

Potential-Driven Statistical Ordering of Transformations

Inki Hong, Darko Kirovski, and Miodrag Potkonjak

UCLA Computer Science Department, Los Angeles, CA 90095-1596 USA

Abstract

Successive, well organized application of transformations has been widely recognized as an exceptionally effective, but complex and difficult CAD task. We introduce a new potential-driven statistical approach for ordering transformations. Two new synthesis ideas are the backbone of the approach. The first idea is to quantify the characteristics of all transformations and the relationship between them based on their *potential* to reorganize a computation such that the complexity of the corresponding implementation is reduced. The second one is based on the observation that transformations may disable each other not only because they prevent the application of the other transformation, but also because both transformations target the same *potential* of the computation. These two observations drastically reduce the search space to find efficient and effective scripts for ordering transformations. A key algorithmic novelty is that both conceptual and optimization insights as well as all optimization algorithms are automatically derived by organized experimentation and statistical methods. On a large set of diverse real-life examples improvements in throughput, area, and power by large factors have been obtained. Both qualitative and quantitative statistical analysis indicate effectiveness, high robustness, and consistency of the new approach for ordering transformations.

1 Introduction

1.1 Motivation

Transformations are alternations in the structure of a computation such that the initial input/output specification is completely maintained [1]. Transformations have been recognized as the high-level synthesis and compilation step with the highest impact on design metrics. It has been demonstrated that exceptionally high improvements in all design metrics including area, throughput and latency, power, transient and permanent fault-tolerance overhead, and testability are achievable on both ASIC and programmable platforms.

However, transformations have also acquired reputation as one of the most difficult behavioral synthesis tasks. Transformations are notorious for their ability to unpredictably alter the numerical properties of a design and therefore its required word-length. Furthermore, the implementation and maintenance of a large number

¹This research is supported in part by California MICRO grant 96-182 and Escalade.

Design Automation Conference (®)

Copyright © 1997 by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept, ACM Inc., fax +1 (212) 869-0481, or permissions@acm.org.

0-89791-847-9/97/0006/\$3.50

DAC 97 - 06/97 Anaheim, CA, USA

of transformations in compiler and behavioral synthesis environments is a formidable software task. Also, the accurate prediction of effects of transformations has been rarely addressed and it is widely considered infeasible. Finally, the objection which is most often raised and quoted with respect to the application of transformations is that their full potential can be explored only with proper orders of transformations. However, deriving such orders is mainly an art practiced by experienced developers of compiler and CAD tools, which often result in disappointing outcomes.

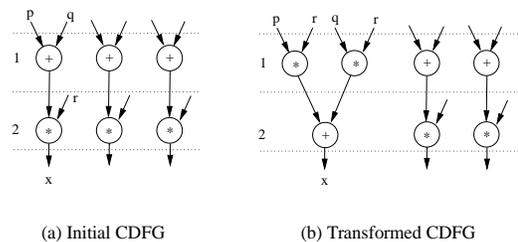


Figure 1: A motivational example for transformations in behavioral synthesis.

In this paper we address the last issue: the development of fully automatic, fast, and effective ordering of transformations for a variety of design metrics, such as throughput, area, and power. We have three major objectives in this paper. The first goal is to gain new insights on the transformations-based high-level synthesis optimization process. The second goal is to develop efficient and effective schemes for ordering transformations with respect to a variety of design metrics under a variety of constraints. The final goal is to demonstrate conceptual simplicity, effectiveness, and robustness of statistical methods for the development of optimization algorithms for complex CAD tasks.

1.2 Motivational Examples

We introduce the key ideas of the new approach for ordering transformations using two small, but meaningful examples. We first consider the example in Figure 1. Figure 1(a) shows the control-data flow-graph (CDFG) of a computation which consists of three additions and three multiplications. We assume that each operation takes one clock cycle and the available time is two cycles. Note that since the length of the critical path is also two cycles, all operations are on the critical path. Obviously, regardless of the used scheduling algorithm, the final implementation requires at least three multipliers and three adders. This design has relatively low, 50%, resource utilization for execution units.

