DIVIDE-AND-CONQUER TECHNIQUES FOR GLOBAL THROUGHPUT OPTIMIZATION

Lisa Guerra†, Miodrag Potkonjak‡, Jan Rabaey†

†EECS Dept., University of California, Berkeley, CA

‡CS Dept., University of California, Los Angeles, CA

Abstract - This paper proposes a divide-and-conquer approach for global throughput optimization designed to coordinate existing techniques and enable their more effective use. The "divide" approach consists of logical partitioning of the computation into subparts. The techniques for partitioning the computation, and the corresponding scheme for classifying the subparts is presented. The subparts are optimized or "conquered" through coordinated application of existing optimization techniques. Optimization techniques that are effective for each class have been characterized in terms of their expected effect on throughput. The approach is not limited to a specific class of computations and gives higher, or at least equal, improvement than previously reported techniques on all examples.

INTRODUCTION

Throughput optimization techniques remain important in meeting the sampling rate requirements of modern DSP and communication applications. Though clock rates for ASICs and general purpose computing devices have been doubling every two years, required sampling rates have been increasing at even higher rates [1]. Additionally, techniques for improving throughput are widely used in the optimization of other design metrics such as area and power.

Numerous techniques for throughput optimization have been proposed; many are based on transformations, but some are based on other approaches such as operation chaining and clock selection. In addition to the development of *individual* techniques, several approaches have dealt with the ordering and coordination of *sets* of techniques [2], [3], [4], [5].

In this paper, a new methodology for sampling rate *maximization* of *arbitrary* computations is presented. The methodology is designed to effectively leverage upon, combine, and coordinate the application of sets of optimization techniques. While previous approaches have dealt with the throughput maximization of specific domains of computations (e.g. non-recursive or linear computations), this approach targets arbitrary computations (computations can be both recursive and non-linear). Furthermore, unlike many previous approaches, this approach is not limited to the use of a specific, fixed set of techniques. A wide range of existing and future techniques can be easily incorporated and utilized.

0-7803-3134-6/96 $5.00 © 1996 IEEE

Overview of the Divide-and-Conquer Technique

The divide-and-conquer approach is an iterative logical partitioning of a computation into isolated subparts, which can then be individually optimized. The strategy is recursive, in that optimization of a subpart may itself involve further dividing and conquering. The overall optimized critical path is the maximum critical path among all parts.

The purpose of *dividing* is to break a difficult problem into more manageable parts. Our divide step both identifies and isolates subparts. As a result, the overall critical path lies within only one of the parts, and independent focused optimization of each computation subpart in accordance with its topological structure is enabled. An important ramification is that this enables the use of techniques which, while not applicable to the entire computation, can be applied to individual subparts.

Conquering involves optimizing each part. To effectively conquer, we have formulated a classification of computations, and have characterized optimization techniques that are effective for each. Any existing or future optimization technique can be utilized in this phase.

Time-loop unfolding, for simultaneous processing of consecutive iterations of a computation, is also used in the divide-and-conquer approach. Time-loop unfolding is a powerful enabler that enhances the effect of subsequent optimizations.

Motivational Example

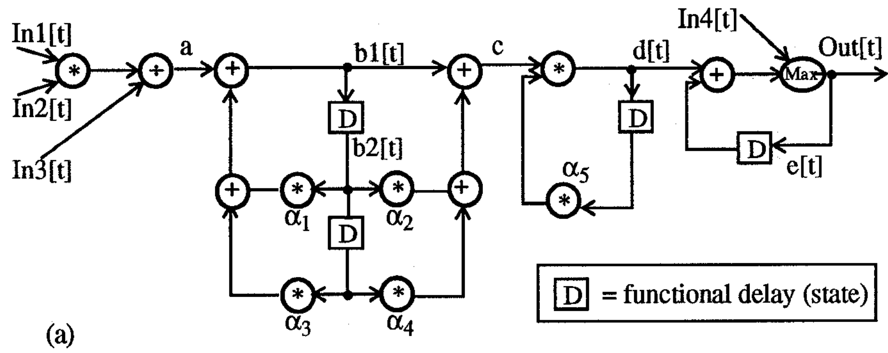
To illustrate the key ideas behind our approach, consider the computation of Figure 1a. The computation has sub-parts with key structural properties similar to those found in many DSP and communication systems (e.g. quantizers, linear and non-linear filters, adaptive computations, and Viterbi decoders).

