

Generation of Yield-Aware Pareto Surfaces for Hierarchical Circuit Design Space Exploration

Saurabh K Tiwary
Carnegie Mellon University
Pittsburgh, PA USA
stiwary@ece.cmu.edu

Pragati K Tiwary
BIT Mesra
Ranchi, Jharkhand India
pragati398@bitmesra.ac.in

Rob A Rutenbar
Carnegie Mellon University
Pittsburgh, PA USA
rutenbar@ece.cmu.edu

ABSTRACT

Pareto surfaces in the performance space determine the range of feasible performance values for a circuit topology in a given technology. We present a non-dominated sorting based global optimization algorithm to generate the nominal Pareto front efficiently using a simulator-in-a-loop approach. The solutions on this Pareto front combined with efficient Monte Carlo approximation ideas are then used to compute the *yield-aware* Pareto fronts. We show experimental results for both the nominal and yield-aware Pareto fronts for power and phase noise for a voltage controlled oscillator (VCO) circuit. The presented methodology computes yield-aware Pareto fronts in approximately 5-6 times the time required for a single circuit synthesis run and is thus practically efficient. We also show applications of yield-aware Pareto fronts to find the optimal VCO circuit to meet the system level specifications of a phase locked loop.

Categories and Subject Descriptors

B.7.2 [Integrated Circuits]: Design Aids

General Terms

Algorithms Design

Keywords

Yield, performance space, Pareto surfaces, optimization

1. INTRODUCTION

Recent advances in design automation and increased computational power has led to a gradual transition from “hand-calculation” based analog circuit design to a simulation-based sizing methodology [3]. Simulation based synthesis uses efficient global optimization techniques to visit many circuit candidates, and fully evaluates each candidate via detailed simulation. This methodology works very well for circuits in the range of a few hundred devices. However, for larger circuits, the simulation time required for a single simulation is too large to do a practical simulator-in-the-loop circuit sizing. Also, with large number of design variables, the design space in which to search for optimal design points becomes too large for these circuits to be handled by the tools available today. To handle this problem, methods for modeling the performance space of circuits have been presented [10] [14] [5]. Using

these performance models, efficient system level design space explorations can be done for topology selection [9] as well as circuit synthesis [13]. Of late, much work has been done in the field of analog circuit macromodeling which aims at simulating these circuits faster [16] [8] [12]. These macromodels capture the important functional characteristics of the circuit while discarding the redundant information. They are faster to simulate than the original transistor level circuit and hence could be used to replace the circuits that they model in a system level simulation context. A class of models called *Pareto surfaces* generated in the performance space of the circuits has often been used with these macromodeling tools for hierarchical synthesis of larger systems [13]. These Pareto surfaces can be generated by both stochastic [19] [17] and deterministic algorithms [6] [15].

Pareto surfaces represent the best performance that can be obtained from a given circuit topology across its complete design space. These surfaces are generated for the nominal values of the process parameters and are often used for optimizing the performance parameters of the circuit in order to meet the system level specifications [13]. However, with reduction in feature sizes, the effect of process variations on circuit performance is becoming more pronounced. Such a system level design methodology gives us no idea of how the circuit would perform once fabricated with the optimized performance parameters in the presence of process variations. In the worst case, it is likely that the nominal design point lies in the tail of the performance distribution when Monte Carlo analysis is performed for the optimized circuit candidate across process variations. This, in turn, would result in poor *yield* (fraction of circuits that meet the system level specifications across all the circuits that are fabricated). Therefore, a *yield-aware* Pareto surface (points on the Pareto surface guaranteeing a fixed yield number) would be more useful for hierarchical synthesis applications.

In this paper, we present an efficient, novel algorithm for generating a yield-aware Pareto surface for an arbitrary custom circuit. To the best of our knowledge, this is the first realistic approach successfully implemented and validated for this problem. Sec.2 gives some basic background on Pareto methods and the functionality missing in prior efforts. Sec.3 describes the elements of our yield-aware Pareto generation methodology: nominal “soft” Pareto generation; yield-aware Pareto for a particular design point; and optimization of yield-aware Pareto curve across the design space. Sec.4 shows experimental results for a voltage controlled oscillator (VCO) circuit. Nominal and yield-aware Pareto fronts for different yield targets of 20%, 50% and 80% are generated for the circuit efficiently. The total time taken for the front generation is only 5-6X the CPU time required for circuit synthesis runs. These Pareto fronts are then used to hierarchically synthesize a PLL. Sec.5 offers concluding remarks followed by acknowledgements in Sec.6.