An easy and effective way to improve this design is to apply transformations. Figure 1(b) shows the same CDFG after the application of distributivity on the isolated component which computes output x . Again, all operations are on the critical path. Therefore, the only feasible schedule is one shown in Figure 1(b). It is easy to see that only two multipliers and two adders are required.

It is interesting and important to analyze why the CDFG in Figure 1(b) is more amenable for the implementation with high resource utilization. The transformed design has one more operation than the initial design. So, one may expect that the transformed CDFG is inferior. However, this design has more evenly distributed operations of the same type along the time axis.

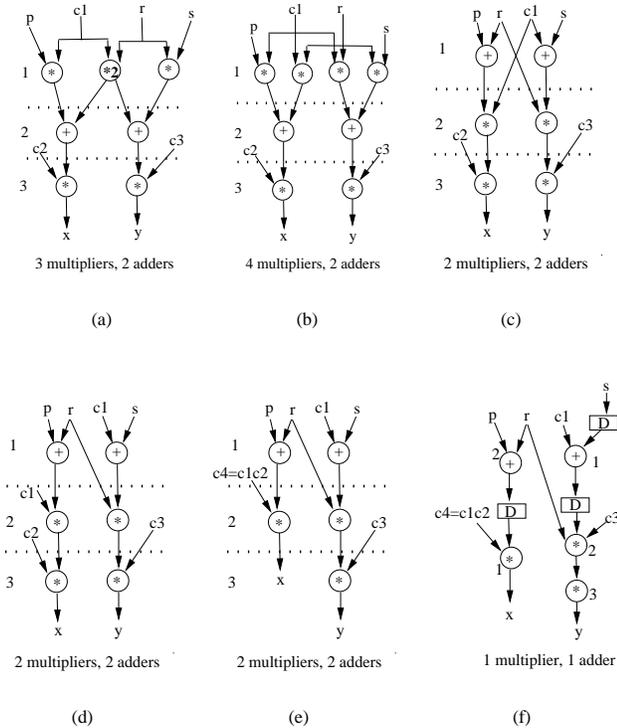


Figure 2: A motivational example for ordering transformations: Importance of considering enabling and disabling effects. All data denoted by c_i are constants.

How can one generalize the just mentioned observations so that they can be incorporated in a synthesis program? The essence of the approach can be summarized in the following way. The initial CDFG has a high *potential* to be improved with respect to the area of the final implementation because it has data precedence constraints which force the low resource utilization of execution units. This can be observed by comparing the current solution (or bounds on the quality of the current solution) with the solution derived with the assumption that all units can be used in all control steps. In this example the bounds of the initial solution are three multipliers and three adders, while the bounds for fully utilized units are two multipliers and two adders. To achieve improvement one needs such transformation that will more evenly distribute operations of the same type along the time axis, i.e., alter data dependencies in such a way that some of multiplications can be moved in the second control step, and some of additions in the first control step. It is easy to deduct that distributivity is such transformation. As shown, distributivity is indeed effective.

Another important observation is that when all execution units are well utilized, no further improvement can be achieved using transformations which only alter data and timing dependencies. This is because all *potential* along the dimension has already been realized (as indicated by absolute bounds on the number of execution units assuming 100% utilization). Thus, for further improvement, transformations which address some other *potential* of the

computation are required. We conclude the discussion of our first motivational example by noting that the substitution of constant multiplications with shifts and additions is such transformation.

The second motivational example demonstrates the importance of simultaneous consideration of the effects of more than one transformation. It also illustrates the conceptual complexity of transformations ordering and provides initial hints about how this task can be simplified while preserving the power of transformations.

The design goal is area optimization under throughput constraint. Figure 2(a) shows the initial CDFG which has five multiplications and two additions. The available time is three clock cycles. In the same fashion as in the first example, all operations are on the critical path. Therefore the only feasible schedule is shown in Figure 2(a). It is easy to see that three multipliers and two adders are required. The resource utilization is low for an ASIC design, for multipliers 55% and for adders 33%.

