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 which not only leverages upon existing techniques, but enables their more effective and coordinated use. The "divide" approach consists of logical partitioning of the computation into subparts falling into one of a set of pre-classified computation types. The subparts are then "conquered" through coordinated application of existing optimization techniques. We have characterized a set of techniques in terms of their expected effect on throughput, and can thus select the most promising techniques for each unique situation. 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.

1.0 Introduction

Throughput optimization techniques remain important for 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 [Gib94]. Techniques for reducing throughput are also widely used during the optimization of other design metrics such as area and power. For example, a highly effective power minimization techniques involves using throughput optimization to enable voltage scaling [Cha92].

Numerous approaches have been proposed for throughput optimization. However, most have limited scope with respect to either application domain or the set of techniques utilized. As a consequence, when applied to modern non-linear, large, and structurally complex DSP computations, they have limited effectiveness.

Our goal in this paper is to demonstrate a new methodology for sampling rate maximization of arbitrary computations which utilizes and effectively combines existing techniques. We have two primary objectives. The first involves developing an approach which will cleanly, intuitively, and effectively leverage on already available optimization techniques. The second involves providing a flexible and extensible methodology wherein modularity is enforced and a multitude of existing and future techniques can be easily incorporated.

To accomplish the targeted goals we have developed a new divide-and-conquer approach for global throughput optimization. An overview of the approach is given in the next section.

1.1 Overview of the Divide-and-Conquer Technique

The divide-and-conquer approach involves iterative logical partitioning of the computation into isolated subparts, enabling the use of efficient techniques which, while not applicable to the entire computation, can be applied to individual subparts to result in optimization of the overall computation.

Dividing involves breaking a difficult problem into manageable parts. For throughput optimization, manageability is determined by a computation's level of non-linearity and cyclic dependencies, since they 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[1] using known techniques [Mes88, Par89, Pot92]. In order to allow independent optimization of subparts, our divide step both identifies and isolates subparts.

Conquering involves optimizing each part. We have formulated a classification of computations, and have characterized optimization techniques that are effective for each. The strategy is iterative, 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 approach also involves using time-loop unfolding as an optimization enabler. Time loop unfolding allows simultaneous processing of consecutive iterations of the computation, and has been established as a powerful enabler in that it enhances the optimization effect of subsequent optimizations.

1.2 Motivational Example

To illustrate the key ideas behind the new approach, consider the computation of Figure 1a. Communication systems, for example, often include quantizers, linear and non-linear filters, adaptive computations, and Viterbi decoders. Figure 1a has sub-parts with key structural properties similar to these typical DSP and communication applications.

Figure 1a's critical path is seven clock cycles, assuming each operator takes one clock cycle. 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 three clock cycles. The exhaustive application of the other popular throughput optimization techniques also provide limited performance improvements. The problem is that no single technique is well suited for addressing this highly non-linear and recursive structure.

Using the divide-and-conquer technique proposed in this paper, much better improvements can be attained. Figure 1b shows the motivational example after unfolding; two iterations of the computation are now performed simultaneously. Subparts are identified, and partitioned by pipeline delays delineated by the dashed lines in the figure. This isolates and shields each subpart from the rest of the computation, to enable independent optimization, or "conquering."

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. Pipelining can be used to reduce this section's critical path to 1 clock cycle. 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 be optimized in the same way, to also achieve a critical path of 1/2. In general, the critical path of any non-recursive section can be reduced to an arbitrary extent using unfolding and pipelining; in particular, it is reduced to 1/(k+1), where k is the unfolding factor.

