input of the addition a_2. Now the critical path length has been increased by one. For

Design	ICPL	CPL of [2]	CPL of New Method	% Red. From ICPL	% Red. From CPL of [2]
1st order Volterra filter	7	6	5	29	17
2nd order Volterra filter	12	9	6	50	33
3rd order Volterra filter	17	12	8	41	33
4th order Volterra filter	22	16	9	59	44
5th order Volterra filter	27	19	9	67	53
3rd order Noise Sharper	11	11	5	55	55
5th order Noise Sharper	15	15	6	60	60
ADPCM	72	5	3	96	40
Echo Canceller	320	7	5	98	29
DAC	220	2	2	99	0
Vocoder	110	5	4	95	20

Table 1: Throughput optimization of non-linear real-life designs with no unfolding: ICPL - initial critical path length

Design	ICPL	CPL of [2]	CPL of New Method	% Red. From ICPL	% Red. From CPL of [2]
1st order Volterra filter	7	6	4	43	33
2nd order Volterra filter	12	9	5	58	44
3rd order Volterra filter	17	12	6	65	50
4th order Volterra filter	22	16	6	73	62
5th order Volterra filter	27	19	6	78	68
3rd order Noise Sharper	11	11	4	64	64
5th order Noise Sharper	15	15	5	67	67
ADPCM	72	$\rightarrow 0$	$\rightarrow 0$	-	-
Echo Canceller	320	$\rightarrow 0$	$\rightarrow 0$	-	-
DAC	220	$\rightarrow 0$	$\rightarrow 0$	-	-
Vocoder	110	$\rightarrow 0$	$\rightarrow 0$	-	-

Table 2: Throughput optimization of non-linear real-life designs with unrestricted amount of unfolding used: ICPL - initial critical path length, $\rightarrow 0$ - arbitrary speed-up

the other input of the addition a_2, we include as many inputs as possible that does not increase the current critical path length. We repeat this process until all inputs are included in the tree.

Note that the maximally fast tree in [9] is a solution for a special case of this problem when all the inputs arrive at time 0. We can verify that the new algorithm constructs the tree with the same height as the maximally fast tree for the special case.

6 Experimental Results

We have used several Volterra filters of different orders, several Noise Sharpers, Echo Canceller, Vocoder, NEC DAC (digital-to-analog converter), and ADPCM (adaptive delta pulse code modulation) for experimental results. Table 1 shows throughput improvement achieved using the best previous approach by [2] and using the new approach with optimization engine of general non-linear computation with no unfolding. Table 2 shows throughput improvement achieved for non-restricted amount of unfolding.

When no unfolding is used, our approach reduces the critical path lengths from the technique of [2] by 35 % on average. When unfolding is used, for the five different Volterra filters, we improve the critical path lengths from the technique of [2] by more than 50 %. For the other designs, both the technique of [2] and ours result in arbitrary speed-up of throughput.

7 Conclusion

This paper addressed an optimal approach for throughput optimization of general non-linear computations using a set of transformations. We demonstrated the effectiveness of the new technique on numerous real-life non-linear designs. When no unfolding was used, our approach reduced the critical path lengths from the previous best technique by 35 % on average. When unfolding was used, we improved the critical path lengths from the previous best technique by more than 50 % for many designs.

REFERENCES

[1] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Brodersen. Optimizing power using transformations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(1):12–31, 1995.

[2] L. Guerra, M. Potkonjak, and J. Rabaey. Divide-and-conquer techniques for global throughput optimization. In *IEEE Workshop on VLSI Signal Processing*, pages 137–146, 1996.

[3] H. T. Kung. New algorithms and lower bounds for the parallel evaluation of certain rational expressions and recurrences. *Journal of the ACM*, 23(2):252–261, 1976.

[4] G. Lakshminarayana and N.K. Jha. FACT: a framework for the application of throughput and power optimizing transformations to control-flow intensive behavioral descriptions. In *Design Automation Conference*, pages 102–107, 1998.

[5] E.A. Lee and D.G. Messerschmitt. Synchronous dataflow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[6] D. G. Messerschmitt. Breaking the recursive bottleneck. In J. K. Skwirzinsky, editor, *Performance Limits in Communication Theory and Practice*. Kluwer Academic Publisher, Amsterdam, 1988.

[7] S. K. Mitra and J. F. Kaiser. *Handbook for Digital Signal Processing*. John Wiley, New York, NY, 1993.

[8] K.K. Parhi and D.G. Messerschmitt. Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding. *IEEE Transactions on Computers*, 40(2):178–195, 1991.

[9] M. Potkonjak and J. Rabaey. Maximally fast and arbitrarily fast implementation of linear computations. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 304–308, 1992.

[10] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak. Fast prototyping of data path intensive architectures. *IEEE Design & Test of Computers*, 8(2):40–51, 1991.

[11] R.E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.