PROBABILISTIC CONTROL SEARCH STRATEGIES FOR HARDWARE AND SOFTWARE OPTIMIZATION DURING SOLUTION SPACE EXPLORATION

Jennifer L. Wong, Seapahn Meguerdichian, Farinaz Koushanfar
University of California, Los Angeles Computer Science Department
Los Angeles, CA
E-mail: {jwong, seapahn, farinaz}@cs.ucla.edu

Advait Morge, Dusan Petranovic
LSI Logic
Milpitas, CA
E-mail: {advait, dusan}@lsil.com

Miodrag Potkonjak
University of California, Los Angeles Computer Science Department
Los Angeles, CA
E-mail: miodrag@cs.ucla.edu

Abstract: In the last several years, system and integrated circuits (IC) semiconductor industry and research has started refocusing from the general purpose computing platform toward application specific devices and appliances. This shift, compounded with the exponentially growing gap between IC potential and design productivity imposes an urgent need for new design methodologies and technologies. There are four main phases in development of application specific systems (ASS): algorithm, architecture, implementation, and semiconductor realization. The last phase is mainly related to the technology CAD field and is out of main scope of the research presented in this paper.

The effectiveness of the first three phases is mainly dictated by employed optimization and search techniques. For this purpose, we have developed a new probabilistic iterative improvement generic technique. The technique has a number of noble properties including high quality of solution, low run-time, flexibility and ease of application. We have demonstrated the effectiveness of the new technique on two tasks: architectural design space for Infinite Impulse Response (IIR) filter and for the widely used Graph Coloring problem.

Introduction

One can identify four major phases in the design of next generation systems-on-chip. The first phase is concerned about translating application requirements into a program with a well-defined syntax and semantics and therefore underlying computational model. Note that this step is not difficult only because of its intrinsic difficulty in capturing all the requirements and constraints of complex applications, but also because there exists a large number of functionally isomorphic but structurally very diverse programs to capture

the same application. It is experimentally noticed that most often one can gain the largest design advantage in terms of common metrics (speed, area, and power) by selecting a suitable structure. As an illustration, let us state that for even small example such as an IIR, there are literally hundreds of different functionally equivalent filters, but in terms of implementation they have very different structures. Direct form, ladder, cascade, parallel, orthogonal, wave digital, and multi variable digital lattice are just a few of the different implementations.

Once the program structure is fixed and functionally debugged, the next step is the selection of suitable architecture. During this step, the designer makes commitments in terms of what kinds of executions units, storage elements, interconnect, and input/outputs will be used. For example, one can decide to use two super-scalar ALUs and a dedicated multiplier, each with register files and four memory banks connected using full crossbar switch. Again, for a given program there exists a very large set of potential architectures that can be used for implementation. One of the key commitments during these steps is which level of programmability will be employed. While the trade-off between programmable and dedicated architectures at the qualitative level is clear: programmable architecture provides flexibility at the expense of additional speed, area, and power overhead, the quantitative trade-off is difficult to exploit without knowing implementation details. The key commitment here is to develop a flexible template design/platform, which can be used with relatively minor modifications for a number of different but similar applications. Therefore, one can identify three major design sub-tasks in this phase 1) the development of template architecture 2) the development of tools for tuning a template for a particular application and 3) tools for mapping a given program to a given architecture. It is widely expected that for the last task retargetable compilers will be of special importance.

Now when the architecture is fixed, the next task is implementation in terms of primitives of IC design. There are a number of well-established fabrics, which can be used to implement the target architecture. For example, on can use custom design or cell-based ASIC, or Field Programmable Gate Arrays (FPGA). Implementation can be in particular crucial for clock cycle time and in deep submicron designs one has to consider in particular long interconnect delay. It is important to emphasize that already today different parts of the design can be implemented using different fabrics. The key trade-off at this level is one between the complexity of mask development, flexibility, and performance. For example, full custom design is superior in terms of performance, but often has low flexibility and very high mask cost. On the other hand, FPGA is rather flexible and has low mask cost, but can rarely match others quality in terms of performance.

The last step is the realization of implementation in silicon or semiconductor technology. Although this task is out of the scope of the paper and mainly technology CAD, proper discipline taken during the synthesis of the design during higher steps can greatly facilitate the manufacturability of a design.

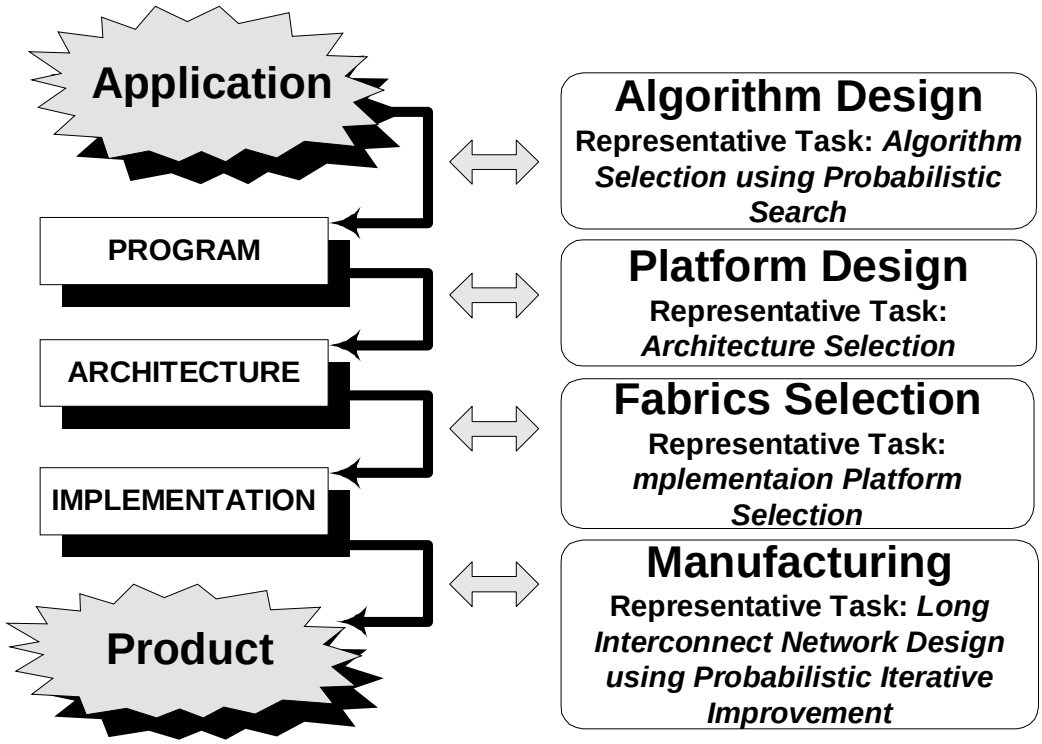

Fig. 1. Modern Design Process: Key Phases and Representative Tasks. Note that effectiveness of task in each phase is directly proportional to ability to search and optimize.