It is often required to apply one or more enabling transformations in order to eventually apply a transformation which will improve a design. Following this direction, we observe that our goal is to move some additions from the second control step to the first one and to move some multiplications from the first control step to the second one. It is easy to verify that distributivity can accomplish this task. However, the application of distributivity is disabled, because the result of multiplication *2 is used in two places. We first replicate multiplication *2 as shown in Figure 2(b). If we stop the transformation process at this stage, it is easy to see that the corresponding implementation requires 4 multipliers and 2 adders, even more hardware than in the initial design. However, we can now apply distributivity twice, as shown in Figures 2(c). We need only 2 multipliers and 2 adders. More importantly, we can continue the optimization process, by first replicating constant $c1$ and then applying constant propagation, as shown in Figures 2(d) and 2(e). Finally we can introduce a pipeline stage (if the computation is in a loop, retiming is performed), and obtain the functionality equivalent computation shown in Figure 2(f). The numbers next to the nodes - operations, indicate the clock cycle in which a particular operation is scheduled. We need only 1 multiplier and 1 adder.

This small example illustrates not only high effectiveness, but also an exceptional richness of degrees of freedom during optimization using sequences of transformations. Nevertheless, as it is shown in the rest of the paper, once the obtained insights are coupled with statistical methods, there is a conceptually simple, effective and efficient solution for ordering of transformations.

1.3 Potential-Driven Statistical Approach for Transformations Ordering - Summary of Key Ideas

We use a macroscopic approach for transformations ordering. In our approach, we use readily available transformation software routines as the smallest atomic optimization entities. All that is required is another layer of an optimization mechanism which invokes individual transformations and realizes the optimization potentials of the targeted computation. The global search is conducted so that the full potential of a given computation is explored along all relevant degrees of freedom.

The methodology behind the new approach can be summarized in the following way. First, we analyze how one can make a computation more amenable for high quality implementation for a given set of design goals and constraints. Using this information, we select transformations which are best suited for this task or which enable transformations which are well suited for the optimization goal. Next, we analyze the selected transformations and classify them with respect to the degrees of freedom of computational potentials it can address. All selected transformations are theoretically analyzed with respect to their mutual enabling impact.

All transformations are classified in k classes. Transformations which belong to class i , enable all transformations which belong to class j , $i \leq j \leq k$. The exact order for the application of transformations for the optimization of a degree of freedom is derived experimentally. All transformations are also classified into a number of groups according to which optimization degrees of freedom they address. At most three transformations which target the same potential are included in any of the experimental scripts.

An exhaustive set of all possible scripts which satisfy enabling and potential criteria is applied on a set of learning examples. We terminate the evaluation of longer sequences of transformations, when it is recorded that the longer sequences do not perform better on more than 5% of the examples or when the implementation studies (scheduling and hardware mapping) show that further optimization along this degree of freedom does not have a potential for improvement in comparison with the shorter sequence which uses the same order of transformations. Usually, less than hundred scripts are examined, and the whole experimentation for a given design metric takes about one week. The experimental data generation and recording are fully automatic.

The performance of each script is recorded and best-performing scripts are manually combined into a single script. The selected script is statistically validated on a set of testing examples. The technical details of the selection process are presented in Section 4.

The rest of the paper is organized in the following way. In the next section we survey the related work. Next, after summarizing relevant assumptions and targets, we introduce a notion of potential for both computations and transformations. The backbone of the paper is presented in Section 4 where the connection between potentials of computations and potentials of transformations is developed. Section 5 describes obtained scripts by the new method. Finally, the effectiveness and efficiency of the scripts are validated and analyzed on a large set of examples and conclusion and directions for future work are outlined.

2 Related Work

The approaches for transformation ordering can be classified in 7 groups: peephole optimization [9], manually-derived static scripts [2], exhaustive search-based “generate and test” methods [8], algebraic ordering of linear loop-control flow transformations [13], probabilistic search techniques [3], bottleneck removal methods [4], and microscopic and special-domain enabling-effect based techniques [11]. This paper addresses several design metrics, establishes new insights in enabling-disabling effects with respect to the design metrics, develops a method for quantifying these effects, and provides a technique for automatic script development and validation.