Permission to make digital or hard copies of all or part 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. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC 2006, July 24–28, 2006, San Francisco, California, USA.
Copyright 2006 ACM 1-59593-381-6/06/0007 ...\$5.00.

2. BACKGROUND

2.1 Circuit Sizing

A circuit sizing (synthesis) problem is to find the sizes/values of elements in a circuit block *e.g.*, W and L of transistors, *etc.*, such that the block meets a set of performance specifications. Formally, if \mathbf{p} is a vector of parameters values on which the performance function $\mathbf{f}(\mathbf{p})$ depends, then the sizing problem is to find a set of values for \mathbf{p} represented by \mathbf{p}_0 such that

$$\mathbf{f}(\mathbf{p}_0) < \mathbf{f}_g \quad (1)$$

Here, \mathbf{f}_g is the set of goal specifications for the circuit. Lower bounds on the performance parameters can be specified using this formulation by multiplying the goal value by -1.

If the statistical variation of the parameters is to be considered, *e.g.*, process variations, then the circuit sizing problem is rephrased as a yield improvement problem. The goal in such cases is to increase the *percentage* of circuit meeting the goal specifications (\mathbf{f}_g) for a particular set of input parameters (\mathbf{p}_0). A common method used to find the performance values for a particular set of input parameters is circuit simulation.

$$\mathbf{p} \mapsto \text{circuit simulation} \mapsto \mathbf{f}(\mathbf{p}) \quad (2)$$

SPICE like circuit simulators are the most common tools used for such mapping. Being the most accurate tool for finding such a mapping, the process of circuit simulation is also the most expensive. Automatic circuit sizing tools are often designed in such a way so as to do minimal number of SPICE simulation. For a constrained range of input parameters, regression based approaches for modeling the performance function is often employed for fast evaluations. However, the performance function is very non-linear for large variations in the parameter space. A lot of effort has been devoted to better model the performance as a function of design parameters [5] [10]. Such efforts often require a lot of SPICE level circuit simulation upfront for building the models.

2.2 Pareto Optimal Surfaces

Often, a number of the elements in the objective function $\mathbf{f}(\mathbf{p})$ compete against each other. Thus, the circuit optimization problem turns out to be a multi-objective optimization problem over the range of parameters (\mathbf{p}) such that some sizing constraints ($\mathbf{c}(\mathbf{p}) > \mathbf{0}$) are satisfied. For competing objective functions, it is not feasible to maximize the performance of all of them. In such case, we often have to consider *trade-offs* between the different performance parameters. Improving one might result in decreasing another competing performance function. In such cases, we strive for pareto optimality which ensures the best overall performance one could extract in the presence of competing performance functions.

Defining it formally, a set of performances \mathbf{f}_a is considered more optimal than another set \mathbf{f}_b if \mathbf{f}_a *dominates* \mathbf{f}_b .

$$\mathbf{f}_a \succ \mathbf{f}_b \Leftrightarrow \forall (f_{a_i} \geq f_{b_i}) \wedge \exists (f_{a_i} > f_{b_i}) \quad \text{for } i = \{1, \dots, n\} \quad (3)$$

A set of performances \mathbf{f}^* is considered pareto optimal if it is not dominated by any other set of performance parameters \mathbf{f} . The surface generated in the performance space by the complete set of pareto optimal points make up the *pareto optimal surface*.

2.3 Finding the Pareto Optimal Surface

Many tools for circuit sizing [3] have focused on finding a single point on the pareto surface. This problem can be transformed into a scalar optimization problem by weighting the different performance parameters.

$$\max \mathbf{o}(\mathbf{p}) = \sum_{i=1}^n w_i f_i(\mathbf{p}) \quad \text{s.t. } \mathbf{c}(\mathbf{p}) > \mathbf{0} \quad (4)$$

A set of weights $\mathbf{w} = \{w_1, \dots, w_n\}$ leads to a particular solution on the pareto front. The methods for obtaining the solution to this problem are fairly mature and various commercial softwares are also available for the purpose [2].