For all four phases, the key component for successful accomplishment is the ability of optimize. In this paper, we propose a new generic probabilistic iterative improvement search and optimization procedure. The key novelty of the procedure is the incorporation of explicit enabling moves for the avoidance of local minima. The procedure is generic in the sense that it enables very easy adaptation to essentially any arbitrary search or optimization task. The procedure has as input the specification of the problem in terms of objective function and constraints. Note that partitioners not only in optimization but also in the CAD field readily accomplish the task of defining an arbitrary problem in this form.

We make the following distinction between optimization and search. Optimization is used for well-defined tasks such as scheduling assignment, register allocation, and template matching. On the other side, search is used in this situation when evaluation of a particular proposed solution is time consuming. One has to evaluate the quality of the proposed solution in terms of accuracy using simulation which will provide information of bit error rate of a particular communication algorithm, required number of bits for realization of particular transfer function, or the expected and worse case runtime of particular program on a particular architecture.

The remainder of the paper is organized in the following way. The next section surveys the related research and development efforts in several fields. After that, we present the essential preliminary specifications in order to make the paper self-contained. In the fourth section, we present the core of the paper in terms of how the generic

probabilistic iterative improvement technique is defined for optimization purpose. Next, we explain how the new technique can be use for design space exploration and search. After experimental results, we summarize the results presented in this paper.

Related Work

The related work can be traced along six dimensions: design space exploration, IIR filters, algorithm selection and design, heuristic optimization algorithms, probabilistic optimization algorithms, and VLSI estimation techniques.

Although, IC component reuse has been widely practiced at many design centers since the beginnings of silicon designs, only in the last five years a strong consensus has been forming that IP reuse will be the dominant enabling force for the future generation of designs. A number of design companies have been making strong efforts to develop their IP portfolio, often mainly for internal use. There are also several companies (e.g. Artisan) who have completely based their business model on providing design IP. IP creation, assembly, and testing have also received significant recent research attention [Sch99, Del99, Zor99].

In the last several years, the system design process made a great shift from traditional concepts to new paradigms. In particular, the main emphasis has been placed on the development and effective use of platforms.

A fundamental task in digital signal processing, control applications, and communications is the filtering or processing of streams of data. [Opp89]. Infinite Impulse Response (IIR) filters are commonly used dur to their relatively low complexity of implementation. A number of computer-aided approaches for the design of IIR filters have been developed [Bur70, Hel71, Oet75, Rab74]. A wide mix of topological structures have been proposed for the realization of IIP filters, including direct form, cascade, parallel [Opp89], continued fraction [Mit72], ladder [Gra73], Wave digital [Fet86], state-space digital [Mul76], orthogonal [Rao84] and multi variable digital lattice [Vai85]. Commercial design tools such as MATLAB [MAT] and SPW [SPW] also support the synthesis of IIR filters.

Areas such as Artificial Intelligence have made algorithm selection and design a popular research topic. There are four main directions that have emerged in this area: first order logic-based methods [Gre69], transformational approaches [Dar81], schema-based programming [Wil83], and rewrite systems [Der85]. All of these techniques share one common weakness, the inability to scale to problems of practical importance, even though the techniques are very different in terms of strategy and algorithm.

Oppenheim and Nawab [Opp92] and Proudler et al. [Pro96] addressed the algorithm selection and design process for VLSI DSP. Our approach is in a sense most similar to the work by Potkonjak and Rabaey [Pot99]. While their approach is limited to the algorithm selection process, our approach explores the design space with a number of vital algorithm parameters taken into consideration. The major distinction between the approaches with respect to all the surveyed work is that our main goal at this level of

abstraction is to develop a technique for creation of IP property at the higher level of abstraction.

It appears that the first paper that introduced heuristic search is by Newell and Ernst [New65] and Lin [Lin65]. The first comprehensive experimental study of the heuristic procedure was done by Doran and Michie [Dor66]. In 1968, Hart, Nilsson, and Raphael [Har68] introduced a generic heuristic search technique named A*. Pohl [Poh77] and Pearl [Pea84] theoretically analyzed the procedure. Simultaneously, in the Operations Research community, another heuristic search technique was introduced, Branch-and-Bound [Law66]. Held and Karp [Hel71] in 1970 introduced the idea of relaxation-based heuristics. Prieditis described automatic generation of effective heuristic search techniques [Pri93]. The most constrained variable heuristic was introduced by Bitner and Reingold [Bit75]. The technique was systematically analyzed by Purdom [Pur83]. Brelaz [Bre79] introduced the idea of the least constraining variable heuristic. The minimally conflict heuristic was introduced by Gu [Gu89] and Minton [Min92].

The first iterative improvement technique was introduced by Kernighan and Lin [Ker70]. By using pair swap moves, the algorithm is able to iteratively move elements from partition to partition. Many different improvements on the strategy have been proposed. The most well-known being by Fiduccia and Mattheyses [Fid82], Sanchis [San89], and Krishnamurthy [Kri84]. Alpert and Kahng give a survey on the topic [Alp95].

Until recently many of the probabilistic algorithms proposed were probabilistic iterative improvement algorithms. Two of the main techniques are the Metropolis algorithm [Met53] and Simulated Annealing [Kir83, Joh91]. Simulated Annealing has been used for a number of applications in engineering, computer science and image recognition [Aar89]. Other approaches have been proposed such as Genetic Algorithms [Hol75, Sak97], Neural Networks [Hop82], Simulated Evolution [Fog66], and Tabu Search [Fri90, Glo89]. Most recently, a new probabilistic algorithm has been proposed, where the solution is iteratively built, instead of improved [Won01].