Consider next section 2, which is linear and has feedback. 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,

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

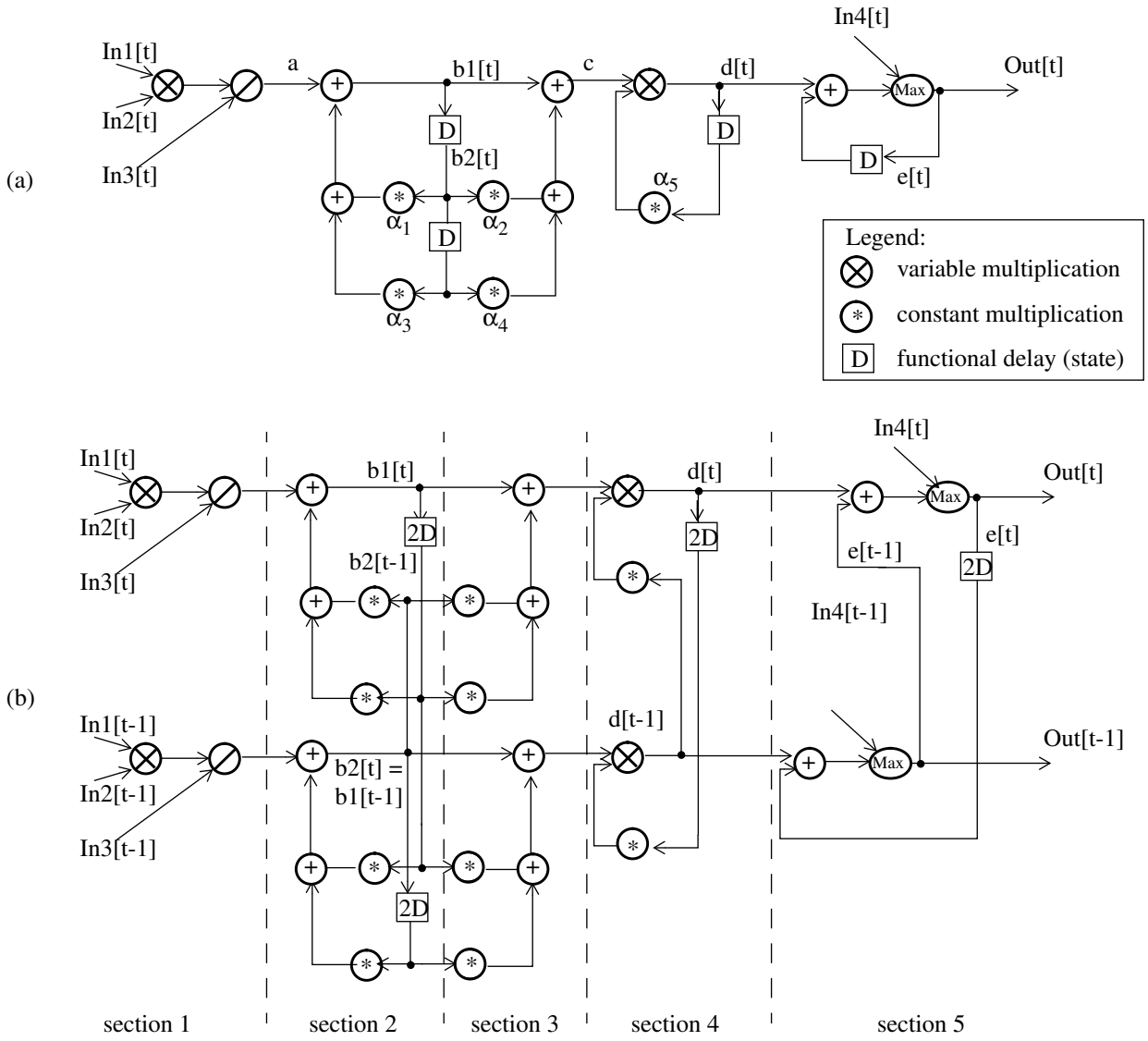

Fig. 1. Motivational example a) original b) unfolded by a factor of 1

respectively (the effective critical path remains 3 after unfolding). As illustrated in Figure 2, the critical path in the unfolded structure can be reduced from 6 to 3, however, using the "maximally fast" technique of [Pot92]. The technique applies a static script of algebraic and redundancy manipulation transformations to reduce the critical path to $\lceil \log_2(NumStates + 1) \rceil + 1$, where *NumStates* is the number of internal states that have to be computed (e.g. section 2 has two internal states, b1 and b2). Since two iterations are computed, the effective critical path of section 2 is 3/2, or 1.5. Note that since *NumStates* stays constant regardless of the number of unfoldings, arbitrary levels of speed up are attainable. In general, the "maximally fast" optimization applies for any linear or feedback linear computation. Section 4 which is feedback linear and Section 5 which is linear can both be optimized in the same way to achieve effective critical paths of 1. A key point to note is that while both sections 2 and 5 are each individually linear, a computation containing both is not, and cannot be optimized using previously published techniques. The divide-and-conquer methodology's technique of

isolating computation subparts enables the use of efficient techniques which, while not applicable to the entire computation, can be applied to individual subparts to result in optimization of the overall computation.