Considerable research has focussed on the automated generation of the pareto surfaces using deterministic and stochastic optimization techniques. One of the common methods for generating the points is by choosing different values for w_i in Eqn. 4. In such a case, the challenge is to find sets of weight values such that pareto front could be captured with least number of point evaluations of the surface. A method to automatically generate suitably spaced points on the pareto is the normal boundary intersection (NBI) method [6]. The basic idea of the algorithm is shown in Fig. 1. First, the global optimal points on the pareto are obtained for each

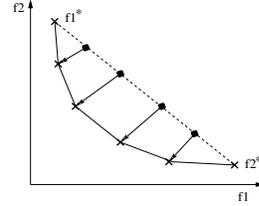


Figure 1: Normal boundary intersection (NBI) method.

of the performance dimensions ($f1^*$ and $f2^*$). The convex hull between the individual optima is described by

$$\mathbf{F} \cdot \mathbf{w} = w_1 f1^* + w_2 f2^* \quad \text{for } w_1, w_2 \geq 0; w_1 + w_2 = 1 \quad (5)$$

The convex hull is intersected at equidistant points as shown in Fig. 1. At each of these points the best performance set for the circuit is obtained in the direction of the normal to the convex hull. These points, obtained using standard circuit optimization techniques, are used to approximate the pareto surface.

Another method often employed for generating these pareto fronts is the non-dominated sorting genetic algorithm (NSGA) [7]. The basic idea is to start with a population of samples in the parameter space. These sets of parameters for points in the parameter space (*individuals*) are called *chromosomes*. During each generation, the individuals are sorted based on a *fitness* function (dominating points (Eqn. 3) are *fitter* for pareto surface generation). The second generation of population is generated by *crossover* and *mutation* of the chromosomes of the fitter individuals [7].

Simulated annealing [11] is another method quite extensively used for global optimization. We start at a randomly chosen point (*seed*) in the parameter space and then move to another point in the space with certain probabilities. The probability to move to the new point in the parameter space is higher if the performance values improve at this new point. Even in cases where the performance value degrades, the move is sometimes accepted (with low probability). This step ensures hill climbing properties of the algorithm.

2.4 Challenges in Generating Pareto Surfaces

The primary challenge that one faces while generating the pareto front is the curse of dimensionality. The number of points required to model a surface increases exponentially with the number of dimensions. The normal boundary intersection method, for example, requires proportional number of circuit optimization runs as the number of points required on the surface. Another problem is that the additional constraint of searching for optimal points in the direction perpendicular to the convex hull (Fig. 1) makes the optimization problem extremely difficult if we are using a stochastic optimizer to find the optimal point on the line. Also, for each point on the pareto front, one needs to run a complete sequence of the global optimization routine. This makes the method intractable for problems with high dimensionality.

Stochastic algorithms that dynamically handle the pareto front like non-dominated sorting genetic algorithm (NSGA-II) [7] are very efficient in generating the pareto fronts. The problem with using such an algorithm with applications in modeling the pareto surface of a given circuit topology is that, the resulting pareto front has no notion of *yield* for the circuit. The fronts thus generated only consider the *nominal* design parameter values for the circuit. Once the circuit is fabricated for a particular set of design parameter values corresponding to a point on the pareto surface, it is probable that a high percentage of the fabricated circuit’s performance is inferior to that point on the pareto due to variations in process parameters. This probability is very high for “aggressively” generated pareto fronts. Often the performance function is highly non-linear with respect to the process parameters (e.g. tox, vth etc.) of the circuit. Thus, if we neglect the process variation information, the optimizer may choose designs whose nominal performance parameters lie on the pareto front but, this nominal design actually lies in the tail of the performance distribution function across process variations.

The right – and so far, missing – solution is a yield-aware pareto surface. A trivial and brute-force approach would simply compute the yield for all circuit candidates through Monte Carlo sampling and thus, optimize the yield-aware pareto fronts. However, this method would be computationally very expensive if not prohibitive. The rest of the paper develops a new methodology that seeks to address this problem and generate yield-aware pareto surfaces efficiently.

3. YIELD-AWARE PARETO GENERATION

We have developed a *yield-aware* pareto front generation algorithm. The key steps are shown in Algorithm 1. The basic idea is to generate “soft” pareto surfaces for nominal design (without taking process variations into account). By “soft” we mean that we do not aggressively search for the actual pareto front using the stochastic algorithms described in Section 3.1. We stop the search after moderate convergence of the algorithm. Using points on this “soft” pareto, a fast *local* yield estimator algorithm is used to search for locally optimum pareto points. These points are then combined together to form the yield-aware pareto fronts. Before we discuss the details of our algorithm, let us revisit the nominal pareto generation problem.

