

AN INTEGRATED FRAMEWORK FOR OPTIMIZING TRANSFORMATIONS

Shan-Hsi Huang and Jan M. Rabaey

Dept. of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

Abstract - This paper proposes a framework aimed at the optimization of speed, area, or power consumption of custom ASIC DSP designs through algorithmic transformations. This framework systematically selects and orders transformations for optimization. The methodology behind the framework combines bottleneck analysis (why the transformations should be applied), transformation ordering (the order in which the transformations are applied), algorithm partitioning (which parts of an algorithm should be transformed), transformation analysis/selection (which transformations to apply), and transformation execution (how to apply the selected transformations). Assisted by this framework, designers can easily and quickly exploit a variety of optimizing transformations to explore the algorithmic design space to reach better designs.

1. INTRODUCTION

In the past few years, numerous researchers in high-level synthesis have successfully exploited algorithmic transformations in optimizing a design with respect to a variety of objectives, including speed, area, power, and memory. Most approaches consider only individual or small sets of transformations. Since the applicability of an individual transformation is often restricted, it is desirable to integrate a large number of transformations. While this enhances the effectiveness, it may, however, greatly increase the complexity of the optimization process.

Given a set of transformations, determining which transformations to apply, where to apply them, and in what order is a non-trivial task. A commonly used approach is to provide an interactive user interface which requires users to make all the decisions themselves. This approach has the disadvantage of being ad hoc and might not work in cases where the cost-function is complex, such as in area minimization. Another approach is to develop a dedicated algorithm (often with heuristics) which integrates all the transformations into one. This approach has little or no extensibility in terms of adding new transformations. As a large number of transformations exist, the above approaches are neither practical nor efficient.

While previous research efforts have been devoted to the exploration of new individual transformations (or new algorithms for the known transformations), it is now becoming more compelling to have a CAD environment that *integrates and systematically utilizes a large set of existing transformations, and has the flexibility for easy integration of new ones*. In this paper, we propose one such framework,

TAO (Transformation for Algorithm Optimization). This framework can help designers effectively exploit a variety of optimizing transformations to improve their designs.

To substantiate these concepts, we will use area optimization for custom DSP chip design as an example. The problem with area optimization is that the global impact of a single transformation is often ambiguous. Typically, the interplay between many transformations is needed to get a substantial area improvement. For functional units, [2] and [3] proposed two different approaches - improving the resource utilization and reducing operation count. The fundamental concepts for area optimization will become apparent in the course of this paper. The proposed framework is also extensible to other cost-functions, e.g. speed and power. In the next section we present the *TAO* framework. The effectiveness of the approach is demonstrated in Section 3. This paper concludes with a summary.

2. TRANSFORMATION FRAMEWORK

Figure 1 shows the structure of the *TAO* framework. The inputs are a designer's algorithm represented as a control/data flowgraph (*CDFG*), and design constraints such as time, area, or power. Within this framework, a set of prediction models (*P-models*) and a transformation library (*T-lib*) are predefined (Section 2.1 & Section 2.3, respectively).

The **bottleneck analyzer** uses these P-models to identify bottlenecks (Section 2.2). Designs implemented directly from their initial algorithm specifications often do not meet all specified constraints. The factors that violate the constraints are called the *bottlenecks* of the design. For example, if a design cannot meet the timing constraint (e.g., sample period for DSP applications), the execution time is the bottleneck; the resources dominating the area are the bottlenecks when area constraints are not met. Resources include functional units (multipliers, ALUs, etc.), registers, interconnect, and memory units. Similarly, the bottlenecks for power are those resources that consume significantly more power than others. The objective of the optimization process is then to apply those transformations that specifically address the identified bottlenecks.

To achieve this goal, the optimizer resorts to a predefined set of transformations. The transformations are characterized in Section 2.4 and ordered in Section 2.5. Ordered transformations are represented as a set of *T-graphs* which is the primary object through which the modules in the framework communicate. This representation ensures our framework extendable to add new transformations.

