Efficient Multiobjective Synthesis of Analog Circuits using Hierarchical Pareto–optimal Performance Hypersurfaces

Tom Eeckelaert Trent McConaghy Georges Gielen
Katholieke Universiteit Leuven,
Department of Electrical Engineering, ESAT–MICAS
Kasteelpark Arenberg 10, B–3001 Leuven
Tom.Eeckelaert@esat.kuleuven.ac.be

Abstract

An efficient methodology is presented to generate the Pareto–optimal hypersurface of the performance space of a complete mixed–signal electronic system. This Pareto–optimal front offers the designer access to all optimal design solutions: starting from the performance specifications, a satisfactory point can a posteriori be selected on the hypersurface which immediately determines the final design parameters. Fast execution is guaranteed by using multi–objective evolutionary optimization techniques and hierachical decomposition. The presented method takes advantage of the Pareto hypersurfaces of the subblocks to generate the overall Pareto front. The hierarchical approach combines behavioral simulation with behavioral models at the higher levels, with SPICE simulations with transistor–level accuracy at the lowest level. Storing the performance data of all subblocks enables reuse for other systems later on.

1. Introduction

To keep up with the drastically increasing complexity of todays electronic systems, a lot of research has been done in the field of analog design automation [2]. A lot of prototype CAD tools resulted from the research [6]: IDAC , OASYS, OPTIMAN, AMGIE, ASTRX/OBLX, ANACONDA.

The most successful tools use optimization techniques to search for the optimal design solution according to the given specifications. Most of these tools however are limited to rather small building blocks. Because of the increasing complexity of electronic systems, there is a shift in the analog/mixed–signal research towards design methodologies for higher hierarchical design levels. By hierarchically decomposing the system in smaller less complex building blocks, dedicated simulators can be used at each hierarchical level. Examples are SD–OPT [5] and DAISY [1] for the high–level design of Sigma–Delta converters. However, these methodologies simulate at a higher hierarchical level without taking into account lower–level simulations. When enclosing simulations of lower–level building blocks, the design space explodes and these methodologies will fail to find an optimum solution. Moreover, because the majority of the existing tools is based on the optimization of one single objective function, the optimum that is found is depending on the a–priori choice of the weighting coefficients used to combine the different objectives.

Therefore, the methodology presented in this paper is based on the hierarchical use of powerful Multiobjective Evolutionary Algorithms (MOEA) [7]:

- Through their inherent parallelism, these EA are well suited in handling the hierarchical decomposition of large systems.
- These multiobjective techniques are able to find the Pareto–optimal trade–off hypersurface between the different objective functions, in one single optimization run.
- To cope with the disadvantage of the exploding design space when including lower–level simulations of the building blocks, we will show that lower–level Pareto–fronts can be used to store the hierarchical performance information and make the whole approach very efficient.
- This also makes it possible to simulate each hierarchical building block by the best suitable simulator. This way, the use of SPICE–like simulators at the transistor level preserves the accuracy that these simulators provide.
- The generated Pareto–optimal data points at each level can be stored and modeled into a continuous Pareto–hypersurface. The stored data of the subblocks can be reused in future other systems without the need of resimulation.

The paper is organized as follows. Section 2 describes some definitions about multi–objective optimization and the algorithm that was selected for use in the presented methodology. In section 3, the methodology is explained and choices in the

1530-1591/05 $20.00 © 2005 IEEE

implementation are discussed. Next, in section 4, the method is illustrated by generating the Pareto–optimal performance–hypersurface of a $\Delta\Sigma$ converter is generated with and without the use of lower–level Pareto–optimal surfaces, and the two methods are compared. The final section, gives conclusions about the research.

2. Multi–objective Optimization

In this section first some basic definitions about *multi–objective optimization* (MOO) and *Pareto optimality* are given. Next, the Strength Pareto Evolutionary Algorithm (SPEA), which was used to implement the presented methodology, is briefly discussed. discussed.

2.1. Basic Definitions

In a general **multi–objective optimization problem** the goal is to find optimal values for the components of a vector function $f(\mathbf{x})$ subject to some constraints $e(\mathbf{x})$:

$$\begin{array}{ll} \text{minimize/maximize} & \mathbf{y} = f(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x})) \\ \text{subject to} & e(\mathbf{x}) = e_1(\mathbf{x}), e_2(\mathbf{x}), \ldots, e_k(\mathbf{x}) \leq 0, \\ \text{with} & \mathbf{x} = (x_1, x_2, \ldots, x_m) \in X, \\ & \mathbf{y} = (y_1, y_2, \ldots, y_n) \in Y \end{array} \tag{1}$$