Algorithm 1 Pseudo-code for yield-aware pareto front generation

- 1: *Given:* A circuit topology, ranges for design parameters and distributions of process parameters
 - 2: Generate “soft” nominal pareto using NSGA with “Ice Ages”
 - 3: Choose equi-distant pts $\{x_1, \dots, x_N\}$ on pareto front of order k
 - 4: **for all** x_i as anchor points **do**
 - 5: Generate *local* yield model
 - 6: Locally optimize the yield front around anchor points
 - 7: **end for**
 - 8: Merge all the local yield fronts to generate a yield-aware pareto front for the given circuit topology
-

3.1 NSGA with “Ice Ages”

We use a combination of genetic algorithm (GA) [7] and simulated annealing (SA) [11] ideas to generate our nominal pareto front (Figure 2). Genetic algorithm and simulated annealing optimizations are sequentially used with underlying non-dominated sorting ideas to improve the pareto front. Each set of these sequences is termed an “epoch”. During the whole process, points are sorted using their non-domination rank and crowding distance

[7] for efficient, spread-out pareto surface generation with minimal number of points.

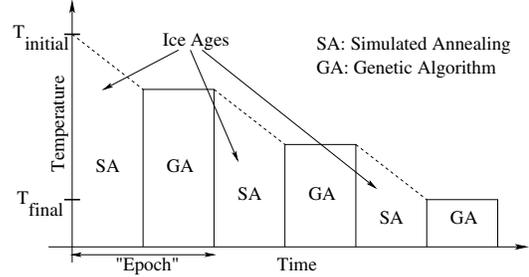


Figure 2: Pareto front generation schedule.

If we take a close look at GA, the two steps that bring in diversity to the population sample are: (a) crossover and (b) mutation. Crossover intermingles the parameter values of the highly ranked individuals in the population. Mutation, on the other hand, results in sampling of new regions of the parameter space that are close to the spatial location of the current individual. This step is similar to the choice of new points in the simulated annealing algorithm where we randomly pick a new candidate and probabilistically accept it after evaluating its set of performance functions. The heuristic of sequential application of GA and SA algorithm is based on the notion that GA based algorithms perform an excellent job of finding better candidates from a “mix” of the current population. However, the parameter space that they visit is rather constrained by the set of parameter values of the highly ranked individuals in the population. Passes of SA ensure a better stochastic search in the whole of the parameter space. Though, it might seem that the mutation step of GA is similar to a pure SA pass. However, the amount of effort that a standard GA implementation puts in mutation is quite less. Sequential runs of both of these algorithms provide the opportunity for new and (hopefully) attractive regions of the parameter space to be discovered (by SA) and the performance further improved amongst the candidate set (through GA).

This notion is similar to the concept of *ice ages* during the evolution of organisms on earth. During the warm periods, regular crossover and mutation of chromosomes occurred between the organisms leading to the development of better living creatures. However, during periods of extremely cold temperatures (*ice ages*), these superior organisms did not get the right environment to further their breed. Rather, some other inefficient (during the warm periods) organisms got the opportunity to mature, develop and compete against the superior organisms when the new “warm” periods ensued. We apply three such *epochs* (SA followed by GA) to generate our final pareto fronts.

3.2 Yield-Aware Pareto Surfaces

As discussed in earlier sections, if we ignore the process variations during the generation of the pareto surface, the points on the pareto may have poor yield values even though they represent the best set of performances that the circuit can achieve for the nominal design. This is primarily because the global optimizer used to generate the pareto works extremely hard to push (improve) the nominal design. In such cases, it often pushes the nominal design towards the tail in the performance distribution curve to get the non-dominated point on the pareto. However if we plot the performance distribution across process variation for that particular nominal design corresponding to a point on the pareto, we would often find the nominal point lying along the edges of the resulting distribution (Fig. 3). This means that once the circuit is fabricated using the particular set of design parameters corresponding to the point on the pareto front, most of the circuits would have performances

much inferior to co-ordinates of the pareto point in the performance space.

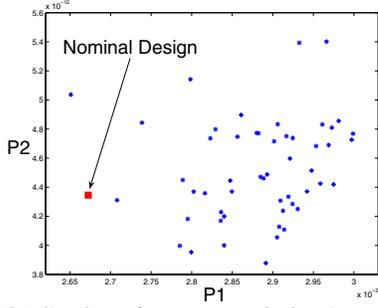


Figure 3: Distribution of process variation based Monte Carlo samples in the performance space for a design that was part of the pareto front. The nominal design lies at the edge of the distribution thereby resulting in very low yield if the corresponding co-ordinates of the nominal design are considered as performance specifications for the circuit.