With the identified bottlenecks and the set of T-graphs, the **transformation manager**, performing *algorithm partitioning*, *transformation analysis*, and *transformation selection*, determines which transformations should be applied and where to apply transformations (Section 2.6). The selected transformations are represented as a refined T-graph. Finally, the **transformer module** applies these transformations in an order dictated by the T-graph onto the subgraphs (Section 2.7). The transformed CDFG is then sent back to the bottleneck analyzer for

evaluation. If nothing better can be achieved, the best solution obtained so far is returned to the user. Otherwise, a new bottleneck is identified for the next iteration.

The *TAO* framework supports *dynamic optimization flow*. Depending on identified bottlenecks, different transformations might be applied in different orders through the optimization process. This benefit cannot be realized by any dedicated algorithm which has built-in static optimization flow. In the rest of this section, we will discuss each module in more detail. To substantiate the concepts, we use area optimization as an example.

2.1 Prediction models (P-models)

Rabaey et al. [4] have shown that there exist strong correlations between the performance metrics of a design and a number of *structural properties* of the algorithm. These high-level properties can be used to derive the P-models. For area optimization, a set of layered P-models are derived to predict the area cost of resources such as functional units (FUs), registers, and interconnect busses. These P-models later will also be used to characterize transformations in Section 2.4. For simplicity, we use FUs as an example. The layered P-model for FUs contains three submodels (Figure 2), each of which reveals a different degree of information. The first layer is the simplest. It only considers the *count* information (number of operations). This submodel determines the absolute min-bound on the amount of resources. Since different types of operations usually have different costs, the second layer of the model considers both the number of operations as well as their weights. The corresponding sub-model is the *weighted sum*. A further refinement of the model is to account for data dependencies as well. The submodel at this layer turns out to be *the maximal height of the distribution graph* [5]. Similar models can

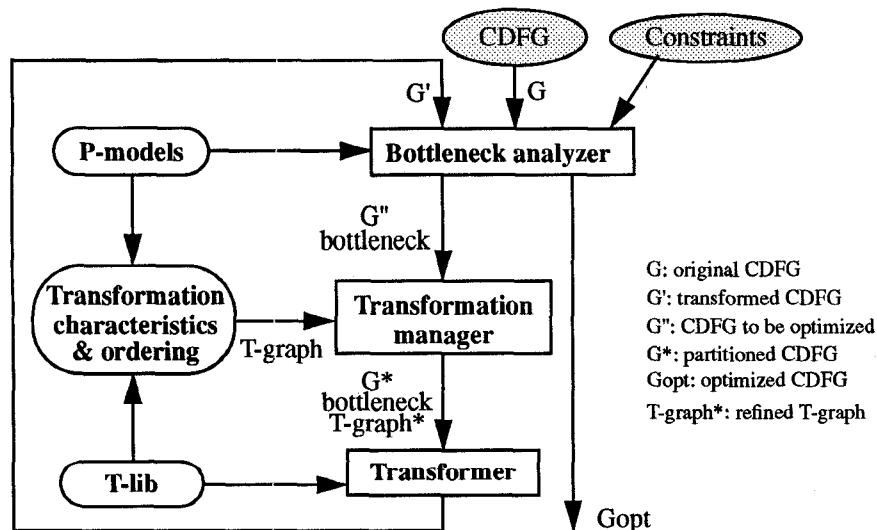


Figure 1: Structure of the *TAO* framework

also be derived for other resources like registers and interconnect busses. Given the distribution graphs, the bottleneck analyzer can quickly predict the area cost of all resources and identify which resource should be optimized.

2.2 Bottleneck analyzer

The bottleneck analyzer uses the P-models to identify bottlenecks. The identified bottlenecks will later be optimized using transformations. Depending on the design constraints, there could be a variety of bottlenecks, each of which may need different transformations. Optimizing all bottlenecks in a design simultaneously is not feasible. A *divide-and-conquer* strategy is used in our framework. The bottleneck analyzer picks a dominant bottleneck and defers others to later iterations. This allows the optimizer to eliminate the bottlenecks one by one.