3 Preliminaries

3.1 Computational and Hardware Models

Our methodology assumes that a synthesis or compilation system provides the complete implementation of transformations. The software platform on which we tested our ideas was mainly the Hyper high level synthesis system [12] from University of California, Berkeley. The selection was mainly due to the fact that the Hyper system is well suited for our goals: it uses a number of modular transformation subroutines and provides both complete implementation and means for measurement and/or prediction of a number of design metrics, such as area and power. Therefore, we use the synchronous data flow model of computations and the dedicated register file model [12]. To eliminate coupling effects between transformations and scheduling, we also use the BETS scheduler in the

mode that only area optimization is considered [5].

3.2 Transformations

We consider the following transformations: direct and inverse associativity, direct and inverse distributivity, inverse element law, common subexpression replication, time loop unfolding by an arbitrary factor k , The Leiserson-Saxe retiming for critical path, pipelining for critical path with k pipeline stages, maximal pipelining for critical path, and substitution of constant multiplications with shifts and additions/subtractions.

The transformations are organized in the following optimization software routines:

1. associativity for critical path (ACP)
2. associativity and common subexpression replication for critical path (ACSCP)
3. associativity for iteration bound (AI)
4. associativity and common subexpression replication for iteration bound (ACSI)
5. retiming for critical path (R)
6. pipelining with k -pipeline stages for critical path (P)
7. loop unfolding by a factor k (U)
8. constant multiplication substitution with additions/subtractions and shift (CM)

All transformations subroutines which use associativity also use integrated inverse element law transformation [11, 12].

The initial selection of transformations was mainly influenced by the availability of robust software implementations capable of handling numerous real-life designs. The final selection was mainly dictated by their eventual effectiveness and run-time efficiency as well as their power to fully realize transformation potentials of almost all the testing examples. One of the major points of the work presented in this paper is that the limited subset of transformations is most often sufficient to realize all available potentials of most of the designs. This leads to a conclusion that the transformations ordering is much simpler tasks than it was believed previously and that more involved and complex methods for transformations ordering are rarely required.

3.3 Design Metrics

We applied the new approach for ordering transformations for a number of design goals. In this paper, due to the space limitation, we restrict our attention to throughput, area, and power optimization. Development of transformations ordering for other design metrics is strategically straightforward with varied difficulties.

We consider the following three, probably the most popular and most widely used, design scenarios:

- Throughput optimization with no imposed constraints;
- Area optimization under throughput constraint;
- Power optimization under throughput constraint.

The metrics are measured in the following way. The critical path is directly calculated using the dynamic programming-based subroutine. The cycle length time is estimated using the cycle time estimation program of the Hyper system [12] which takes into account both gate and interconnect delays. The area is estimated after

hardware mapping using the Hyper system. This estimator has been statistically validated to produce results within less than 20% of error in more than 95% of the cases. Finally, the power is estimated using the Hyper-LP [3] estimation tools which was also statistically validated to be within 20% of the real power consumption in more than 95% of the instances.

4 Development of Transformation Scripts

4.1 Potentials of Computations

Potentials for improvement of a computation using transformations can be defined as a combination of the degrees of freedom along which the computation can be altered so that the final implementation is optimized. The *Potentials* of computations are closely related to the targeted design goals and constraints.

The analysis indicates the following potentials for the selected design scenarios:

- Throughput: the length of the critical path and the cycle time.
- Area: the number of cycles, the hardware cost of execution units available for the implementation of operations of a particular type, the number of operations of a particular type, the uniformity of distribution.
- Power: the cycle time, the length of the critical path, the number and type of operations, the hardware cost of execution units available for implementation of operations of a particular type, the number of operations of a particular type, the uniformity of distribution.

The intuition behind the outlined degrees of freedom is the following. The throughput optimization is the best defined scenario in the sense that it is feasible to accurately predict the cycle time and to exactly obtain information about the required number of cycles. Obviously, one wants to optimize both. For area optimization the goal is to avoid hardware expensive operations, such as multiplications and to achieve uniformly distributed operations and data transfers, and a large number of control cycles. A large number of control steps for a given throughput corresponds to a short cycle time. Finally, during power optimization the goal is to minimize the number of switching events through the minimization of the number of operations and data transfers and through minimizing the length of interconnects as well as to minimize the critical path, so that the throughput-voltage trade-off can be explored [3]. It has been shown that there is a very strong correlation between the area and the length of the interconnects. Therefore, power minimization can be seen as simultaneous throughput and area optimization, with additional degree of freedom for the minimization of the number of operations. Since the voltage reduction is most effective for power optimization, it is the primary goal to be addressed.