The critical path is seven clock cycles, assuming each operator takes one clock cycle. As the structure is highly non-linear and recursive, techniques for arbitrary[1] speedup [6], [7], [8] do not apply. 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 three clock cycles.

Using the divide-and-conquer technique proposed in this paper, much better improvements can be attained. Figure 1b shows the example after unfolding so that two iterations of the computation are performed simultaneously. Subparts are identified and partitioned by inserting pipeline delays (at the dashed lines). The pipeline delays isolate each subpart from the remainder of the computation, thus enabling independent subpart optimization.

Consider the optimization of the sections one-by-one. Section 1 is non-linear, but has no feedback, so it can be easily optimized using known techniques. In general, the effective critical path of any non-recursive section can be reduced an arbitrary extent using unfolding and pipelining; in particular, it is reduced to 1/(k+1), where k is the unfolding factor. For this section, pipelining reduces the critical path to 1

1. In a theoretical sense, though its implementation may be practically infeasible.



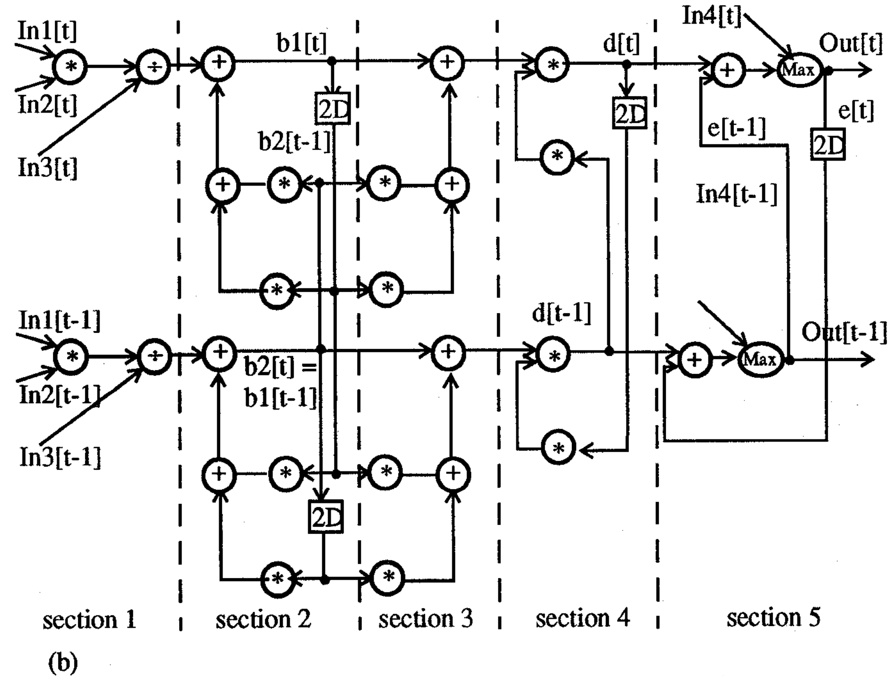


Fig. 1. Motivational example a) original b) unfolded by a factor of 1

clock cycle, and since two iterations are being computed, the effective critical path per iteration is 1/2 clock cycle. Section 3, which is linear and non-recursive, can likewise be optimized to a critical path of 1/2.

Consider next section 2, which is recursive but linear. Since this section is linear, it can also be handled using known techniques. Notice that the critical paths of this section before and after unfolding are 3 and 6, respectively. The critical path in the unfolded structure can be reduced from 6 to 3, however, using the "maximally fast" technique of [8]. Since two iterations are computed, the effective critical path of section 2 is 3/2, or 1.5. Since the critical path can be reduced to 3 regardless of the number of unfoldings, arbitrary levels of speed up are in fact attainable. For a given

```
divideAndConquer (Graph) {
        parts <- divideAndSort(Graph);
        foreach (j in parts) {
                if ( IsHandleable(j) ) {
                        conquer(j);
                } else {
                        divideAndConquer(j);
                }
        }
}
```

Fig. 2. Core of the divide-and-conquer approach

unfolding factor, k, the critical path can be reduced to 3/(k+1). Section 4 which is feedback linear [8] and Section 5 which is linear can both be optimized using this same technique to achieve effective critical paths of 1.

With the optimization of each subsection complete, the computation's resulting overall critical path is 1.5, the maximum critical path among all 5 stages (for unfolding by a factor of 1). Once the critical path of the limiting section is determined (in this case section 2), it is only necessary to speed up other sections to that extent. Since each subpart can be arbitrarily sped up using larger unfolding factors, the overall computation can also be arbitrarily sped up.