Wide estimation techniques can be used at all stages of the design process to predict the tree main metrics of design speed, area, and power [Rab94, Lan93, Ver94, Con99].

Preliminaries

Filters are functions or devices that implement a function that transforms a given sequence of input number according to a given rule into a sequence of output numbers. In particular they are often used to extract a certain part of the frequency spectrum from an input signal. Internal structure of the filters can be with or without feedback. Filters that do have feedback in their structure are commonly referred to as Infinite Input Response (IIR) filters, and often require significantly less computational resources for implementation or target mapping than structures that do not have feedback structures.

An IIR filters transfer function will completely represent its functionality. Figure 2 displays a typical transfer function for a low-pass IIR filter. Parameters such ass stopband and passband frequencies, stopband attenuation, gain, 3-dB bandwidth, and passband ripple help to characterize a filter. Any of the designs previously mentioned can be used

to implement an arbitrary transfer function. Each of the designs does differ in terms of hardware. Potkonjak and Rabaey have shown that different structures are better for different throughput requirements [Pot99].

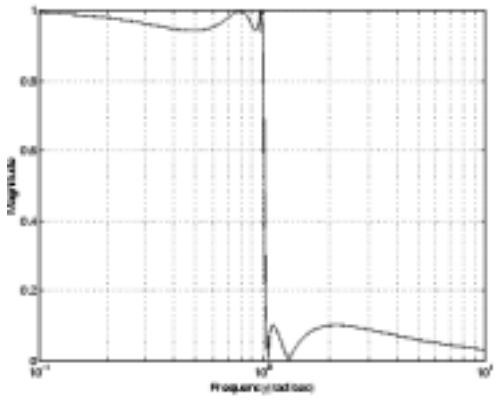

Fig. 2. Typical transfer function for low pass IIR filter (Elliptic IIR Filter)

We assume that the optimization problem is formulated in the following way. There are four components: input variables, output variables, objective function, and constraints. In particular an interesting and common case is when there is a single output variable. Let us denote the input variables as $X = [x_1,\ldots,x_n]$and the output variable by Y. We assume that the objective function is in the form MAX(Y) = cX, where c is a matrix with non-negative entries. Constraints are given in the form, $AX \geq B$, where the matrices A and B contain non-negative values. Note that in general, the matrices do not necessarily have constants as entries, but may have functions dependent upon the variables x_i. In our current implementation, we restrict our attention to a very special case, where matrices A, B, and c are constant matrices, and variables x_i are binary. Note that even with these restrictions, a large number of popular and common optimization problems can be represented in this form [Nem88].

There are two main components in the difficulty of conducting design space exploration. The first one is that usually the number of potential options is very high, and the topology of the solution space is not smooth. The second one is that it is difficult to evaluate the objective function even at a single given point. For example, one has to conduct high level and bit true simulation for long sequences of inputs. Or one has to figure out the expected or worse case run-time of a given program on a given architecture, which is given only in terms of its register transfer-level primitives. Therefore, in order to conduct efficient and effective design space exploration one often has to calculate or estimate the value of the objective function in a given point using some approximation scheme. The complete example of design space exploration is given in [Meg01].

Probabilistic Iterative Improvement Optimization

In this section, we introduce the generic form of our probabilistic iterative improvement optimization procedure. The first key enabling idea behind the technique is that many computational intractable problems can be represented in a generic form, which consists of three components: variable definition, objective function and constraints. Once the problem is stated in that way, it can be treated using the same optimization strategy and more importantly the same optimization software. For this task we propose a randomized (probabilistic) iterative improvement procedure. It is well known that iterative improvement algorithms are both fast and effective in solving numerous optimization problems. Superimposition of probabilistic mechanisms, not only further increases the effectiveness of the approach but also facilities even more favorable trade-offs between the quality of solution and run-time.

We make the assumption that all problems are defined in the following form.

$$\begin{aligned} &MAX(Y) = CX \\ &AX \geq B \end{aligned} \tag{1}$$

where A, B, and C are matrices with non-negative entries. We use this form without loss of generality because other forms can be accommodated with slight straightforward modifications.

We start with a randomly generated solution and iteratively go through the steps of selecting the best candidate component for solution improvement and deciding in which way to improve the current solution. During this procedure, special emphasis is placed on avoiding being stuck in local minima by following greedy improvements, as it is the case in the traditional iterative improvement techniques. In addition to randomization for this purpose, we use explicit enabling moves and newly developed maximally constrained minimally constraining heuristics integrated into iterative improvement. In the remainder of this section, we further elaborate the probabilistic iterative improvement procedure. In addition to explaining the generic procedure, we use graph coloring as our driver example to illustrate the relationship between the problem stated using objective function and constraints format and in its initial natural domain of graph representation.

Before we formally introduce our approach using pseudo code, in order to make the presentation easier to follow and self-contained, we introduce the graph coloring problem. The graph coloring problem is a natural abstraction of many applications where resources have to be shared. For example, all compilers use graph coloring to assign variables to registers in such a way that a minimal number of registers are used. In this case, each variable is represented as a node and there is an edge between two nodes if the corresponding variables have overlapping lifetimes. Each color corresponds to a register.

Graph coloring has a number of different applications [Jen95] including microcode optimization, scheduling, resource binding and sharing, register assignment, state encoding of finite state machines [DeM94], circuit testing [Bus95], operations scheduling [Cou97], polygon-guarding problems [Hof96], load balancing [Jos95], wireless spectrum estimation [Kha98], circuit clustering [Sin99], and multi-layer planar routing [Con93].