With the optimization of each subsection complete, the computation's resulting overall critical path is the

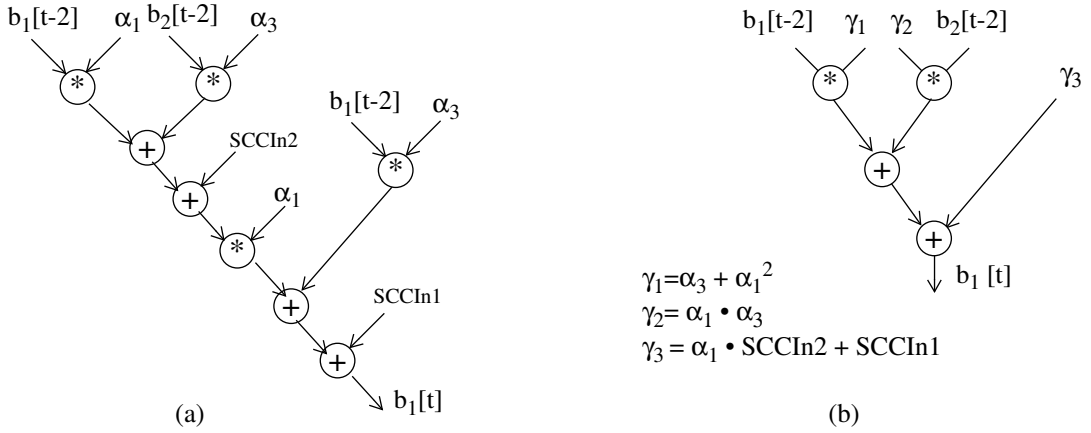

Fig. 2. Critical path a) before b) after the application of the maximally fast technique

maximum critical path among all 5 stages. In this case, the effective critical path is the max(0.5, 1.5, 0.5, 1, 1), or 1.5 (two iterations of the computation are completed in three clock cycles). More importantly, as mentioned above, since each subpart can be arbitrarily sped up using larger unfolding factors (without influencing other parts and while preserving functionality), the overall computation can also be arbitrarily sped up. In particular, for a given unfolding factor, k, Figure 1's critical path can be reduced to 3/(k+1).

This section has illustrated the use of the divide-and-conquer technique for arbitrary speed up of Figure 1, despite its non-linearities and cyclic structure. As will be shown in this paper, the new throughput optimization methodology can be used for efficient optimization of any structure (though not always to arbitrary levels), by both leveraging upon existing techniques, and enabling their more effective and coordinated use.

1.3 Paper Organization

The remainder of this paper presents the divide-and-conquer methodology and techniques. Section 2.0 presents background material about the computational and hardware models, and defines nomenclature. Section 3.0 presents related work, and highlights the new contributions of this work. Section 4.0 presents the global divide-and-conquer strategy. Section 5.0 highlights the optimizations used in the "conquer" step. Finally, sections 6.0 and 7.0 present experimental results and conclusions, respectively.

2.0 Preliminaries

We represent a computation 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 [Lee87], 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. An important ramification of the semantics of the selected model of computation is that it lends itself to efficient, static scheduling, avoiding the run-time scheduling overhead incurred by more general models. Furthermore, under this model, throughput is the relevant metric of performance.

Throughput is the maximum rate at which a design can accept and process incoming samples. The inverse of throughput is the critical path. We use the most broad definition of throughput, which considers the length of all paths which start at any of the primary inputs or delays and finish at any of the primary outputs or delays.