This section has introduced the divide-and-conquer approach, and illustrated its effectiveness for arbitrary speed up of the computation of Figure 1, despite non-linearities and cyclic structures. Notice that while both sections 2 and 5 are each individually linear, the computation containing them is not, and cannot be arbitrarily sped up using previously published techniques.

PRELIMINARIES

Before detailing the divide-and-conquer methodology and techniques, this section presents several underlying assumptions and definitions. Computations are represented by a hierarchical data-control flowgraph consisting of nodes representing data operators or sub-graphs, and edges representing the data, control, and timing precedences. The computations operate on periodic semi-infinite streams of inputs to produce semi-infinite streams of outputs. The underlying computation model is homogeneous synchronous data flow [9], a model widely used in application domains such as DSP, video and image processing, communications, and control. Under this model, the operators consume a single sample from each input and produce a single sample on each output, on every execution. Note that implementations are driven by an on-chip clock, and thus operation delays are quantized into an integral number of clock cycles.

The maximum throughput rate at which a design can process incoming samples is the inverse of the critical path length, where a path through the computation starts at any primary inputs or delay and ends at any primary output or delay.

DIVIDE-AND-CONQUER APPROACH

The core of the approach is presented in the pseudo-code of Figure 2. The

strategy is a hierarchical iterative divide-and-conquer optimization, which is embedded in an outer loop which finds the optimal unfolding factor. The following sub-sections explain the key ideas behind the strategy.

Dividing the Computation

The goal of dividing is to isolate parts so that the overall critical path lies within only one of the parts, and to enable focused optimization of each computation subpart in accordance with its topological structure. For throughput optimization, a computation's level of non-linearity and cyclic dependencies, give an accurate indication of the difficulty of speeding it up. For example, any computation that is non-recursive, linear, or feedback linear, can be sped up arbitrarily using known techniques [6], [7], [8].

The divide step partitions the computation by identifying and characterizing mutually exclusive subparts which fall into one of a finite set of classes represented by the leaves of the classification tree shown in Figure 3. For each class, different combinations of optimization techniques may be used. Optimizations will be discussed in the next section.

There are three hierarchy levels in the partitioning process. The first level in the classification tree involves identifying operations inside feedback cycles, and those outside of feedback cycles. This is done by identifying the computation's strongly connected components (SCCs), using a depth-first search. 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 trivial SCCs (those with a single operation) are not part of a feedback cycle (class 1). Note that while the number of cycles may be an exponential function of the number of operations, the number of SCCs is linear in the number of operations and can be efficiently identified.

Next, non-trivial SCCs are isolated from each other and from non-recursive parts using pipeline delays. During subpart optimization, the bordering pipeline states are treated like the subpart primary inputs and outputs. Once this is done, all primary outputs and states within a subpart depend only on that subpart's primary inputs and states. Since none of the other states in the computation and none of the primary inputs to other subparts affect it, the subpart is effectively isolated.

The non-trivial SCCs are further classified as being either linear or non-linear. In

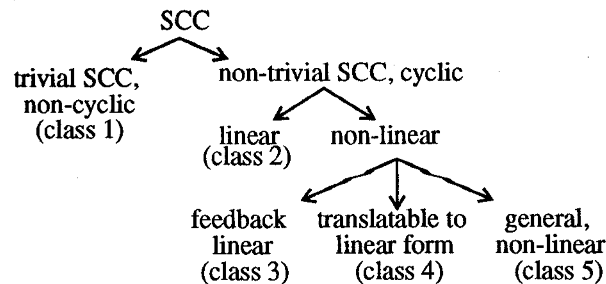


Fig. 3. SCC classification

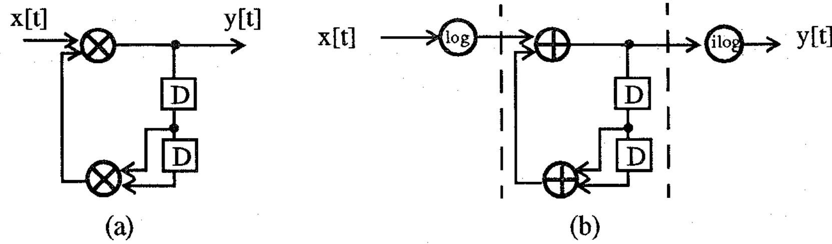


Fig. 4. a) Multiplication-only non-linear feedback structure b) after the application of the logarithmic identity

linear computations (class 2), the next state and outputs are linear functions of the previous states and inputs. Linearity is not restricted to arithmetic + and *. Systems containing only min and + operators or only max and + operators are also linear.