Optimization algorithms for graph coloring may be classified into three major classes: exact [Bre79, Cou97], heuristic constructive [Kar98, Hal93, Lei79] and iterative

improvement [Cha87, Mor86, Veg99]. In addition several specialized hardware platforms for efficient graph coloring has been developed including, ones based on couplet oscillators [Wu98] and reconfigurable FPGAs [Lee98].

```
1.  Translate to OF + Constraints form
2.  Problem Characterization
3.  Generate Initial Solution
4.  Evaluate Initial Solution
5.  While (control strategy is not satisfied)
6.  {
7.      While (stopping criteria is not satisfied)
8.      {
9.          Evaluate all candidates
10.         Rank all candidates
11.         Select candidate (BCS)
12.         Evaluate all moves for candidate
13.         Rank all moves for candidate
14.         Select Move for candidate (Best Move selection)
15.         Update Parameters
16.         Update current solution
17.     }
18.     Gather Information
19.     Update Current solution (solution integration)
20.     }
```

Fig. 3. Generic Non-Greedy Probabilistic Iterative Improvement Procedure

The graph coloring problem can be formally stated using the standard Garey-Johnson [Gar79] format in the following way:

Problem: ***Graph K-Colorability***
Instance: *Graph G(V,E), positive integer* $K \leq |V|$
Question: *Is G colorable, i.e., does there exist a function* $f : V \rightarrow 1,2,3,\ldots,K$ *such that* $f(u) \neq f(v)$ *whenever* $u,v \in E$*?*

The new procedure can be defined using the pseudo-code shown in figure 3. The first step of the procedure is to manually state the problem in the form of objective function plus constraints. As we already stated, this form has three components: optimization variable definition, objective function and constraints. Although for numerous problems, both the objective function and constraints will be linear and there will be a requirement that the variables have integer values, it is important to emphasize that the application domain is by no means restricted to the integer linear programming (ILP) problems. This step is often straightforward and easy, particularly if one does not insists that restriction to linear and integer constraints are enforced. However, at the same time this step also requires a certain level of creativity, in particular if one wants to further enhance

the effectiveness of the overall optimization procedure. This is so because the definition of the objective function and constraints often has high impact on the behavior of the optimization process.

Let us illustrate this step using the Graph Coloring problem. First, we define variables $\boldsymbol{x_i}$ and $\boldsymbol{x_{ij}}$:

(2) $\boldsymbol{x_i}$ = 1 if color i is used; 0 otherwise.
$\boldsymbol{X_{ij}}$ = 1 if node j was colored with color i; 0 otherwise.

The objective function of the graph-coloring problem is to minimize the number of colors used to color every node of the graph. Thus, the objective function can be written in the form

(3) $$\text{Max(Y)} = c\boldsymbol{x}$$

where c is an identity vector. The next step is to define the constraints for this problem. The first constraint relates $\boldsymbol{x_i}$ and $\boldsymbol{X_{ij}}$. The constraint requires $\boldsymbol{x_i}$ to be 1, when color i was used, that is, if there is at least one $\boldsymbol{X_{ij}} \neq 0$.

Therefore, we have the following constraint:

(4) $$\forall j \in V : x_i \geq X_{ij}$$

The second constraint guarantees that two connected nodes will not have the same color. Let's consider an edge matrix E containing elements $\boldsymbol{e_{mn}}$, which are 1, when nodes m and n are connected, and 0, otherwise. Therefore, the second constraint can be stated as:

(5) $$\forall e_{mn} = 1 : X_{in} + X_{im} \leq 1$$

The third constraint guarantees that all vertices of the graph are colored:

(6) $$\forall j = V : \sum_j X_{ij} \geq 1$$

In the second step, we extract characteristics from the instance of the optimization problem. There are two broad classes of characteristics that are targeted. The first one is related to the problems intrinsic nature. This set includes characteristics such as linearity or type of non-linearity in the objective function and constraints, number of different types of constraints, and number and type of constraints imposed on variables. The second part of characteristics targets the specific properties of the specific instance. They include the strictness of the constraints, the number of constraint in which each variable occurs, correlation between the appearances of different variables, and the ratio of number of constraints versus the number of variables.

Let us illustrate these two types of characteristics using specific examples. In the first group, we are trying to extract what is the nature of the problem. For example, a number of problems such as Maximal Independent Set and Maximal Clique [Gar79] are focused

on finding a part of the problem where a particular property holds. In other problems, such as Set Cover and Graph Coloring, the goal is to address properly all components of the instance. Obviously the search for a part of the solution has to be organized somewhat differently than an effort where all components of the problem have to simultaneously considered.

Regarding the second set of properties, the goal is to identify specific characteristic of an instance. For example, very sparse graphs have to be colored differently than dense graphs, graphs with uniform density of edges have to be colored differently than graphs with one or few large cliques and otherwise low density of edges.

The guidance about how to address the problem in all stages of optimization process is provided by problem and current solution properties. The properties can be classified along two lines: global and local, and one related to constraints and one related to variables. The global properties include the ratio of the number of constraints to the number of variables, average and median difficulty of constraints, the number of constraints, the number of variables, and the average and median level of constraints imposed on all variables. For the local properties, we define scope as the extent to which we analyze neighborhood of a variable or constraint. For example, level 1 indicates that only the variable is considered, level 2 that the variable and all other variables that occur in at least one constraint simultaneously with that variable are considered, and so on. In addition, we analyze the relationship between variables and the objective function, and constraints and the objective function. For all constraints, we define a notion of difficulty that indicates how likely it is that a particular constraint is satisfied during the optimization process.

The next step is to generate an initial solution or starting point for the iterative improvement approach. There are two main types of random initial solutions, feasible and non-feasible. Generate initial solution (Line 3) has the meaning that each variables is assigned a value. For the graph coloring problem this means that each node is assigned a color. Feasible initial solutions are solutions that satisfy all constraints in the problem instance. Of course this solution almost never is of high quality. A non-feasible solution can be specified further to incorporate the level of constraint violation that the solution has. These solutions could be specified by the percentage of constraints in the instance which are violated, or by specifying that the solution must not have more than k constraints violated. Note that this can make the generation of an initial solution extremely expensive or even impossible. Therefore, we provide a run-time limit on the Generate Initial Solution procedure.

Once the initial solution is generated, we have two options: one is to accept it, and initiate probabilistic iterative improvement optimization or to reject it. In general, there are two main reasons why a solution may be rejected. The first one it that it is of so few quality in terms of the OF, and number and difficulty of violated constraints, that it is not an attractive starting point for iterative improvement. The second reason for elimination is that the solution is too close to one of the already calculated initial or final solutions that there is a high probability that it will result in a solution already found. In the current version of our work, we are focused on the problems that can be defined using only binary variables. Although it may sound as overly restricted assumption, it is important to note that almost all NP-complete problems and numerous practically important optimizations belong to this class of problems.