For area reduction, the resources that dominate the chip area are the bottlenecks of interest. Among these, the bottleneck which has the highest *potential* for improvement is selected and others are deferred to later iterations. The potential can be identified by analyzing the distribution graphs. For example, the distance between the height of the distribution graph and the absolute min-bound roughly shows the potential improvement range.

2.3 Transformation library (T-lib)

The considered transformation set is defined in the T-lib. Because the scope of the design space is determined by the transformation set, it should be reasonably large and versatile. We currently consider the transformations which are most important in our application domain. For example, *algebraic transformations* and *retiming/pipelining* are of critical importance in DSP applications due to their computation-intensive nature and temporal property. The T-lib in our framework consists of algebraic transformations (*associativity, distributivity, reverse distributivity, commutativity, algebraic identity, algebraic inverse, constant folding, constant multiplication expansion*, and a few others), temporal transformations (*retiming, pipelining, time-loop unfolding*), loop transformations (*loop unrolling, loop fusion, loop distribution*) and some generic transformations (*common sub-expression replication/elimination, dead code elimination, loop invariant code motion*).

Layered P-model		Behavior of T's	Effects
count	$\Sigma(\#op_i)$	op reduction	reducing abs-min-bound
weighted sum	$\Sigma(\#op_i * weight_i)$	op conversion	reducing weighted abs-min-bound
height of distribution	$\Sigma(\#op_i * weight_i)$	op reordering	improving utilization

Figure 2: Layered P-models vs. Behavior of transformations (for FU's)

2.4 Transformation characteristics

With such a large set of transformations, the *characteristics* of transformations can help us understand their capabilities and interactions. This is important for transformation ordering and selection. In order to characterize a transformation, we determine which layer of a P-model the transformation has an effect on. In the case of FU's being the bottleneck, we can classify transformations into the following categories: (as shown in Figure 2)

(1) *operation reduction* [count] — This is the most often used technique for area reduction. Transformations in this category can reduce the number of operations, therefore the absolute min-bound. Possible transformations are common sub-expression elimination (CSE), reverse distributivity (also known as factoring), constant folding, algebraic identities, and loop invariant code motion.

(2) *operation conversion* (or *strength reduction*) [weighted sum] — This technique trades expensive operations for cheaper ones so as to reduce the overall weighted sum. This however may increase the total operation count. The weight of an operation is the module area given in the hardware library. A representative transformation in this category is constant multiplication expansion (CM).

(3) *operation reordering* [height of a distribution graph] — This class of transformations improves the utilization of hardware resources by changing the computational order of the operations and therefore their distribution. Possible transformations in this category are associativity, distributivity, retiming, pipelining, loop fusion/distribution and loop unrolling.

(4) *none of the above*: There are also transformations that cannot directly improve the area of FU's, for example, commutativity and common sub-expression replication (CSR). They might, however, enable the application of transformations of classes 1-3.

2.5 Transformation ordering

When a set of transformations are available, the order in which they are applied often affects their effectiveness [8-11]. According to the transformation characteristics described above, a simple bound on the potential improvement of each transformation can be obtained (Figure 2). Operation-reduction reduces the operation count, therefore the absolute min-bound. Operation conversion directly reduces the weighted sum, but the total operation count is bounded. Operation-reordering improves the resource utilization by reducing the height of a distribution graph, but the potential improvement is bounded by the (weighted) absolute min-bound. The transformations at the lower layer have a more direct impact on the area, but those at the upper layer may lead to a better solution. Transformations should therefore be ordered in a top-to-bottom fashion. At first, operation reduction is used to reduce the absolute min-bound. If possible, operation conversion follows to trade expensive operations for cheaper ones such that the weighted min-bound is reduced. This is the best possible solution that we can eventually achieve. Ultimately, operation reordering moves the final resource requirement as close as

possible to the absolute min-bound by improving the resource utilization.