4.2 Transformations Potentials

The transformations potentials are classified in the following classes. Note that a transformation can belong to more than one class.

1. Transformations for relaxation or elimination of timing and data precedence constraints: Essentially this group of transformations equalizes the distribution of operations of a particular type along time axis. Among the transformations considered the following are in this group: associativity for both iteration bound reduction and critical path reduction with and without common subexpression replication, retiming, pipelining, and loop unfolding.

2. Transformations which substitute expensive operations with less expensive ones: Constant multiplication substitution with shifts and additions is the most effective transformation from this class. Theoretically, the inverse element law can substitute all subtractions with slightly less expensive additions, but we did not find any single example that this was the case.
3. Transformations for control cycle time reduction: Constant multiplication substitution with shifts and additions reduces the clock cycle time in the cases that all multiplications are by constants, by eliminating a need to accommodate multiplications within a cycle.
4. Enabling Transformations: Finally, in some cases transformations do not have optimization power, but can enable other transformations. One such transformation is common subexpression replication. We also used loop unfolding almost exclusively for this purpose.

4.3 Matching Transformations with Computations and Design Metrics

The development of optimization scripts is done in the following way. First all selected transformations are characterized according to their optimization potential and enabling effectiveness. This is done using exhaustive manual enumeration of all small computational structures. This step is explained at several references for a variety of transformations (for example, see [11]). Figure 3 summarizes this information for the selected set of transformations.

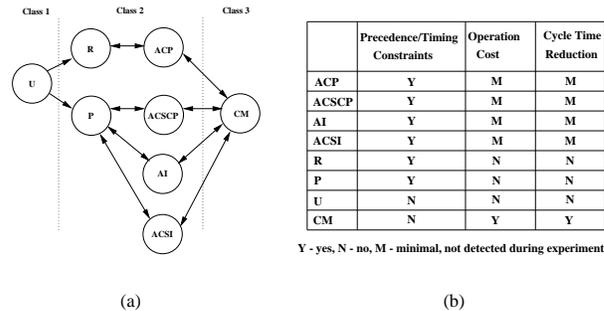


Figure 3: Optimization potential and enabling effectiveness of the transformations considered.

Once the information about optimization potential and enabling effectiveness of the transformations is obtained, a set of scripts is generated using the procedure presented in Section 1.3. The results are recorded and statistically processed.

Our goal is to select the smallest number of scripts which cover all best results for a given metric. This problem can be solved directly in an optimal fashion by including all scripts which are winners for at least one test example. Unfortunately, this results in including too many scripts. The second option is to select the smallest number of scripts, so that either the best or the second best script is selected for each example. This combinatorial problem can be solved in polynomial using the standard matching algorithms [10]. If the goal is to select as few scripts as possible, when it is sufficient to include for each example at least one script which provides at least k th-best solution, the problem is NP-complete [10]. In this case, we used simulated annealing [7] to select the smallest set of scripts. We set $k = 3$ and used a geometric cooling schedule and standard stopping criteria [7]. The experimental results presented in the next section are obtained using this approach.

5 Potential-Driven Statistical Scripts

The final scripts for the throughput, area, and power optimization are obtained by manually combining the selected winner scripts for a given metric, and are given by the following pseudocodes:

```

Throughput_Optimization() {
  k = 0;
  while (there is throughput improvement) {
    Unfold k-times starting from the initial specification;
    Associativity_for_Iteration_Bound;
    Maximum_Pipelining;
    k ++;
  }
}

Area_Optimization() {
  while (there is throughput improvement) {
    Constant_Multiplication_Substitution;
    Associativity_for_Iteration_Bound;
    while (there is area reduction) Pipeline_by_adding_1_stage;
  }
}

Power_Optimization() {
  k = -1;
  while (there is power improvement) {
    k ++;
    Unfold k-times starting from the initial specification;
    Constant_Multiplication_Substitution;
    Associativity_for_Iteration_Bound;
    while (there is power reduction) {
      Pipeline_by_adding_1_stage;
      Reduce_voltage_using_binary_search_to_find_the_minimal_power_solution;
    }
  }
}