This behavior was observed in almost all the points on the pareto front that we generated for a particular circuit. Figure 3 serves as the motivation behind our work for generation of yield-aware pareto surfaces.

Formally, we define a *yield-aware pareto surface with a yield ‘Y’* as a surface in the performance space generated using a set of non-dominated performance points such that for each of these points, there exists atleast one set of design parameters that results in yield greater than ‘Y’. The yield being measured by setting the co-ordinates of the points in the performance space as the circuit’s performance specification.

A simple way of generating these fronts is to keep track of the points on the yield curves (for a specific yield number) for each set of design parameters during a global optimization routine and then trying to improve the front generated by these points. This can be done by generating Monte Carlo samples across process variations at each design point. However, the method is computationally prohibitive since it requires Monte Carlo simulations for each candidate point in the parameter space. In the next section, we describe an approximate but fast methodology for computing Monte Carlo distributions that can be used to characterize the yield surface for a particular design point.

3.3 Approximate Monte Carlo Simulation

Monte Carlo analysis for a particular design point (that we call *anchor points*) is generally very expensive since it requires evaluation of the performance function for large number of input process parameter sets. Latin Hypercube (LH) sampling [4] is the most common method to reduce the number of evaluations required while still ensuring reasonable accuracy in computing the performance distribution function.

In our methodology, we generate sets of 10 LH samples using variational information about the *process* parameters and we repeat this step 10 times, thereby generating 100 input parameter sets. We then find the performance distribution function for the circuit candidate using SPICE simulations. Due to the nonlinear nature of the circuits, the resulting distributions often do not conform to any standard distribution *e.g.*, Normal, Weibull *etc.* We use a transformation function to convert these distributions to a normal distribution. Each dimension (x) in the performance space is transformed into a new space (y) using either one of the *transformation equations* shown in Eqn. 6 such that the distribution function matches that of a *normal distribution* with the same mean and variance as that for the given data in the transformed space. This is an opti-

mization problem where values of the variables (a, b, c, d) is to be determined such that our performance distribution data best *resembles* a Gaussian in the transformed space. The choice of these particular functions was made after experimentations which suggested that these equations provided the required fidelity for transforming arbitrary distributions, obtained during Monte Carlo simulations of the circuit, to a Normal distribution using minimal number of parameters. The optimization is performed using Brent-Powell based local optimization scheme [18]. The metric for *resemblance* is the sum of squared error between the distribution functions.

$$\begin{cases} y = a + b * (x + c)^d \\ y = a + b * \log(c * x + d) \end{cases} \quad (6)$$

Once the optimal transformation is obtained, we pick the set of 10 LH samples that best *resemble* the full performance distribution of the 100 LH samples in the transformed space. We call this set of 10 LH samples *optimal latin hypercube sampling* LHS_{opt} for the particular anchor point. We use LHS_{opt} to predict the distribution function of the circuit candidate in its performance space. For a design point close to the anchor point, we simulate the circuit for these 10 LHS_{opt} points. The performance values obtained through simulations are then transformed into a new space by applying Eqn. 6 using the values for parameters a, b, c and d obtained through optimization in the previous steps. In the new transformed space, the performance distribution functions are expected to match a Normal distribution as was the case for the anchor point. Using the mean, variance and correlation matrix of these points in the transformed space, large number of pseudo points are generated for the underlying Normal distribution. The pseudo points are directly generated using the distribution functions and do not require actual circuit simulation. Hence, this step is extremely fast. Once, we have these large number of points, we can compute the yield of the circuit for that particular design point for any arbitrary performance goal. Alternatively, we can also compute the performance values which would give us a particular yield number for the circuit candidate under consideration.

The set of optimum LH samples works fine for approximating the performance distribution function in regions of the parameter space (\mathbf{p}) that are *close* to the anchor point. We, therefore, need different LHS_{opt} for each region of the parameter space that we need to model.

3.4 Generating Yield-Aware Pareto

We first generate the nominal pareto front using the *new* non-dominated sorting based algorithm described in Sec. 3.1 (NSGA with ice ages). The primary pareto front thus generated is called the pareto front of order 1. A pareto front of order k is the front generated after removing all the points of pareto fronts of order 1 to $(k - 1)$.

We start with N equi-distant points on the k^{th} order front that we call *anchor points* (line 3 of Algorithm 1). k is usually chosen as 3 or 4 to give us a “soft” pareto front (line 2 of Algorithm 1). We then compute LHS_{opt} for these anchor points on the “soft” pareto front. Using LHS_{opt} , we compute the points in the performance space that would give a yield greater than some fixed quantity ‘Y’ (line 5 of Algorithm 1). The resulting set of points form a yield front. Figure 4 shows one such yield front computed for a particular design point with a yield of 80%.