The non-linear SCCs are further classified as being either feedback linear [8], being transformable to a form in which all non-linearities are moved outside of the SCC, or being neither. A feedback linear SCC (class 3) is a computation which can contain not only addition and constant multiplications, but also variable multiplications. However one operand of each variable-multiplication must be generated by an operation that lies outside of the SCC.

A computation containing only variable multiplications (Figure 4a) is an example of a recursive non-linear computation which can be translated into a form with all non-linearities outside of the SCC (class 4). Using the logarithmic identity $\log(x_1 \cdot x_2) = \log(x_1) + \log(x_2)$, we can eliminate variable multiplications. Sign bit and zero checking can be done with a small amount of additional logic. Other class 4 computations include those which have only variable multiplication and variable divisions and those which have only maximum and multiplication operations.

Within each general non-linear SCC, further partitioning is done (Figure 5). The key idea is that while an SCC as a whole may be non-linear, many of its parts often are easily optimized. Isolation in this case is not achieved by inserting pipeline delays (which is infeasible within a SCC) but by separating the SCC's sub-computations for each of its primary outputs and next states (each sub-computation is called a cone, since it has only one "output"). Partitioning into cones reduces sharing of common sub-expressions, which can enable more effective subsequent

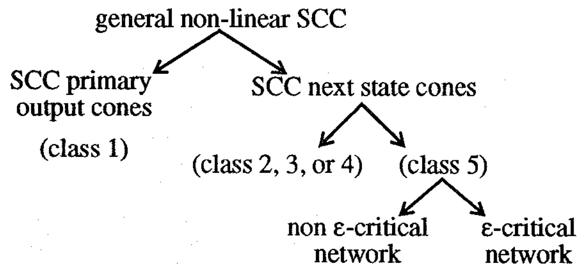


Fig. 5. Partitioning and classification of general non-linear SCCs

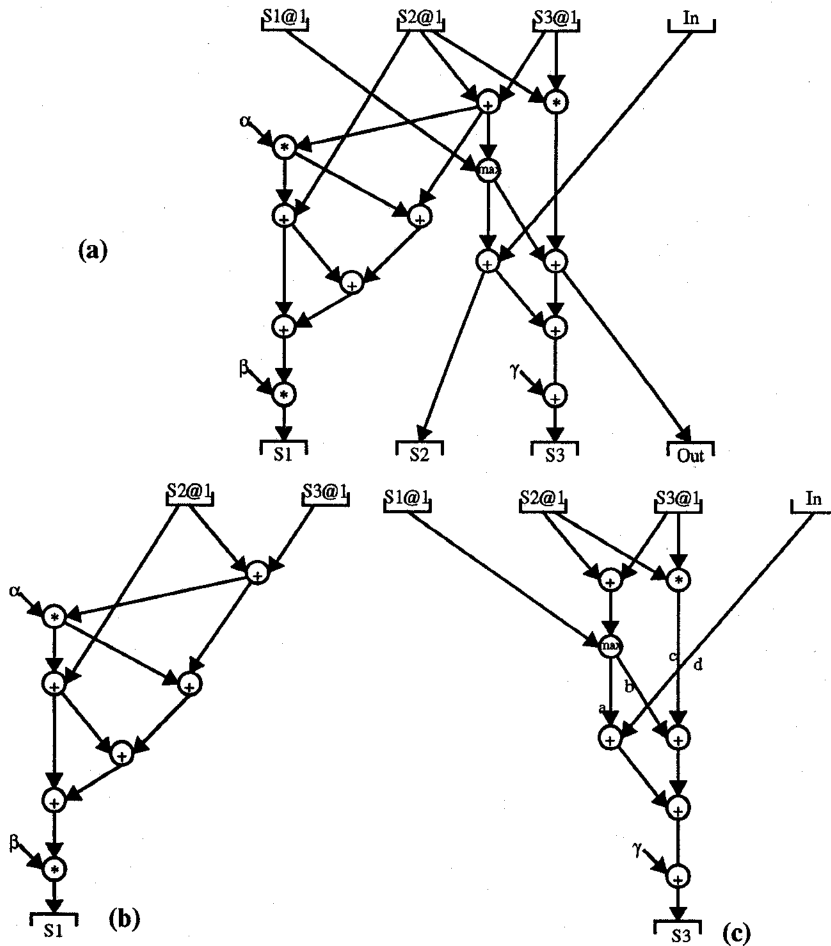


Fig. 6. a) General non-linear SCC, b) S1 cone, c) S3 cone. S2 and output cone are not shown.

optimization. Cones are optimized independently; the resulting critical path is determined by the longest critical path among cones.