```

Several important observations can be made about the developed scripts. Several very popular transformations did not show enough potentials to be included in any of the three methods for transformation ordering. In particular retiming, associativity, and common subexpression replication were not very effective. Only associativity for iteration bound was used, and it has relatively minor impact on most of the test examples. The explanation is that all of them mainly target data dependencies and timing constraints, which are also targeted by pipelining. However, pipelining is usually sufficient to realize all potentials along this dimension. Also, unfolding by larger factors was very rarely beneficial and often resulted in long run-times. The maximal unfolding factor was only 3. Maximal pipelining is a very powerful transformation, but often has negative impacts on both area and power. So, it was often important to add one pipeline stage at a time and evaluate the achieved impact. Backtracking to the previous solution when the impact was counterproductive was beneficial. Associativity should be tried before pipelining for realization of the optimization goals, since pipelining often disables the effectiveness of associativity. Finally, as the optimization goal is more complex, the more complex and longer sequences of transformations are required.

6 Experimental Results

In this section, we present the experimental results obtained using the potential-based transformational scripts on the following set of

benchmark designs. We used 55 small and 10 large (systems) designs. Typical examples include a variety of one and two dimensional FIR, IIR, nonlinear, and adaptive filters, various linear transforms (e.g. FFT and DCT), linear and nonlinear controllers, error-correction codes and systems such as modems, echo-cancellers, digital-to-analog converters, video decoders, and ghost cancellers. The specifications of designs were written by several authors. They applied different levels of manual optimization. All designs, except five large designs are public domain programs which can be electronically obtained from University of California, Berkeley.

Table 1 gives the statistics for throughput improvements using the potential-based script. During this experimentation we excluded the examples without feedback paths, although our script produced exceptionally high improvements for them. It is interesting to note that the implementation area grows at slower pace than the throughput improvement for more than 80% of examples. The median and average area increases for designs after the application of the throughput transformation script were by factors of 1.37 and 2.02 respectively. It is interesting to note that no single transformations were able to improve the critical path by a median factor large than 2. Two most efficient individual transformations were maximal functional pipelining and associativity respectively. No improvement was registered in only 2 examples. The reason in both cases was apparently extensive manual optimization applied by the designer on the initial specifications.

Minimal	Median	Average	Maximal
1.00	2.00	3.99	18.00

Table 1: Throughput improvement factors.

Available Time	Minimal	Median	Average	Maximal
ICP	1.00	3.41	4.93	15.78
1.5*ICP	1.00	2.36	3.22	11.22
5*ICP	1.00	1.67	2.37	9.01

Table 2: Area improvement factors. ICP - initial critical path length

The statistics for area improvements under initial throughput constraints are given in Table 2. We tested the script for area optimization using three different levels of scheduling difficulty scenarios which correspond to three different ratios of the available time and the critical path for the initial computation. The first interesting observation is that improvements are significantly higher when the initial available time is closer to the critical path. Majority of the results in high level synthesis literature belong to this category. Again, no single transformation was able to achieve individually more than half of the effectiveness of the script. The two most effective transformations were constant multiplication conversion (CM) and pipelining. In a number of design associativity for critical path optimization was also very useful. It is also interesting to notice that the CM transformation was crucial for area reduction when the available time was much larger than the critical path. Pipelining was instrumental to improve the design with the strict timing constraints, and was mainly counterproductive when the available time was much larger than the critical path.

Table 3 shows the power reduction obtained using our new ordering of transformations. Table 4 shows a subset of design examples that the least improvement was observed. The first and most important observation is that the improvements for this metric were by far most impressive. Even the application of individual transformations often resulted in high improvements, although no single

Minimal	Median	Average	Maximal
1.34	15.1	33.7	207

Table 3: Power improvement factors.