We estimate that a solution is of low quality by employing that following formula

$$F = \alpha \text{ (Value of OF)} + \beta \left(\sum_{i}^{n} d_i \right) \quad (7)$$

where d_i is the difficulty of constraint i, and n is the number of violated constraints. We define weight factors α and β, which the user can assign using the feedback from the properties. The similarity of two solutions is quantified using the Hamming distance between two vectors that represent the binary solution. Each component of the hamming distance can be weighted according the number of occurrences each variable has on the constraints and/or the variables weight in the objective function.

One mechanism that often enhances the benefits of this step is to use linear programming to solve the problem. The linear program will assign rational values to the variables, instead of binary 0-1 values that are the ultimate goal is. Based on these values, an initial assignment of the variables can be formed by probabilistically assignment. For example, in the graph coloring problem, each node has n variables that denote all possible colors that can be assigned to the variable. One of the colors is assigned to the node with the probability determined by the linear programming value assigned to the variable. Specifically, we assign the largest value to be 1, and all other values for that node to be 0. Another mechanism is to pre-evaluate the random solutions to determine its potential for success. The potential of success is proportional to the value of the objective function and inversely proportional the number of violated constraints.

The control strategy can be user specified or determined by no improvement observed in the quality of the solutions after *k* number of attempts. The stopping criteria are satisfied in one of two ways. The first is that a feasible solution is found. This option is not applicable when the initial solution is a feasible solution. The second is to stop searching for a better solution when no improvements in the OF is made after k attempts.

Since we restricted our attention to 0-1 variables candidates and moves are naturally defined in the following way. Each 0-1 variable is a candidate, and a move is either switching from 0 to 1 of from 1 to 0. Therefore, in this case the selection of the move is fully determinate by the selection of the candidate, or in the case of graph coloring, the variable.

To evaluate each of the possible candidates or variables, we look at two different criteria. The first is the improvement a move on the variable will have on solution. One improvement is on the value of the objective function. The second is on the number of violated constraints. The second criterion is a measure of the variables enabling quality. We say that a variable is enabling if by changing the value of the variable we allow other variables to have more options than the number available before the change. It is possible that an enabling move will be at the expense of the objective function.

Currently we use two options for the control strategies. One is automatic and guided by the estimated likelihood that better solutions will be found within a reasonable amount of additional run-time. The second is that the user input is followed and feedback is provided about the probability that a better solution will be found. The decision to terminate the search is left to the user.

Probabilistic Iterative Improvement Search

In this section, we explain how probabilistic search techniques can be used for design space exploration at the algorithm level for realizing an IIR filter with given characteristics. For the architecture, we target an ASIC implementation with dedicated functional units and hardware sharing. As implementation fabrics, we target a cell-based design with dedicated register files and custom interconnect network. We assume a feature size of 0.35 microns. There are several parameters that impact the performance and the computational complexity of IIR filters. Table 1 lists the parameters that we consider during the design process. The number of bits, B, is deduced using the comparison of the double floating point (infinite precision) and simulation for the fixed number of bits varying from 5 to 40. It is interesting to note that in some cases for some structures even 40 bits were not sufficient, while in a few cases (e.g. some forms of wave digital structure) 4 bits were significant to achieve the desired transfer function. The second parameter is T, the number of taps. We vary the value in the range of 2 to 10. Sometimes 10 taps were not sufficient enough to achieve the transfer function, in this case the hardware requirements were too high to justify further search. We considered as a starting point 10 different filter structures, mainly those listed in the related work section.

It is well known that transformations often have dramatic impact on the key parameters of design. Therefore we considered 5 different transformations which usually have the highest impact on the area or speed. They are 1) the way how multiplication is implemented on the temporal transformation for retiming and pipelining, and 2) algebraic transformations associatively as applied to consecutive additions and rescaling as applied to multiplications that are on the same path from the input to the output.

B	Bits in data path {5,6, …, 39, 40}
T	Number of taps (subsections) {2,3, …, 9, 10}
S	Structures {10 different including parallel, cascade, ladder}
M	Multiplication realization {10 different}
R	Retiming {10 - area, critical path, …}
A	Consecutive addition realization {5}
P	Level of pipelining {1, 2, 3, 4, 5}
R_T	Rescaling Transformation {10 different schemes}

Table 1 – IIR Filter Parameters

Probabilistic iterative improvement search is conducted in the following way. Note that due to the large size of the solution space, almost 10 million sample points, exhaustive search methods are not applicable. First, the solution is evaluated using hyper-synthesis and estimation tools. In addition, the solution is evaluated at k different points which are neighbors to that point. Among them, the one which has best improvement in objective function (area) and which is the furthest form the currently best observed solution is accepted as the new state solution. This search is conducted as long as no improvement is detected in n steps. During our implementation we used $n = 4$ to obtain the experimental results. This basic search routine is with in the control strategy loop

which uses all mechanisms of the generic probabilistic iterative improvement technique to enhance the chances of finding high quality solutions. Specifically, we generate the initial solutions and evaluate them according to their potential and diversity to the already considered solutions.

An instance of an IIR filter can be defined by the following performance characteristics *(i)* 3-dB bandwidth, *(ii)* Area, *(iii)* Throughput, and *(iv)* Latency. The SPW software simulations are used to measure gain, 3-dB bandwidth, passband ripple and stopband attenuation characteristics. The HYPER behavioral synthesis tool [Rab91] is used to obtain area, throughput and latency. We use the HYPER tools for early estimation of both active logic area (execution units, registers and interconnect) as well as statistical tools for prediction of total area. The final implementation is obtained using Hyper and Lager tools [Rab91].

Entering the user specified transfer function in SPW does the evaluation of each candidate for implementation. This will generate Silage code which is used as input to the HYPER behavioral synthesis tool. Timing information, such as the number of cycles used and the length of the clock cycle is also outputted by HYPER. This information is then used to calculate throughput and latency.

Experimental Results

In this section, we present the experimental results obtained by testing probabilistic iterative improvement on two benchmarks. Graph coloring is used for the evaluation of the effectiveness of the new procedure as an optimization tool. The selection of the design of the IIR filter is used for the evaluation of the new procedure as a design space exploration tool.