Since the primary outputs by definition are a non-recursive function of all inputs and next states, their sub-computations fall into class 1. Each of the next state sub-computations fall into one of classes 2 - 5. For next state cones of type 5, a final partitioning is done, into ε-critical network[2] and non-ε-critical network parts. An example of cone partitioning is shown in Figure 6.

Ranking the Optimization Difficulty

Since the computation's overall critical path is determined by the maximum critical path among all SCC subparts, it is limited by the SCC whose optimized critical path turns out to be the greatest. Similarly, a non-linear SCC's critical path

2. The ε-critical network contains all paths of length greater than (1-ε)•critical path length.

is limited by the cone whose optimized critical path is largest. Since global decisions (e.g. select clock period, types of modules, voltage, etc.) are made just once yet affect all parts, we allow them to be made to aid the optimization of the subpart whose optimization is most difficult. Furthermore, once the limiting section's critical path is determined, we can use this information to avoid over-optimizing other sections.

For all subparts in classes 1 - 4 arbitrary speed up is achievable. The subparts in class 5 are ranked using the following metrics; the greater their values, the more difficult the optimization is expected to be:

1. Formal degree - degree of the largest polynomial computed. For example, a linear computation has a formal degree of 1, and the computation $y = x^3 + x^2$ has a formal degree of 3. This is a measure of the computation's non-linearity.

2. Ratio of edges originating within the subpart to the total subpart edges. This is based on our observation that optimization when operators (e.g. variable multiplication) have both operands originating in the SCC, versus when one operand originates externally is much more difficult.

3. Concurrency ratio - ratio of subpart number of operations to subpart critical path. This value is 1 for sequential computations, and equals the number of operations in a maximally parallel computation. The more sequential the computation, the more potential for critical path optimization.

Enabling Using Time-Loop Unfolding

A key part of the divide-and-conquer approach is unfolding the time-loop by a factor of i. This results in simultaneous computation of i+1 iterations of the computation, as opposed to sequential treatment of consecutive iterations. For computations that can be arbitrarily sped up, the throughput will continue to improve with larger factors of unfolding. For other computations, it will at some point asymptotically level off with greater factors. The optimal unfolding factor can in some instances be exactly calculated or at least bounded (e.g., [11]). In general, different unfolding factors must be tried to find the best one. We use an outer loop surrounding the core divide-and-conquer optimization to systematically search for the smallest optimal unfolding factor using a binary search.

The search starts with an unfolding factor of 1. As long as the throughput continues to improve, the factor is doubled and the core divide-and-conquer optimization is re-run. Once improvement ceases, a binary search between the current factor, i, and previously tried factor, i/2, is done to find the smallest optimal factor.

CONQUER OPTIMIZATION TECHNIQUES

Once the computation has been divided into parts, each part is individually optimized. Identifying and characterizing effective techniques for this task is thus essential. We have compiled an index of synthesis and transformation actions for throughput optimization. Currently, we have 27 techniques characterized, some of which are individual actions and some of which are short static scripts of actions.

Additional techniques may be incorporated as they are developed and identified.

For the purpose of maximizing throughput, a small set of techniques is sufficient for speeding up parts in classes 1 - 4. As is summarized in Table 1, arbitrary speed up is attainable for these computations. Optimization of general non-linear parts is more challenging. Arbitrary speed up is not known to be achievable for this class of computation. Kung's proof [12] that speed up is limited to a constant factor for some classes of computations, while made under a slightly different set of assumptions, suggests that there indeed exist computations that cannot be sped up.

Using the divide-and-conquer approach, however, non-linear algorithms, are optimized efficiently since the non-linearities are isolated into SCC cones. For optimization of these general non-linear parts, a heuristic search is necessary to find an effective combination of techniques such as algebraic manipulations, redundancy manipulations, template matching, module selection, and parameter selection (clock period and voltage). Computational complexity is further reduced, since this search is just applied to the ε-critical network of an SCC cone.