3.0 Related Research Efforts

Numerous techniques for throughput optimization have been proposed, many based on transformations [Mes88, Par89, Fet90], but some also based on template-matching [Not91, Cor93], and clock selection [Cor93]. Previous approaches involve not only using isolated techniques for throughput improvement, but also effectively ordering and coordinating sets of individual techniques. Popular techniques include peephole optimization [McK65], the use of static scripts [Bra84], exhaustive implicit search-based "generate and test" methods [Mas87], linear algebra-based ordering of subsets of loop-control flow transformations [Wol91], generic probabilistic iterative improvement techniques [Cha92], bottleneck removal methods [Hua94], and microscopic and special-domain enabling-effect based techniques [Whi90, Sri95a].

The proposed approach differs from previous methods in several key aspects. Firstly, it uses logical partitioning as an optimization mechanism in a simple but powerful divide-and-conquer framework. Secondly, it utilizes and coordinates the application of a much wider spectrum of point optimization techniques than was done in previous approaches. Our approach is fully modular and can be easily augmented with new throughput optimization point techniques. Finally, the approach yields an arbitrarily high improvement on a larger class of computations than previously known.

4.0 Divide-and-Conquer Throughput Optimization Approach

The core of the approach is presented in the pseudo-code of Figure 3. The strategy is a hierarchical iterative

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

Fig. 3. Core of the divide-and-conquer approach

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.

4.1 Dividing the Computation

As was illustrated in the motivational example, while a given computation may overall be difficult to

optimize, some of its subparts can be optimized easily. This observation forms a partial motivation behind our divide-and-conquer optimization strategy. Much more importantly, 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.

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 trees shown in Figure 4a. For each class, different combinations of optimization techniques may be used. These specific optimization schemes will be presented in Section 5.0.

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 the standard depth-first search-based algorithm [Tar72]. 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 thus part of a feedback cycle. Note that while there may be an exponential number of cycles, the number of SCCs is linear in the number of operations and thus can be efficiently identified.

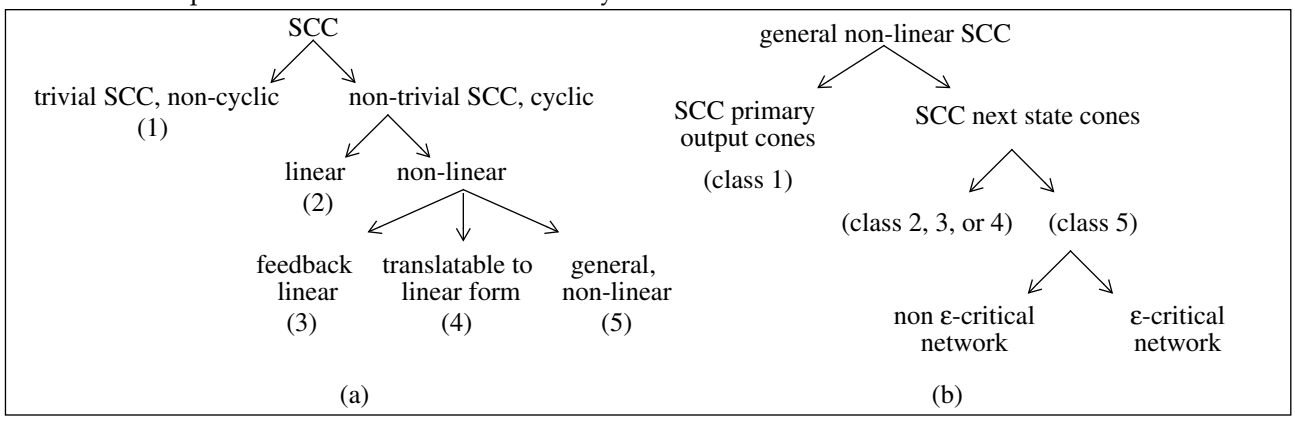

Fig. 4. a) SCC classification b) Breakdown of general non-linear SCCs

The non-trivial SCCs are isolated from each other and from non-recursive parts using pipeline delays, which allows independent optimization. During subpart optimization, the inserted pipeline states are treated like a subpart primary input or primary output, depending on whether the subpart reads from or writes to the state. Once this is done, all "primary outputs" and states within a subpart depend only on that subpart's "primary inputs," and states. None of the other states in the computation and none of the primary inputs to other subparts affect it, and thus in this sense, the subpart is isolated.