We then try to improve this yield surface using local optimization techniques like Brent-Powell [18] (line 6 of Algorithm 1). At each yield evaluation for a particular design point during Brent-Powell optimization, we generate a set of points on the yield front (Figure 4). We use a non-dominated sorting algorithm to keep the yield points computed in the current evaluation and all previous

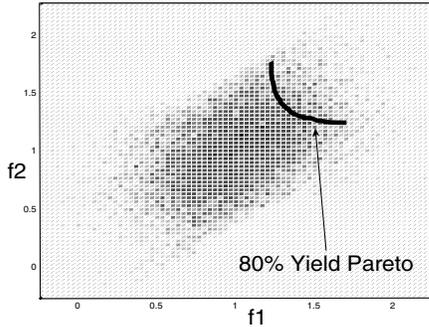


Figure 4: Points on the curve are equ-probability regions for an 80% yield corresponding to a particular design point. Yield based points for different design parameters can be combined together to compute the yield-aware pareto front.

evaluations as part of the evolving pareto. Since LHS_{opt} computes an *approximate* yield number, we do not aggressively search for incremental improvements in yield numbers that lead to points in the parameter space that are further away from the *anchor* point where the LHS_{opt} was generated. Once converged, the resulting pareto surface represents the best points in the performance space for which we could find a design such that the yield for that particular point is greater than the fixed number ‘Y’ for which the pareto was generated. Finally, we merge the yield fronts corresponding to all the N anchor points to obtain the final yield-aware pareto front for the particular circuit topology under consideration (line 8 of Algorithm 1). We could repeat this process for multiple yield numbers to generate multiple fronts – one each for a given yield number. Or, we could piggy back multiple yield number fronts on a single pareto front improvement step.

4. EXPERIMENTAL RESULTS

We have implemented the complete flow described in the previous sections. For verification of the results for the voltage controlled oscillator (VCO) that we have used as a test case, we employ a behavioral modeling infrastructure for efficiently computing the system level specifications of the phase locked loop (PLL) in which the VCO is used based on the algorithms presented in [13]. The tool uses Spectre [1] as its simulator for the computation of the performance values. The simulator chosen is arbitrary and our methodology is independent of the choice of the underlying function evaluator (circuit simulator). Since, most of the optimization routines are highly parallelizable, our tool used 4 machines with 3.2 GHz processors and 4GB of RAM each for evaluating the function values (\mathbf{f}) for input parameter sets (\mathbf{p}), simultaneously.

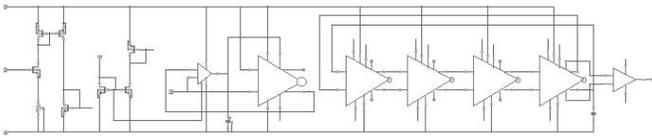


Figure 5: Circuit schematic of ring oscillator VCO circuit.

Fig. 5 shows the schematic of the ring oscillator VCO circuit whose pareto surface was to be generated. The performance specifications (\mathbf{f}) across which the pareto front was to be generated were phase noise and power (current, assuming constant supply voltage). There were some nominal specifications which the candidate circuits had to meet before they could be considered for being a point on the pareto. Table 1 lists the nominal specifications. The number of input design variables (\mathbf{p}) across which the optimal pareto front was generated was 12. Each parameter had a particular range

within which they could assume any value. These ranges were granulated based on design rules for manufacturability.

Table 1: Nominal specifications for the VCO circuit

Performance Variable	Goal Value
Min. o/p voltage	< 0.1
Max. o/p voltage	> 1.7
Min. o/p frequency	< 400MHz
Max. o/p frequency	> 500MHz
Linearity (o/p freq. Vs V_{ctrl})	> 0.5 & < 1.5

The pareto front generation process was started using 20 random seeds. The methodology presented in Section 3.1 was followed to generate the front. Fig. 6 shows the state of the pareto surface after the completion of each epoch (simulated annealing followed by genetic algorithm). Around 5,000 points were generated in each epoch. A regular circuit synthesis problem for this VCO using a commercial circuit synthesis tool [2] typically involved 4,000-5,000 circuit evaluations. Thus, the total effort required to generate the complete pareto front is equivalent to finding 3-4 points on the pareto front using the standard normal boundary intersection (NBI) method as described in Section 2.3.