Table 2 provides the experimental results for the application of the probabilistic iterative improvement procedure to the Graph Coloring problem. Testing was performed on instances from [Con93, Dim93]. The first column identifies the name of the instance and the second and third show the number of vertices and edges, respectively. The chromatic number for each instance is listed in the fourth column. The results of the probabilistic iterative improvement technique are shown in the last column. The experimental results in all cases are as strong as the best previously published or better.

We also applied the probabilistic iterative improvement technique to a set of random $R(V, p)$, $p = 0.5$ graphs. These graphs are exceptionally difficult for coloring, as is well known. Our results are compared to those of the lmRLF algorithm [Kir98].

Table 3, the first row rpresents the size of the random coloring instance. The second row is the chromatic number for each of these instances. We present the average number of colors used to color the graph and the average runtimes in seconds for a set of 10 graph instance for each random graph, for both the lmRLF and the probabilistic iterative improvement technique. On average, the probabilistic iterative improvement technique colors the random graphs *0.5* colors less and the runtime was an order of magnitude lower than the lmRLF algorithm's runtime.

Name	V	E	χ	# colors
c-fat200-1	200	1534	12	13
DSJC125.1	125	736	5	6
Exam1	200	17124	126	126
Exam3	300	36801	162	162
flat300_20_0	300	21375	20	40
le450_25c	450	17343	25	28
le450_5d	450	9757	5	8
MANN_a9	45	918	18	18
queen7_7	49	476	7	10
queen9_9	81	2112	10	12
queen13_13	169	6656	13	17
R125.5	250	3838	36	38
sgelq2.I.2	182	3254	26	26
school1_nsh	352	14612	14	24

Table 2 - Experimental Results for Graph Coloring.

	R (125, 0.5)		**R (250, 0.5)**		**R (500, 0.5)**		**R (1000, 0.5)**	
	16		***27***		***46***		***80***	
	Cols	Time	Cols	Time	Cols	Time	Cols	Time
lmXRLF	18.2	46.2	29.9	172	49.7	4612	85.1	18083
Probabilistic	17.9	9.12	29.4	37.1	48.9	843.4	84.5	3004.3

Table 3 - Experimental Results of random Graph Coloring examples.

Throughput *(μs)*	**It. Improv. Area** *(mm^2)*	**Average Area** *(mm^2)*	**Reduction** *%*	**Structure**
5	5.62	15.75	64.32	Ladder
4	5.84	18.27	68.04	Parallel
3	5.84	19.94	70.71	Parallel
2	5.84	21.08	72.30	Parallel
1	5.99	35.81	83.27	Parallel
0.5	11.63	69.98	83.38	Cascade
0.25	22.14	158.90	86.07	Cascade
0.1	43.76	415.80	89.48	Cascade
0.05	91.36	899.08	89.84	Cascade

Table 4 – Performance of Probabilistic Iterative Improvement Algorithm on IIR Filter

of probabilistic iterative improvement search on a set of IIR designs specified in the following way: $\omega_{p1}=0.411111\pi$, $\omega_{p2}= 0.466667\pi$, $\varepsilon_p=0.015782$, $\omega_{s1}=0.3487015\pi$, $\omega_{s2}=0.0.494444\pi$, $\varepsilon_s=0.0157816$, where ω_{p1} and ω_{p2} are the bandpass frequencies, ω_{s1} and ω_{s2} are stopband frequencies, ε_p is the passband ripple, and ε_s is the stopband ripple (assuming a standard normalized filter characteristics).

The results of applying the probabilistic iterative improvement search to the IIR filter are shown in Table 4. The throughput of the filter (μs) is shown in the first column. The next column displays the best solution found by the search procedure in terms of area. The third column presents the average case solution, again in terms of area, while the fourth column displays the percentage improvement in terms of area after applying the search technique. Finally, the type of structure used for the filter is presented in the last column. An average of 78.6% and median of 83.27% improvement in area was obtained over all the designs.

Conclusion

Effective and efficient optimization techniques are the main enabling technology in all three phases of the next generation of reusable platform-based synthesis CAD tools: algorithm selection, architecture definition and implementation realization. To address this need we have developed a new probabilistic iterative improvement generic optimization technique.

The technique enables generation of high quality solutions in low run-time by combining key features of iterative improvement techniques and randomization. The single most important feature of the technique is exceptional flexibility and ease of use for a great variety of optimization and search problems. In order to evaluate the effectiveness of the new technique we chose two tasks: architectural design space for the IIR filter and the widely used graph coloring problem. In both cases were able to achieve results of superior quality in shorter run-times than the ones reported by previously published techniques.

References

[Aar89] E. Aarts, J. Korst. Simulated Annealing and Boltzmann Machines: a stochastic approach to combinatorial optimization and neural computing**.** New York: Wiley, 1989.

[Alp95] C.J. Alpert, A.B. Kahng. Recent Directions in Netlist Partitioning: A Survey. Integration: The VLSI Journal, Vol. 19, 1995.

[Bit75] J. Bitner, E. M. Reingold. Backtrack programming techniques. Communications of the ACM, Vol. 18, pp. 651-655, 1975.

[Bre79] D. Brelaz. New methods to color the vertices of a graph. JACM, Vol. 22, No.4, pp. 251-256, 1979.

[Bur70] C.S. Burrus, T.W. Parks, "Time domain design of recursive digital filters. IEEE Transactions on Audio and Electroacoustics, vol.AU-18, (no.2), June 1970. p.137-41.

[Bus95] F.Y. Busaba, P.K. Lala. A graph coloring based approach for self-checking logic circuit design. Asian Test Symposium, IEEE Computer Society Press, pp.327-330, 1995.

[Cha87] M. Chams, A. Hertz, D. de Werra. Some Experiments with Simulated Annealing for Coloring Graphs. European Journal of Operations Research, Vol. 32, pp. 260-266, 1987.

[Con93] J. Cong, M. Hossain, N.A. Sherwani. A provably good multilayer topological planar routing algorithm in IC layout designs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol.12, No. 1, pp.70-78, 1993.

[Con99] J. Cong and D. Z. Pan. Interconnect delay estimation models for synthesis and design planning. Asia and South Pacific Design Automation Conf., pp. 97-100, 1999.