We can further derive a (partial) ordering among individual transformations in each category. Those transformations that can directly address the designated goal are called *kernel* transformations. Transformations that can enable the applicability of a kernel transformation are called *enabling* transformations. Since a kernel transformation may also enable other kernels, we can derive the primary (*partial*) ordering among kernel transformations. The primary ordering determines the order in which kernel transformations are applied. As to enabling transformations, they will be invoked as demanded by the kernels (demand-driven).

In our framework, the ordering among transformations is represented as a *transformation graph (T-graph)*, where each node represents a transformation and a directed edge represents the dependency relation between input and output node (Figure 3). Because transformation ordering varies with different objectives (P-models or submodels), different T-graphs are provided for each of them. These T-graphs are predefined in a generic fashion, and will be refined by the transformation manager, according to the design context. The final T-graph is used to drive the transformer module. T-graphs are the primary object in our framework through which the modules communicate. This representation enables our framework a flexibility of adding new transformations later. We have derived the T-graphs for area optimization. Due to lack of space, the details of the T-graphs (and how to handle loops in a T-graph) are not elaborated here. A partial T-graph for operation reduction is shown in Figure 3. Constant folding, algebraic identity and the inverse, and CSE are kernel transformations. Commutativity, associativity, and retiming are enabling transformations. The edge from constant folding to CSE says that constant folding enables CSE.

2.6 Transformation manager

Given the identified bottleneck and T-graphs, the transformation manager determines *which* transformations should be applied and *where* to apply them. The transformation manager in our framework consists of three sub-modules: the algorithm partitioner, transformation analyzer, and transformation selector. The primary responsibility of the T-manager is to reduce the search space and thus

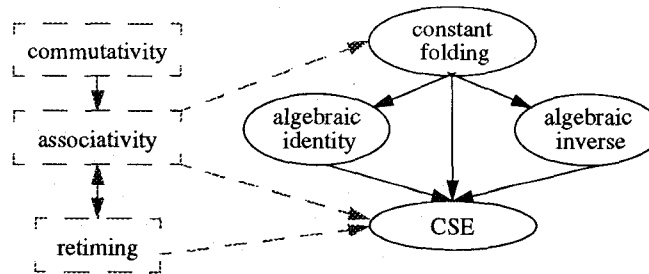


Figure 3: Partial T-graph for operation reduction

improve the efficiency.

(1) Algorithm partitioner:

Based on a given bottleneck, the partitioner extracts the trouble spots of a given CDFG. Transformations will be applied only to these subgraphs. The transformations which are not applicable in these subgraphs are omitted because they cannot improve the bottleneck. This reduces the transformation space.

(2) Transformation analyzer:

The transformation analyzer then predicts the *potentials* of transformations which will be used by the transformation selector. Since only kernel transformations directly affect the bottleneck, potential improvements from them are of a major concern. Rim [12] presented some models to predict performance of some loop transformations. For the *always-win* transformations, like constant folding or dead code elimination, the associated evaluation routines are trivial. We are currently working on models for other transformations. Due to the lack of space, we cannot enumerate the obtained models here. One example is constant multiplication expansion. Its effectiveness can be evaluated by considering the numbers of constant multiplications and variable multiplications, as well as the associated coefficients.

(3) Transformation selector:

Based on the predicted performance of transformations, a set of kernel transformations with high potentials are selected for final execution. Since enabling transformations are used to enable kernel transformations, they will be applied in a demand-driven fashion (to avoid the redundant enablers). There is no need to preselect them. This selection results in a refined T-graph which is a subset of the original T-graph.

2.7 Transformer

After processing by the transformation manager, the bottleneck together with the partitioned CDFG and the refined T-graph are passed to the transformer module. The transformer applies selected transformations to the subgraphs. These transformations are applied in the order dictated by the T-graph. "*Mutually enabled*" kernel transformations are applied together. The cost function is provided by the associated P-model.