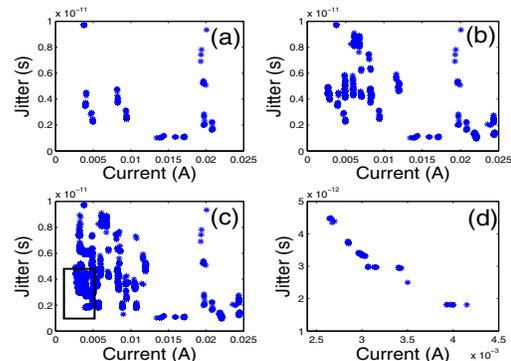


Figure 6: Generation of the pareto fronts across epochs – after (a) 1st epoch, (b) 2nd epoch and (c) 3rd epoch. Subfigure (d) represents the interesting region of the pareto front of order 1 after the pareto generation is complete (box region in epoch 3)

Once the nominal pareto front is generated, we pick a “soft” pareto (pareto front of order 4 (Section 3.4)). We pick equidistant points on this “soft” pareto as *anchor* points. For each *anchor* point, we perform a Monte Carlo analysis using the variational information for the process parameters. We use 10 sets of 10 LH samples to perform the approximate Monte Carlo analysis at each point. The distribution of samples in the performance space is then transformed to another space where distribution along each dimension resembles a Gaussian distribution. Fig. 7 shows the fitted distribution function for the two performance variables along with the fitting errors. The distribution functions in the non-transformed space were well behaved in this case resulting in a close match with the Gaussian distribution in the transformed space. However, we applied the transformations for other non-gaussian distributions. Even for these distributions, our optimization algorithm finds the set of parameters which transform the distribution to a Gaussian quite accurately (The transformations were tried only for distributions with unimodal pdfs).

After optimizing for the parameters for the transformation (Eqn. 6) at the *anchor* points, we picked the set of 10 LH samples that best represent the original distribution function in the transformed space. We use this as LHS_{opt} for computing the performance distribution functions for new, closeby points in the parameter space.

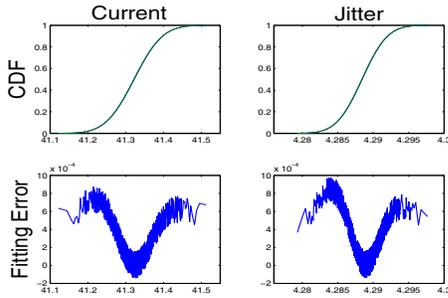


Figure 7: Plots showing the closeness of the distribution functions to the normal distribution in the transformed space.

Thus, for new design points, we simulate the circuit for the 10 LHS_{opt} values; transform the performance space using parameters obtained earlier for the *anchor points*. Using the mean, variance and correlation coefficient metric for these points in the transformed space, we sample the Gaussian distribution with these metrics to generate pseudo Monte Carlo sample points. This step is extremely fast since no circuit simulation is required. We then find pareto points on this distribution that would result in a given fixed yield number. These points are then transformed back to the real performance space as candidates for the pareto front. The front is then improved using a simple Brent-Powell [18] local optimizer which simulates the circuit for LHS_{opt} (process parameter) points repeatedly at each new design points. Fig. 8 shows the yield-aware pareto curves corresponding to different yield number after the local optimization step for the VCO circuit.

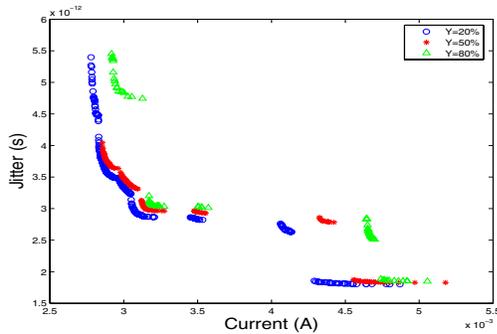


Figure 8: Pareto fronts for 20%, 50% and 80% yield.

The pareto curve for the VCO corresponding to a yield of 50% was used for doing hierarchical synthesis of a PLL using ideas similar to those in [13] and synthesis tools from [1]. The pareto was modeled using a piecewise polynomial equation in VerilogA. The PLL was synthesized for a set of overall jitter, power and lock time constraints. The VCO circuit was synthesized by doing a look-up for the design point that generated the nearest sampled pareto point closest to the optimum pareto point. The yield for that final synthesized design, using a full Monte Carlo analysis, came out to be 47.3%, which is in close agreement to the 50% number that we were expecting.