[Cou97] O. Coudert. Exact coloring of real-life graphs is easy. ACM/IEEE Design Automation Conference, pp.121-126, 1997.

[Dar81] J. Darlington, "An experimental program transformation and synthesis system", Artificial Intelligence, Vol. 16, pp.1-46, 1981.

[Del99] M. Delpasso, A. Bogliolo, L. Benini, "Virtual simulation of distributed IP-based designs", 36-th DAC, pp. 50-55, 1999.

[DeM94] G. De Micheli. Synthesis and optimization of digital circuits. New York: McGraw-Hill, 1994.

[Der85] N. Dershowitz, "Synthesis by Completion", The Ninth Int. Joint Conference on Artificial Intelligence, pp. 208-214, 1985.

[Dim93] ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmark/. Benchmark for graph coloring and clique, 1993.

[Dor66] J. Doran, D. Michie. Experiments With The Graph Traverser Program. Proceedings of the Royal Society of London (A), 294, pp. 235-259, 1966.

[Fet86] A. Fettweis, "Wave digital filters: theory and practice." Proc. of IEEE, vol.74,, Feb. 1986. p.270-327.

[Fid82] C. M. Fiduccia, R.M. Mattheyses. A Linear Time Heuristic for Improving Network Partitions. ACM/IEEE Design Automation Conference, pp.175-181, 1982.

[Fog66] L.J. Fogel, A.J. Owens, M.J. Walsh. Artificial Intelligence through Simulated Evolution. New York: John Wiley, 1966.

[Fri90] C. Friden, A. Hertz, D. De Werra. TABARIS: an exact algorithm based on Tabu search for finding a maximum independent set in a graph. Computers & Operations Research, Vol. 17, No. 5, pp.437-445, 1990.

[Gar79] M.R. Garey, D.S. Johnson. "Computers and Intractability: A Guide to the Theory of NP-Completeness." New York: Freeman, 1979.

[Glo89] F. Glover. Tabu search part I. ORSA Journal on Computing, Vol.1, No. 3, pp.190-206, 1989.

[Gra73] A.H Gray Jr., J.D. Markel, Digital lattice and ladder filter synthesis. IEEE Transactions on Audio and Electroacoustics, vol.AU-21, (no.6), Dec. 1973. p.491-500.

[Gre69] C. Green, "Application of Theorem Proving to Problem Solving", Proc. Of the First International Joint Conference on Artificial Intelligence, pp. 219-239, 1969.

[Gu89] Gu. Parallel Algorithms and Architectures for Very Fast AI Search. PhD thesis, Dept. of Computer Science at The University of Utah, 1989.

[Hal93] M.M. Halldorsson. Approximating the minimum maximal independence number. Information Processing Letters, Vol.46, No. 4, pp.169-72, 1993.

[Har68] P. E. Hart, N. J. Nilsson, B. Raphael. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, Vol. 4, No. 2, pp. 100-107, 1968.

[Hel71] H.D. Helms, "Digital filters with equiripple or minimax responses". IEEE Trans. on Audio and Electronics, vol.Au-19, (no.1), March 1971, p.87-93.

[Hel71] M. Held and R. M. Karp. The traveling salesman problem and minimum spanning trees: Part II, Mathematical Programming, Vol. 6, pp. 62-88, 1971.

[Hof96] F. Hoffmann, K. Kriegel. A graph-coloring result and its consequences for polygon-guarding problems. SIAM Journal on Discrete Mathematics, Vol. 9, No. 2, pp.210-24, 1996.

[Hol75] J. Holland. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. Ann Arbor: University of Michigan Press, 1975.

[Hop82] J.J. Hopfield. Neural Networks and Physical Systems with Convergent Collective Computational Properties. National Academy of Science (U.S.), Vol.79, pp.2554-2558, 1982.

[Jen95] T. Jensen and B. Toft. Graph Coloring Problems. New York: John Wiley & Sons, 1995.

[Joh91] D. S. Johnson, C. R. Aragon, L. A. McGeoch, C. Schevon. Optimization by simulated annealing: An experimental evaluation, part II, graph coloring and number partitioning. Operations Research, Vol. 39, No. 3, pp.378-406, 1991.

[Jos95] B.S. Joshi, S. Hosseini, K. Vairavan. Performance evaluation of a graph coloring based load balancing algorithm. IASTED/ISMM: International Conference Parallel and Distributed Computing and Systems, pp.1-4, 1995.

[Kar98] D. Karger, R. Motwani, M. Sudan. Approximate graph coloring by semi-definite programming. Journal of the ACM, Vol. 45, No. 2, pp.246-65, 1998.

[Ker70] B. W. Kernighan, S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. Bell System Technical Journal, Vol. 49, No. 2, pp. 291-307, 1970.

[Kha98] S. Khanna, K. Kumaran. On wireless spectrum estimation and generalized graph coloring. IEEE INFOCOM '98, the Conference on Computer Communications, Vol. 3, pp. 1273-1283, 1998.

[Kir83] S. Kirkpatrick, C. Gelatt, M. Vecchi. Optimisation by Simulated Annealing. Science, No. 220, pp. 671-680, 1983.

[Kir98] D. Kirovski, M. Potkonjak. "Efficient coloring of a large spectrum of graphs." Proceedings of 35th Design Automation Conference (DAC), pp.427-32, 1998.

[Kri84] B. Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. IEEE Transactions on Computers, Vol.C-33, No. 5, pp.438-46, 1984.

[Lan93] P. E. Landman and J. M. Rabaey, Power Estimation for High Level Synthesis, European Conference on Design Automation, pp. 361-366, 1993.

[Law66] E.L. Lawler, D.E. Wood. Branch-and-bound methods: A survey. Operations Research, Vol. 14, pp. 699-719, 1966.

[Lee98] T.K. Lee, P.H.W. Leong, K.H. Lee, K.T. Chan, S.K. Hui, H.K. Yeung, M.F. Lo, J.H.M Lee. An FPGA implementation of GENET for solving graph coloring problems. IEEE Symposium on FPGAs for Custom Computing Machines, IEEE Computer Society, pp. 284-285, 1998.

[Lei79] F.T. Leighton. A Graph Coloring Algorithm for Large Scheduling Algorithms. Journal of Research National Bureau Standards, Vol. 84, pp.489-506, 1979.