where $\mathbf{x}$ is the *decision vector*, $\mathbf{y}$ is the *objective vector*, X is called the *decision space* and Y is called the *objective space*. The k constraints $e(\mathbf{x}) \leq 0$, set restrictions on the decision space.
The multiple objective functions almost always conflict with one another. Therefore, for multi–objective problems, **Pareto optimality** is used. First the concept of *Pareto dominance* should be clear. For a maximization problem, a vector $\mathbf{u} = (u_1, \ldots, u_m)$ dominates a vector $\mathbf{v} = (v_1, \ldots, v_m)$ **($\mathbf{u} \succ \mathbf{v}$)** if and only if

$$\begin{array}{c} \forall i \in \{1, 2, \ldots, n\} : f_i(\mathbf{u}) \geq f_i(\mathbf{v}) \\ \text{and} \\ \exists j \in \{1, 2, \ldots, n\} : f_j(\mathbf{u}) > f_j(\mathbf{v}). \end{array} \tag{2}$$

Additionally, $\mathbf{u}$ covers $\mathbf{v}$ **($\mathbf{u} \succeq \mathbf{v}$)** if and only if $\mathbf{u} \succ \mathbf{v}$ or $\mathbf{u} = \mathbf{v}$. A decision vector $\mathbf{x} \in X$ is now said to be Pareto–optimal if and only if there exists no $\mathbf{z} \in X$ for which $f(\mathbf{z})$ dominates $f(\mathbf{x})$.

2.2. Strength Pareto Evolutionary Algorithms

Evolutionary algorithms are stochastic optimization methods designed by analogy with natural evolution [8]. Starting from a set of solutions (individuals) subsequent sets are composed by means of selection and variation.

We integrated the existing Strength Pareto Evolutionary Algorithm (SPEA) [10] in the implementation of our methodology.
Characteristics of the SPEA are:

- The Pareto–optimal solutions found in each iterative optimization run, are stored in an external set. After each run, the external set is updated by comparison with the newly found Pareto–optimal solutions. This way, Pareto–optimal solutions don't get lost.
- The important fitness assignment step during selection uses the concept of Pareto dominance (2). Only the externally stored Pareto set (size N_p) determines the fitness of an individual. Domination by individuals in the population (size N) is irrelevant. In this step, each of the individuals in the population have to be compared with the individuals in the external Pareto set, for all objectives (M). This is the most time consuming step, resulting in a computing complexity of $O(MNN_p)$.
- To reduce the size of the Pareto–optimal set, clustering is used in the original implementation. In particular, the average linkage method. At the first optimization step, each individual represents one cluster on its own. Then, in each step two clusters of individuals are chosen, using the nearest neighbor criterion, to merge into one cluster. After that, out of each cluster one individual is chosen to represent the cluster in the external set.
  In our opinion it is better not to loose any Pareto–optimal solutions, so we didn't use the clustering technique. A disadvantage to this is the increasing individuals in the external set ($O(MNN_p)$), but there is a gain from the eliminated clustering step.

3. System design space Exploration

In this section, first hierarchical decomposition and behavioral performance modeling, and the relevance to our methodology is discussed. In the second subsection it is discussed how the hierarchical performance information of the lower–level building blocks can be included in the generation of the Pareto–optimal hypersurface at a higher level.

3.1. Hierarchical decomposition and behavioral performance modeling

To cope with the increasing complexity of current analog systems, the systems have to be decomposed into smaller less complex subsystems. When the subsystems are still to complex to design, a second decomposition is performed. This **systematic, hierarchical decomposition** can go on until all subblocks are manageable for design. If at some hierarchical level a subblock has already been designed in the past, then the decomposition of the branch which comprises that block

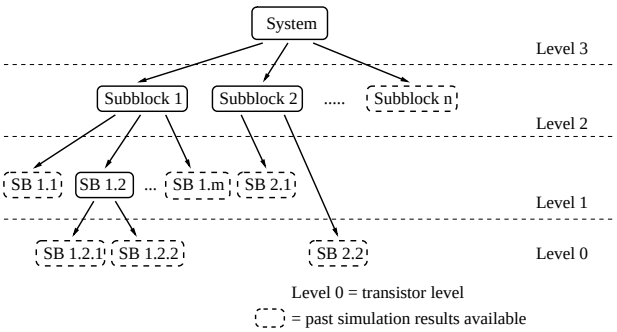

Figure 1. Decomposition of a complex system. Ending nodes in the branches represent subsystems for which no lower–level performance information is needed.

will end at the hierarchical level of that block. Figure 1 illustrates this. When at some branch in the decomposition, the system has never before been designed, then the lowest hierarchical level is the transistor level. At transistor level, the block can be simulated by a SPICE–like simulator to extract its performance behavior. If on the other hand a subsystem has already been used for another system, the past simulation results can be reused. The subblock is then said to be used as intellectual property (IP).
After decomposition of the system, at each higher level, **behavioral models** have to be used to combine the performance information of lower levels. A lot of research has been published about behavioral performance modeling, and the majority of them is already implemented in some kind of high–level simulator, e.g. VHDL–AMS. Our methodology provides the capability of using all possible simulators for a building block. The system is defined to the tool by means of simple description files. For each subsystem a description file has to be composed containing information about the simulator to be used, the design variables, the subsystems from which it uses performance information, and which performance variables have to be extracted from the output file of the simulator.