Class	Technique	Effective critical path
1: non-recursive	unfolding	cp / (k+1)
	unfolding + pipelining	dur_{max} / (k+1)
2: linear	maximally fast [8]	$\langle \frac{\lceil \log_2(NumStates + 1) \rceil + dur_{mult}}{(k+1)} \rangle$
3: feedback linear	same as class 2	
4: translatable to linear form	combination of class 1 and class 2 techniques	

Table 1: Summary of selected techniques for arbitrary speedup-- cp: initial critical path; k: unfolding factor; dur_{max}: max. operator duration; dur_{mult}: multiplication duration; NumStates: number of states

EXPERIMENTAL RESULTS

Table 2 shows throughput improvements achieved using the best previous approach and using the divide-and-conquer approach. Results for both no unfolding and non-restricted amounts of unfolding are shown. In all examples, the new approach yields an arbitrary level of throughput improvement (column 6), while previously published techniques could achieve an arbitrarily high throughput only on completely linear examples (column 5). Even when unfolding is not used, the new technique significantly outperforms previous techniques (columns 3 and 4).

The new technique also yields significantly lower area overhead for the same level of performance. For example, application of the "maximally fast" technique for reducing the DAC critical path to 10 cycles requires 5,000 multipliers and 4,750 adders, while the new technique requires only 240 multipliers and 228 adders, reduction by a factor of more than 20 times. Note that in modern sub-micron technologies the new technique results in practically realistic implementations,

while the "maximally fast" implementation is prohibitively expensive.

Name	Initial critical path	No unfolding		Best previous	New technique
		Best previous	New technique		
DAC[a]	220	10	2	10	arbitrary
Echo Canceler	320	50	7	50	arbitrary
Avenhaus Filter	10	5	3	arbitrary	arbitrary
Polyphase Decimation Filter	21	6	6	6	arbitrary
DCT/DPCM[b]	72	17	5	17	arbitrary

Table 2: Experimental results

a. NEC digital-to-analog converter
b. DCT- domain delta pulse code modulation [13]

SUMMARY

We have proposed a divide-and-conquer approach for throughput optimization which not only leverages upon existing techniques, but enables their more effective and coordinated use. The technique is not limited to a specific class of computations and gives higher, or at worst equal, improvement than previously reported techniques on all examples. While the approach has been presented solely for throughput maximization, its effectiveness for combined throughput-area or throughput-power optimization is being further investigated.

REFERENCES

[1] W.W. Gibbs, "Software's chronic crisis," *Scientific Amer.*, pp. 86-95, Sept. 1994.

[2] W.M. McKeeman, "Peephole optimization," *Communications of the ACM*, Vol. 8, No. 7, pp. 443-444, 1965.

[3] M.E. Wolf, M.S. Lam, "A loop transformation theory and an algorithm to maximize parallelism," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 2, No. 4, pp. 452-471, 1991.

[4] S. Huang, J. Rabaey, "Maximizing the throughput of high performance DSP applications using behavioral transformations," *EDAC*, pp. 25-30, 1994.

[5] D. Whitfield, M. Soffa, "An approach to ordering optimizing transformations," *ACM Symp. on Principles and Practice of Parallel Program.*, pp. 137-147, 1990.

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

[7] K. K. Parhi, D. Messerschmitt, "Pipeline interleaving and parallelism in recursive filters, part 1," *IEEE Trans. on ASSP*, Vol. 37, pp. 1099-1117, 1989.

[8] M. Potkonjak, J. Rabaey, "Maximally fast and arbitrarily fast implementation of linear computations," *Int'l Conf. on Computer-Aided Design*, pp. 304-308, 1992.

[9] E.A. Lee and D.G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. on Computers*, Vol. 36, No. 1, pp. 24–35, 1987.

[10] A. Fettweis, H. Meyr, L. Thiele, "Algorithm transformations for unlimited parallelism," *IEEE Int'l Symp. on Circuits and Systems*, pp. 1756-1759, 1990.

[11] M. Srivastava, M. Potkonjak, "Energy efficient implementation of linear systems on programmable processors," *IEEE Workshop on VLSI Signal Proc. VIII*, 1995.

[12] H.T.Kung, "New algorithms and lower bounds for parallel evaluation of certain rational expressions and recurrences," *JACM*, Vol. 23, No. 2, pp. 252-261, 1976.

[13] J. Jain, A. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Trans. on Comm.*, Vol. 29, No. 12, pp. 1799-1808, 1981.