The non-trivial SCCs are further classified as being either linear or non-linear. In linear computations, the next state and the outputs are linear functions of the previous states and inputs. These equations in matrix form are: $S[n] = A \bullet S[n-1] + B \bullet X[n]$, and $Y[n] = C \bullet S[n-1] + D \bullet X[n]$, where S[n], X[n], and Y[n] are vectors of feedback states (algorithmic delays), primary inputs, and primary outputs, respectively, and A, B, C, and D are constant coefficient matrixes. It is important to note that linearity is not restricted to arithmetic + and *. A system can exhibit linearity over either min or max in place of the + operator, and arithmetic addition in place of the * operator.

The non-linear SCCs may be further classified as being either feedback linear [Pot92], being transformable to a form in which all non-linearities are moved outside of the SCC, or being neither. A feedback linear SCC 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 5a) is an example of a recursive non-linear computation which can be translated into a form in which all non-linearities are moved outside of the SCC. Using the logarithmic identity $\log(x1 \cdot x2) = \log(x1) + \log(x2)$, we can transform the computation as is shown in Figure 5b. Besides the leading and trailing logarithm and exponent (inverse logarithm) operations (note that the log must also be taken on the state initial values), the core computation is now done using addition rather than multiplication operators. Sign bit and zero checking can be done with a small amount of additional logic. Other computations which fall into this class include those which have only variable multiplication and variable divisions and those which have only maximum and multiplication operations.

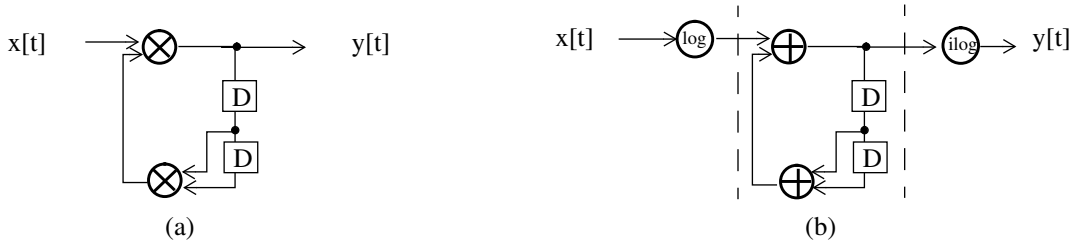

Fig. 5. a) Initial multiplication-only non-linear feedback structure b) after the application of the logarithmic identity

In total there are 5 classes of SCCs, as indicated in Figure 4a.

Within each general non-linear SCC, we have another set of partitioning and characterization techniques (Figure 4b). Isolation in this case is not achieved by inserting pipeline delays (which is anyhow 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"). As an example, Figures 6b-e show the cones for the computation of Figure 6a. Partitioning into cones eliminates sharing of some common sub-expressions, which can enable more effective subsequent optimization. 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 and non-ε-critical network parts.

4.2 Ranking the Optimization Difficulty

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

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. In general, 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 - measure of how concurrent (or sequential) the subpart is: $\frac{\text{number of subpart operations}}{\text{subpart critical path}}$. This value is 1 for sequential computations, and equals the number of operations in a maximally parallel computation. The more sequential the computation is, the more potential it has for critical path optimization.

4.3 Enabling Using Time-Loop Unfolding

Unfolding the time-loop by a factor of i results in the simultaneous computation of i+1 iterations of the computation, as opposed to sequential treatment of the 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 calculated exactly or at least bounded [e.g. Sri95b]. In the general case, different unfolding factors must simply be tried to find the best one. We use an outer loop surrounding the core hierarchical 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 above which improvement will cease.

5.0 Conquer Optimization Techniques

We have compiled an index of synthesis and transformation actions deemed most effective 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. For the sake of brevity, in this section we present only a few of the most effective for each computational class.

In the motivational example of Section 1.2, throughput optimization was done by reducing the critical path measured in clock cycles assuming that each operation takes one clock cycle to complete. The ultimate throughput optimization goal, however, is reduction of the critical path measured in real time (seconds), which is affected by not only the length of the critical path measured in clock cycles, but also by the operation delays and the clock period itself. The interdependency between these three factors is complex and well-documented (e.g. [Cor93]). The operator delays and clock period can have significant impact, especially when only limited speedups are achievable. For cases when arbitrary reduction of the clock cycle critical path is achievable, their optimization is not crucial. They should not be ignored, however, since they can affect secondary design metrics such as area and power consumption.