The transformer in our framework exploits two classes of techniques. The first, which our framework gives preferential treatment to, involves using dedicated approaches. If a dedicated technique matching the cost function and covering the desired transformations exists in the T-lib, it is applied. The dedicated approaches have a higher priority due to their efficiency and prowess. On the other hand, if none exists, generic local-move-based optimization techniques such as simulated annealing and the steepest descent method are then used.

3. EXPERIMENTAL RESULT

Experimentation on a number of DSP filter examples has demonstrated an average of 30-50% area reduction. To illustrate the key ideas behind the *TAO* framework, consider the 2nd order Volterra filter shown in Figure 4(a). We assume both multiplications and additions are unit-cycle operations, but the cost (area) of a multiplier is significantly larger than that of an adder. We also assume that all the constant coefficients are unique. The objective is to minimize area of functional units without violating the given throughput constraint.

The critical path of the original structure is 12 clock cycles. Given a sample period of 12 clock cycles, direct implementation requires at least two multipliers and one adders (absolute min-bound). Since multipliers are expensive, a common approach is to trade multipliers for cheaper adders/shifters by using constant multiplication expansion (CM). The expansion, however, will violate the timing constraint, and thus speed optimization techniques would need to be invoked. The resulting implementation requires at least one multiplier for the variable multiplications. Extra adders and shifters are also needed due to the expansion. It is therefore not clear that applying CM is desirable, especially since no other transformations have been considered yet. With our framework, we can systematically resolve this area optimization problem.

From the P-models discussed in Section 2.1, the bottleneck analyzer identifies multipliers as the bottleneck resource in area. According to the layered P-models, the optimization process iterates through 3 rounds. The first round is operation reduction. The only applicable kernel transformation, reverse distributivity, enabled by associativity and commutativity, reduces the number of multiplications from 17 to 11 (Figure 4b). The absolute min-bound of multipliers is reduced from two to one. The next round is operation conversion, which trades multipliers for adders/shifters. Due to the existence of variable multiplications, at least one multiplier is needed. CM cannot reduce it any further and this round is skipped. The third round is utilization improvement. The distribution graph predicts two multipliers might be needed, but the absolute min-bound of multipliers is only one. Transformations to improve the resource utilization are considered. Applicable

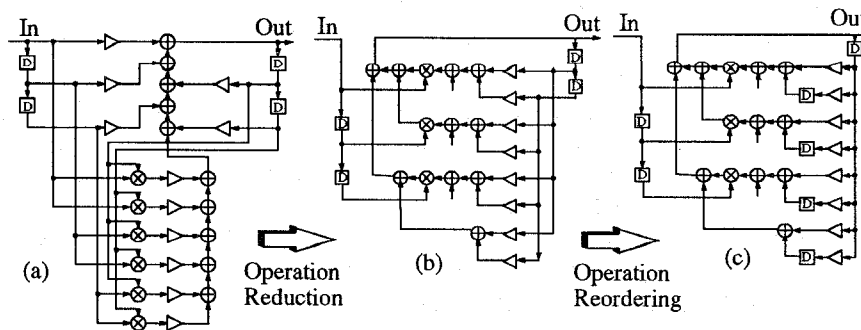


Figure 4: Structures of a 2nd order Volterra filter

kernel transformations are retiming, associativity, and distributivity. Among them, distributivity is undesirable because it may increase the absolute min-bound from 1 to 2. The final structure shown in Figure 4(c) only needs 1 multiplier and 1 adder.

This optimization process saves 1 multiplier without adding extra units, which results in about 50% reduction of the datapath area. This result is superior to any of the existing approaches. Furthermore, speed optimization techniques were not needed. Throughout this process, only capable and applicable transformations are considered at each stage and only the appropriate transformations are applied. This framework systematically determines what to optimize (with the *layered P-models*) and which transformations to use (based on the *characteristics of transformations*), and quickly reaches a better solution. One thing worth notice is that CM is not needed in this case at all, although constant multiplications still exist in the final structure. This example also shows the effectiveness of transformation ordering. If we apply CM first, we are hardly able to reach the best solution. Applying transformations in an appropriate order not only can reach a good solution faster, but also may achieve a better one.