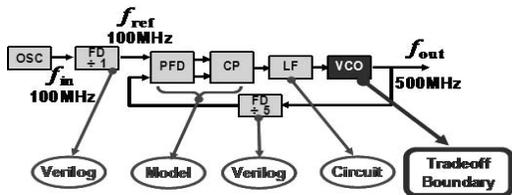


Figure 9: PLL simulation infrastructure. VCO represented by the pareto curve. Loop filter as real devices. All other blocks assumed fixed and represented using behavioral models.

5. CONCLUSIONS

We have presented a novel methodology for generating yield-aware pareto surfaces. The method includes a new, nominal pareto generation algorithm. It then uses efficient local latin-hypercube sampling resulting in fast Monte Carlo analysis for generation of yield-aware pareto fronts. The complete algorithm requires simulation time close to 5-6 full circuit synthesis runs and hence is extremely efficient computationally. A simple experiment to synthesize PLL hierarchically, with a targeted yield, suggests the practicality of the approach. These pareto surfaces can find applications in hierarchical synthesis, topology selection and other yield-aware design applications.

6. ACKNOWLEDGMENTS

This work was funded in part by the MARCO/DARPA Center for Circuits & Systems Solutions (C2S2) and the Pittsburgh Technology Collaborative.

7. REFERENCES

- [1] Spectre simulator from Cadence. <http://www.cadence.com>.
- [2] *High-performance CMOS-amplifier design uses front-to-back analog flow*. A.H.Shah and S.Dugalleix and F.Lemery, EDN, 2002.
- [3] E.Ochotta, R.Rutenbar, and L.Carley. Synthesis of high performance analog circuits in astrx/oblx. *TCAD*, 15:273–294, 1996.
- [4] K. T. Fang, K. Fang, and L. Runze. *Design and Modeling for Computer Experiments*. CRC Press, October 2005.
- [5] R. Harjani and J. Shao. Feasibility and performance region modeling of analog and digital circuits. In *Jrnl. of Analog Integrated Circuits and Signal Processing*, pages 23–43, June 1996.
- [6] I.Das and J.E.Dennis. Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems. In *SIAM Journal on Optimization*, pages 631–657, 1998.
- [7] K.Deb, A.Pratap, S.Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. In *Trans. on Evolutionary Computation*, number 2, pages 182–197, April 2002.
- [8] K.Kundert. Predicting the phase noise and jitter of PLL based frequency synthesizers. <http://www.designers-guide.org>.
- [9] M. W. Kruiskamp and D. Leenaerts. Darwin: Cmos opamp synthesis by means of a genetic algorithm. In *DAC*, pages 433–438, 1995.
- [10] H. Liu, A. Singhee, R. Rutenbar, and L. Carley. Remembrance of circuits past: Macromodeling by data mining in large analog design spaces. In *DAC*, pages 437–442, 2002.
- [11] S.Kirkpatrick, C.D.Gelatt, and M.P.Vecchi. Optimization by simulated annealing. *Science*, (671-680), 1983.
- [12] S.K.Tiwary and R.A.Rutenbar. Scalable trajectory methods for on-demand analog macromodel extraction. In *DAC*, 2005.
- [13] S.K.Tiwary, S.Velu, R.A.Rutenbar, and T.Mukherjee. Pareto optimal modeling for efficient PLL optimization. In *Modeling and Simulation of Microsystems, Nanotech*, pages 195–198, 2004.
- [14] G. Stehr, H. Graeb, and K. Antreich. Feasibility regions and their significance to the hierarchical optimization of analog and mixed-signal systems. In *Intl. Series of Numerical Mathematics*, pages 167–184, 2003.
- [15] G. Stehr, H. Graeb, and K. Antreich. Performance tradeoff analysis of analog circuit by normal boundary intersection. In *DAC*, pages 958–963, 2003.
- [16] T.K.Ogawa and K.Kundert. VCO jitter simulation and its comparison with measurements. In *ASP-DAC*, Jan. 1999.
- [17] D. A. V. Veldhuizen. *Multiobjective Evolutionary Algorithms: classifications, analysis, and new innovations*. PhD thesis, Air Force Institute of Technology, WrightPatterson AFB, USA., June 1999.
- [18] W.H.Press, S.S.Teukolsky, W.T.Vetterling, and B.P.Flannery. *Numerical Recipes in C++ : The Art of Scientific Computing*. Cambridge University Press, 2002.
- [19] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. In *Evolutionary Computation*, pages 173–195, 2000.