The techniques for throughput optimization are characterized by quantifying the following information:

Effectiveness -- The potential of the technique to directly improve throughput;

Enabling Potential -- The potential of the technique to facilitate subsequent throughput reduction by enabling other techniques and enhancing their effectiveness;

Confidence metrics for predictions regarding effectiveness and enabling potential;

Side effects -- The effects on area and power.

No Feedback

Non-recursive subparts can be sped up arbitrarily by time-unfolding alone. Since there is no interaction between operations in successive iterations, the critical path remains unchanged after unfolding. The effective critical path for an unfolding factor of k is thus *initial critical path*/k.

For a given level of unfolding, non-recursive parts can be further sped up by pipelining. In general, pipelining can reduce the clock cycle critical path to exactly $Max(\lceil iterationbound \rceil, maxduration)$, where the iteration bound is the $\underset{c \in cycles}{Max}((computationtime(c))/(numberofdelays(c)))$, and *maxduration* is the duration of the longest operator. For computations in this class, the iteration bound is 0, and thus critical path is reduced to *maxduration*. This value stays constant regardless of the unfolding factor, k, and thus the effective critical path using both unfolding and pipelining is *maxduration*/k.

Linear and Feedback Linear

Linear subparts can be sped up arbitrarily using several techniques. The "maximally fast" technique [Pot92], which combines several algebraic and redundancy manipulation transformations, reduces the effective critical path to $\lceil \log_2(NumStates + 1) \rceil + 1$ /k, where *NumStates* is the number of states and k is the unfolding factor. This technique can also be used for feedback linear computations. For linear computations that are also time-invariant and single input, the technique of [Sri95a] is an alternate option that often results in lower area overhead.

Note that if an entire computation is linear (as compared to just a sub-part), the mentioned techniques can be applied directly, without first dividing the computation into SCCs. The divide-and-conquer technique is likely to result in lower area overhead, however, due to reduced interaction between the computation's states resulting from SCC isolation.

Non-Linear, Translatable to a Form with No Non-Linearities Inside the SCC

For this class of computations, where non-linearities can be moved outside of cycles, we use a combination of techniques for non-recursive and for linear computations. Non-linear operations such as the logarithm and inverse logarithm operations, lie outside of the SCC and can thus be optimized using techniques for non-recursive parts. The linear parts remaining within the SCC may be sped up using techniques for linear computations.

General Non-Linear

Our strategy for optimizing general non-linear computations is again based on the divide-and-conquer paradigm. The key idea is that while an SCC as a whole may be non-linear, many of its parts often belong to classes which are easily optimized. Optimization of these parts often greatly improves the critical path of the overall SCC. As explained in Section 4.1, the computation is divided into output and next state cones. Cones are optimized independently; the resulting critical path is determined by the largest cone critical path.

Since primary output cones are by definition a non-recursive function of all inputs and previous states, they can thus be optimized using techniques for non-recursive computations. Next state cones that are not general non-linear computations can be optimized using the above mentioned techniques for linear, feedback linear, and non-linear translatable computations. Optimization of general non-linear cones is the most challenging. A heuristic search is conducted to find an effective combination of techniques such as algebraic manipulations, redundancy manipulations, template matching, module selection, parameter selection (clock period and voltage). Computational complexity is reduced by restricting the search to optimization of a subset of the cone. The subset used is the ε-critical network, which consists of all operations and connections which lie on paths of lengths greater than (1-ε) times the cone critical path.