Design Name	IP	FP	IR
5th order wave digital filter	55.99	41.83	1.34
6 IIR - Gray Markel	153.06	52.60	2.91
6 IIR - Parallel	17.04	5.22	3.26
8 IIR - Avenhaus	39.33	4.81	8.18
large cascade filter	34.78	21.48	1.62
5 point convolution	19.54	7.38	2.65
7 point convolution	5.11	0.47	10.87
NEC DA converter	130.15	22.39	5.81
GE linear controller - 5 states	120.24	65.58	1.83
modem filter	100.33	53.30	1.88
Honda linear controller	172.35	16.01	10.77
GE linear controller - 3 states	407.45	61.32	6.64
adaptive modem	61.05	11.14	5.48
large modem	60.38	6.31	9.57
Volterra filter	14.36	4.71	3.05
Large Volterra filter	79.94	43.26	1.85
7th order wave digital filter	14.52	4.86	2.99
9th order wave digital filter	17.91	6.34	2.82

Table 4: A subset of benchmark designs where the approach yielded the least amount of power reduction. IP - Initial power in nJ/sample. FP - Final power. IR - Improvement ratio.

transformation had an average improvement higher than by a factor of 10. The most effective transformations were CM and pipelining.

The results for throughput, area, and power improvements obtained using the new transformations ordering approaches are significantly, approximately by a factor of 2, higher than those achieved by the corresponding default Hyper scripts for the optimization of the same design objectives. The typical run times for throughput, area, and power scripts were respectively 0.5, 3, and 10 minutes on a SUN Sparcstation 5. All results have been statistically validated using the bootstrap methods [6].

Our experiments also showed that the random ordering of transformations has a very low benefit. For example, the average improvements for throughput, three area optimization scenarios and power using randomly generated scripts were by factors 1.49, 1.74, 1.42, 1.27 and 2.43 respectively.

7 Conclusions and Future Efforts

We addressed the problem of transformations ordering for a variety of design metrics, such as throughput, area, and power. A connection between suitability of a computation for improvement using transformations and individual and compounded transformations is established and statistically validated. We also developed an approach for analyzing enabling and disabling relationships between transformations. The obtained insights and experimental data are a basis for a new, potential-driven approach for transformations ordering. For the development of ordering of transformations a novel statistical methodology has been employed.

The approach has low complexity from both conceptual and software implementation points of view. The approach is fully modular, easy to modify for new scenarios of design goals and

constraints, run-time efficient, and very effective. On comprehensive real-life benchmark set, improvements in throughputs, area, and power, by large factors have been obtained. The notion of potential of a computation for optimization generalizes the notion of structural bottlenecks and forms a basis for optimization of an arbitrary set of design goals and constraints. We are currently developing several potential-based compilation and synthesis tools for such tasks as template matching, code generation, partitioning and several other metrics including testability and fault tolerance.

Acknowledgments

We would like to thank Anna Poplawski and Toshio Misawa for their help in the first phase of the research presented in this paper.

REFERENCES

- [1] D.F. Bacon, S.L. Graham, and O.J. Sharp. Compiler transformations for high performance computing. *ACM Computing Surveys*, 26(4):345–420, 1994.
- [2] R.K. Brayton, G.D. Hachtel, C.T. McMullen, and A.L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer, Boston, MA, 1984.
- [3] A.P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Brodersen. Optimizing power using transformations. *IEEE Transactions on CAD*, 14(1):12, 1995.
- [4] S. Dey, M. Potkonjak, and S. Rothweiler. Performance optimization of sequential circuits by eliminating retiming bottlenecks. In *International Conference on Computer-Aided Design*, pages 504–509, 1992.
- [5] S. Dey, M. Potkonjak, and R. Roy. Exploiting hardware-sharing in high level synthesis for partial scan optimization. In *International Conference on Computer-Aided Design*, pages 20–25, 1993.
- [6] B. Efron and R.J. Tibashirani. *An introduction to the bootstrap*. Chapman & Hall, New York, NY, 1993.
- [7] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
- [8] H. Massalin. Superoptimizer: A look at the smallest program. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 122–126, 1987.
- [9] W.M. McKeeman. Peephole optimization. *Communications of the ACM*, 8(7):443–444, 1965.
- [10] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ, 1982.
- [11] M. Potkonjak and J. Rabaey. Maximally fast and arbitrarily fast implementation of linear computations. In *International Conference on Computer-Aided Design*, pages 304–308, 1992.
- [12] 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.
- [13] M.E. Wolf and M.S. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):452–471, 1991.