4. FUTURE WORK

Within *TAO*, a framework for area optimization has been developed. The transformation framework for speed optimization that we developed in the *HYPER* system is being integrated into the *TAO* framework [11]. Power optimization will be considered next. We are currently enhancing the transformation manager to make the *TAO* framework more effective and efficient. In addition, our main attention so far has been focused on the optimization of datapath resources such as FUs, registers, and interconnect. Since many real-time DSP applications are memory-intensive, memory-related issues will also be considered. This extension will require the introduction of a larger set of loop transformations.

5. CONCLUSION

In this paper, we have described the *TAO* framework, an integrated framework for optimizing transformations. This framework can *systematically* choose effective transformations to apply at the right place and in an appropriate order. We have proposed a set of layered prediction models for area, and characterized transformations into four categories (for FU's). The proposed framework is *extensible*, in that new cost functions and new transformations can be easily integrated, and supports *dynamic optimization flow*.

6. ACKNOWLEDGEMENTS

The authors would like to thank M. Potkonjak for valuable discussion, and Semiconductor Research Cooperation for sponsoring this project (96-DC-324).

7. REFERENCES

- [1] S.-H. Huang, J.M. Rabaey, "A methodology to apply optimizing transformations," UCB/ERL memorandum, M95/32, Feb. 1995.
- [2] M. Potkonjak, J. Rabaey: "Optimizing the resource utilization using transformations," *Proc. of ICCAD*, Nov 1991.
- [3] M. Janssen, F. Catthoor, H. D. Man, "A specification invariant technique for operation cost minimization in flowgraphs," *Int'l Sym. High Level Synthesis*, pp. 146-151, 1994.
- [4] J.M. Rabaey, L.M. Guerra, "Exploring the architecture and algorithmic space for signal processing applications," *ICVC - 3rd Int'l. Conference on VLSI and CAD 1993 (ICVC '93)*, pp. 315-319. Nov. 1993.
- [5] P.G. Paulin, J.P. Knight, "Force-directed scheduling for the behavioral synthesis of ASIC," *IEEE Trans. on CAD*, vol. 8, no. 6, pp. 661-679, June 1989.
- [6] L. Guerra, M. Potkonjak, J.M. Rabaey, "System-level design guidance using algorithm properties," *VLSI Signal Processing VII*, IEEE Press, NY, pp. 73-82, 1994.
- [7] D. Whitfield, M.L. Soffa, "Investigating properties of code transformations," *Proc. of Int'l Conference on Parallel Processing*, pp. II-156-160, Aug 1993.
- [8] D. Whitfield, M.L. Soffa, "An approach to ordering optimizing transformations," *2nd ACM Symposium on Principles and Practice of Parallel Programming*, pp. 137-147, March 1990.
- [9] D. Whitfield, M.L. Soffa, "Automatic generation of global optimizers," *Proc. of the ACM SIGPLAN '91 Conf. on Programming Language Design and Implementation*, pp. 120-129, June 26-28, 1991.
- [10] M. Potkonjak, J. Rabaey: "Maximally fast and arbitrarily fast implementation of linear computations," *Proc. of ICCAD*, pp. 304-308, Nov. 1992
- [11] S.-H. Huang, J.M. Rabaey, "Maximizing the Throughput of High Performance DSP Applications Using Behavioral Transformations," *Proc. of EDAC-ETC-EUROASIC 94*, pp. 25-30, March 1994.
- [12] M. Rim, R. Jain, "Estimating performance characteristics of loop transformations," *Int'l Symp. on Circuits and Systems*, pp. 249-252, June 1994.
- [13] D.J. Kolson, A. Nicolau, N. Dutt, "Integrating program transformations in the memory-based synthesis of image and video algorithms," *Proc. of ICCAD*, pp. 27-34, Nov 1994.
- [14] R. Karri, A. Orailoglu, "Transformation-based register optimization in high-level synthesis," *Conf. Record of Asilomar Conf. on Signals, Systems and Computers*, pp. 894-898, 1992.