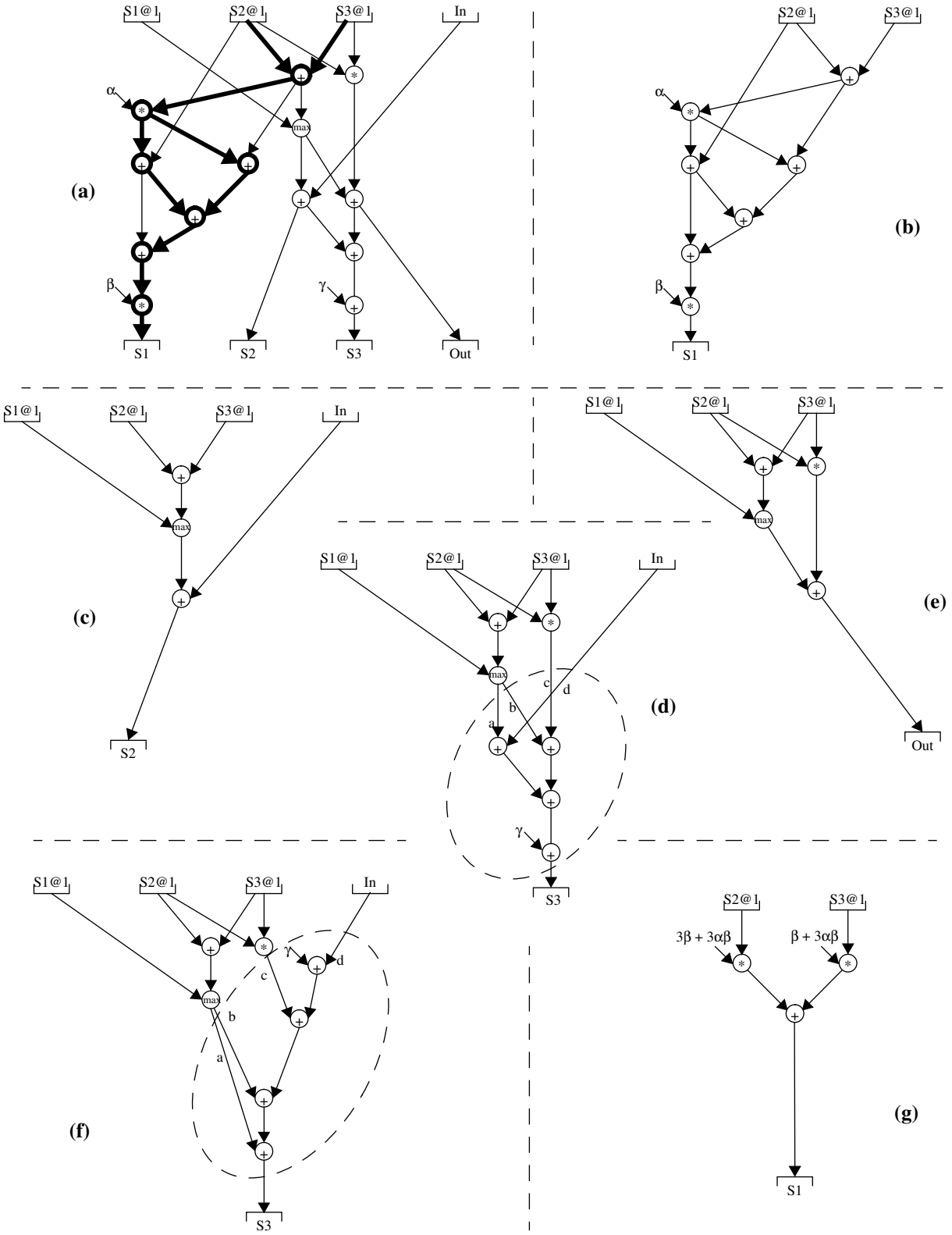

Fig. 6. a) General Non-linear SCC, b) S1 cone, c) S2 cone, d) S3 cone, e) output cone, f) optimized S3 cone, g) optimized S1 cone.

Consider the non-linear recursive example of Figure 6a, whose initial critical path is 6. The single output and the 4 next state cones are shown in Figures 6b-e. The S1 cone's critical path is 6 (Figure 6b), but can be reduced to 2 using the maximally fast technique for linear computations (Figure 6g). S2 is also linear, but its critical path is already at its minimum value of 3 (Figure 6c). The output cone's critical path of 3 (Figure 6e), can be reduced to 1 using pipelining if desired. Since it is not the critical path bottleneck, however, this is not necessary. S3, the general non-linear cone, is optimized by using associativity on the 4 additions which are circled in Figure 6d to give a critical path of 4 (Figure 6f). Overall the critical path is reduced from 6 to 4.