[Lin65] S. Lin. Computer solutions of the traveling salesman problem. Bell Systems Journal, Vol. 44, pp. 2245-2269, 1965.

[MAT] http://www.mathworks.com/

[Meg01] Seapahn Meguerdichian, Farinaz Koushanfar, Advait Morge, Dusan Petranovic and Miodrag Potkonjak. MetaCores: design and optimization techniques. ACM/IEEE Design Automation Conference. pp. 585 – 590, 2001.

[Met53] N. Metropolis, A. Rosenbluth, R. Rosenbluth, A. Teller, & E. Teller. Equation of state calculations by fast computing machines. Journal Chemical Physics, Vol. 21, pp.1087-1092, 1953.

[Min92] S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling. Artificial Intelligence, Vol. 58, pp. 161-205, 1992.

[Mit72] S.K. Mitra, R.J. Sherwood, "Canonic realizations of digital filters using the continued fraction expansion." IEEE Trans. on Audio and Electroacoustics, vol.Au20, (no.3), Aug. 1972. p.185-94.

[Mor86] C. Morgenstern, H. Shapiro. Chromatic Number Approximation Using Simulated Annealing. Unpublished, 1986.

[Mul76] C.T. Mullis, R.A. Roberts **"**Synthesis of minimum roundoff noise fixed point digital filters". IEEE Trans. on Circuits and Systems, vol.CAS-23, p.551-62, Sept 1976.

[Nem88] G. L. Nemhauser and L. A. Wolsey. Integer and Combinatorial Optimization. John Wiley, 1988.

[New65] A. Newell, C. Ernst. The search for generality. Proc. IFIP Congress 65, Vol. 1, pp.17-24, 1965.

[Opp89] A.V. Oppenheim, R.W. Shafer: "Discrete-time Signal Processing", Prentice Hall, Englewood Cliffs, NJ, 1989.

[Opp92] A.V. Oppenheim, S.H. Nawab, "Symbolic and Knowledge-based Signal Processing", Englewood Cliffs, NJ, 1992.

[Oet75] G. Oetken, T.W. Parks, H.W. Schussler, “New results in the design of digital interpolators.” IEEE Trans. on Acoustics, Speech and Signal Processing, vol.ASSP-23, (no.3), June 1975. p.301-309.

[Pea84] J. Pearl. Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, 1984.

[Poh77] I. Pohl. Practical and theoretical considerations in heuristic search algorithms. Machine Intelligence, Vol. 8, pp. 55-72, 1977.

[Pot99] M. Potkonjak, J.M. Rabaey, “Algorithm selection: a quantitative optimization-intensive approach.” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.18, (no.5), IEEE, May 1999.

[Pri93] A. E. Prieditis. Machine discovery of effective admissible heuristics. Machine Learning, Vol. 12, pp. 117-141, 1993.

[Pro96] I.K. Proudler, J.G. McWhirter, M. Moonen, G. Hekstra, “Formal Derivation of a systolic array for recursive least squares estimation”, Trans. On Circuits and Systems II: Analog and Digital Signal Processing, Vol. 43, No. 3, pp.247-254, 1996.

[Pur83] P.W. Purdom. Search rearrangement backtracking and polynomial average time. Artificial Intelligence, Vol. 21, pp. 117-133, 1983.

[Rab74] L.R. Rabiner, J.F. Kaiser, O. Herrmann, M.T. Dolan, “Some comparisons between FIR and IIR digital filters.” Bell System Tech. Journal, vol.53, Feb. 1974. p.305-31.

[Rab91] J.M. Rabaey, C. Chu, P. Hoang, M. Potkonjak, “Fast prototyping of datapath-intensive architectures.” IEEE Design & Test of Computers, vol.8, June 1991. p.40-51.

[Rab94] J. M. Rabaey, M. Potkonjak. Estimating implementation bounds for real time DSP application specific circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol.13, No. 6, pp. 669-683, 1994.

[Rao84] S.K Rao, T. Kailath, “Orthogonal digital filters for VLSI implementation.” IEEE Transactions on Circuits and Systems, vol.CAS-31, (no.11), Nov. 1984. p.933-45.

[Sak97] A. Sakamoto, X. Liu, T. Shimamoto. A Genetic Approach for Maximum Independent Set Problems. IEICE Transaction on Fundamentals, Vol. E80-A, No. 3, pp.551-556, 1997.

[San89] L.A. Sanchis. Multiple-way network partitioning. IEEE Transactions on Computers, Vol.38, No. 1, pp.62-81, 1989.

[Sch99] P. Schaumont, R. Cmar, S. Vernalde, M. Engels, and others. “Hardware reuse at the behavioral level”, Design Automation Conference, pp. 784-789, 1999.

[Sin99] A. Singh, M. Marek-Sadowska. Circuit clustering using graph coloring. Proceedings. International Symposium on Physical Design, pp.164-169, 1999.

[SPW] http://www.cadence.com/eda_solutions/

[Vai85] P.P Vaidyanathan, S.K. Mitra, “A general family of multivariable digital lattice filters”. IEEE Trans. on Circuits and Systems, Vol. CAS-32, (No.12), Dec. 1985. p.1234-1245.

[Veg99] S.R. Vegdahl. Using node merging to enhance graph coloring. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Vol.34, No. 5, pp.150-4, 1999.

[Ver94] I. M. Verbauwhede, C.J. Scheers and J. M. Rabaey. Memory Estimation for High Level Synthesis. ACM/IEEE Design Automation Conference, pp. 143-148, 1994.

[Wil83] D.S. Wile, “Program Developments: Formal Explanations of Implementations”, Comm. of the ACM, Vol. 26, No. 11, pp. 902-911, 1983.

[Won01] J. L. Wong, F. Koushanfar, S. Meguerdichian, M. Potkonjak. A Probabilistic Constructive Approach To Optimization Problems. ICCAD, 2001.

[Wu98] C. W. Wu. Graph coloring via synchronization of coupled oscillators. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, Vol.45 No. 9, pp. 974-978, 1998.

[Zor99] Y. Zorian, E.J. Marinissen, S. Dey, “Testing embedded-core-based system chips”, Computer, vol.32, pp. 52-60, 1999