Arbitrary speed up is not known to be achievable for this class of computation. Kung's proof [Kun76] 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.

6.0 Experimental Results

Table 1 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 techniques result in practically realistic implementations, while the "maximally fast" implementation is prohibitively expensive.

Name	Initial Critical Path	Best Previous (no unfolding)	New Technique (no unfolding)	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
Vocoder	110	27	5	27	Arbitrary
ADPCM[b]	72	17	5	17	Arbitrary

Table 1: Experimental Results

a. NEC digital-to-analog converter
b. adaptive delta pulse code modulation

7.0 Summary

We have proposed a novel 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.

8.0 References

[Bra84] R.K. Brayton, G.D. Hachtel, C.T. MvMullen, A.L. Sangiovanni-Vincentelli, "Logic minimization algorithms for VLSI synthesis", Kluwer, Boston, MA, 1984.

[Cha92] A. Chandrakasan, M. Potkonjak, J. Rabaey, R.W. Brodersen, "Hyper-LP: Design system for power minimization using architectural transformations", *Int'l Conference on Computer-Aided Design*, pp. 300-303, 1992.

[Cor93] M. Corazao, M. Khalaf, L. Guerra, M. Potkonjak, J. Rabaey, "Instruction set mapping for performance optimization", *Int'l Conference on Computer-Aided Design*, pp. 518-521, 1993.

[Fet90] A. Fettweis, H. Meyr, L. Thiele, "Algorithm transformations for unlimited parallelism", *IEEE Int'l Symposium on Circuits and Systems*, pp. 1756-1759, New Orleans, 1990.

[Gib94] W.W. Gibbs, "Software's chronic crisis", *Scientific American*, pp. 86-95, September 1994.

[Hua94] S. Huang, J. Rabaey, "Maximizing the throughput of high performance DSP applications using behavioral transformations," *EDAC*, pp. 25-30, 1994.

[Kun76] H. T. Kung, "New algorithms and lower bounds for the parallel evaluation of certain rational expressions and recurrences," *JACM*, vol. 23, no. 2, pp. 252-261, 1976.

[Lee87] 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.

[Mas87] H. Massalin, "Superoptimizer: A look at the smallest program", *Int'l Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 122-126, 1987.

[McK65] W.M. McKeeman, "Peephole optimization", *Communications of the ACM*, Vol. 8, No. 7, pp. 443-444, 1965.

[Mes88] D. G. Messerschmitt, "Breaking the recursive bottleneck," *Performance Limits in Communication Theory and Practice*, J.K. Skwirzinsky (ed.), Kluwer Academic Publisher, Amsterdam, 1988.

[Not91] S. Note, W. Geurts, F. Catthoor, H. De Man, "Cathedral-III: Architectural-driven high level synthesis for high throughput DSP applications," *ACM/IEEE Design Automation Conference*, pp. 597-602, 1991.

[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 and 1117-1134, 1989.

[Pot92] M. Potkonjak, J. Rabaey, "Maximally fast and arbitrarily fast implementation of linear computations," *Int'l Conference on Computer-Aided Design*, pp. 304-308, 1992.

[Sri95a] M. Srivastava, M. Potkonjak, "Optimum and heuristic transformation techniques for simultaneous optimization of latency and throughput", *IEEE Trans. on Very Large Scale Integration Systems*, Vol. 3, No. 1, pp. 2-19, 1995.

[Sri95b] M. Srivastava, M. Potkonjak, "Energy efficient implementation of linear systems on programmable processors", *IEEE Workshop on VLSI Signal Processing VIII*, 1995.

[Tar72] R. E. Tarjan, "Depth first search and linear graph algorithms," *SIAM Journal on Computing*, Vol. 1, No. 2, pp. 146-160, 1972.

[Whi90] D. Whitfield, M.L. Soffa, "An approach to ordering optimizing transformations," *ACM Symposium on Principles and Practice of Parallel Programming*, pp. 137-147, 1990.

[Wol91] 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.