3.2. Modeling and reusing lower–level Pareto surfaces

As was mentioned in the previous subsection, performance trade–off information from previous simulation runs can be stored and reused when a subblock is used again in another system. A look–up table could be used, but it would even be best to find a mathematical model for the hypersurface from which information can be extracted without performing a simulation. In this way, only the very first time a subblock is used, transistor–level simulations are needed to create the data and generate the Pareto hypersurface, which can then by used for all later hierarchical synthesis runs in which that subblock is used. Advantages of this approach are:

- Algorithm speedup by saving time in the absence of lower–level simulation runs: either a lookup table used instead, or the mathematical model for the Pareto–optimal performance trade-off hypersurface has to be evaluated.
- When no lower–level performance information is used, the design variables of the high–level system, as well as the design variables from all the lower–level building blocks have to be determined during the optimization. This way the design space gets very large resulting in a huge population set (large N) and a huge externally stored set (large N_p) needed to sample that space. Through the use of lower–level Pareto–optimal performance hypersurfaces, the computing complexity ($O(MNN_p)$) of the algorithm can thus be reduced.

The following derivation however shows that attention should be paid to the way the lower–level Pareto–optimal surfaces are generated:
In a hierarchical design, performance characteristics from a lower level are used as design variables at the higher level. Suppose for the sake of simplicity a minimization problem with a two–dimensional decision space (x_1, x_2) and a two–dimensional objective space $(f_1(x_1,x_2), f_2(x_1,x_2))$. If it can be shown that there exists a dominated point (definitions in section 2.1) in the decision space which contributes to the Pareto–optimal curve in the objective space, then it is proven that essential information is lost when only the non–dominated points (Pareto–optimal 2D–curve) in the decision space are used.
Suppose a point (x_1^*, x_2^*) in the decision space is dominated

$$\Rightarrow \exists (x_1^{'}, x_2^{'}) \mid x_1^{'} \leq x_1^* \ \& \ x_2^{'} \leq x_2^* \tag{3}$$

For this point not to contribute to a Pareto–optimal objective point, $(f_1^*, f_2^*) = (f_1(x_1^*, x_2^*), f_2(x_1^*, x_2^*))$ should be dominated. So,

$$\exists (f_1^{'}, f_2^{'}) \mid f_1^{'} \leq f_1^* \ \& \ f_2^{'} \leq f_2^* \tag{4}$$

Combining (3) and (4) results in the 2 following requirements:

$$\begin{aligned} 1)\ & \left.\frac{\delta f_1}{\delta x_1}\right|_{x_1^*} > 0 \ \& \ \left.\frac{\delta f_1}{\delta x_2}\right|_{x_2^*} > 0 \\ & \text{or if } \left.\frac{\delta f_1}{\delta x_1}\right|_{x_1^*} < 0 \Rightarrow \left\|\left.\frac{\delta f_1}{\delta x_1}\right|_{x_1^*}\right\| < \left\|\left.\frac{\delta f_1}{\delta x_2}\right|_{x_2^*}\right\| \\ & \text{or if } \left.\frac{\delta f_1}{\delta x_2}\right|_{x_2^*} < 0 \Rightarrow \left\|\left.\frac{\delta f_1}{\delta x_2}\right|_{x_2^*}\right\| < \left\|\left.\frac{\delta f_1}{\delta x_1}\right|_{x_1^*}\right\| \end{aligned} \tag{5}$$

$$\begin{aligned} 2)\ & \left.\frac{\delta f_2}{\delta x_1}\right|_{x_1^*} > 0 \ \& \ \left.\frac{\delta f_2}{\delta x_2}\right|_{x_2^*} > 0 \\ & \text{or if } \left.\frac{\delta f_2}{\delta x_1}\right|_{x_1^*} < 0 \Rightarrow \left\|\left.\frac{\delta f_2}{\delta x_1}\right|_{x_1^*}\right\| < \left\|\left.\frac{\delta f_2}{\delta x_2}\right|_{x_2^*}\right\| \\ & \text{or if } \left.\frac{\delta f_2}{\delta x_2}\right|_{x_2^*} < 0 \Rightarrow \left\|\left.\frac{\delta f_2}{\delta x_2}\right|_{x_2^*}\right\| < \left\|\left.\frac{\delta f_2}{\delta x_1}\right|_{x_1^*}\right\| \end{aligned} \tag{6}$$

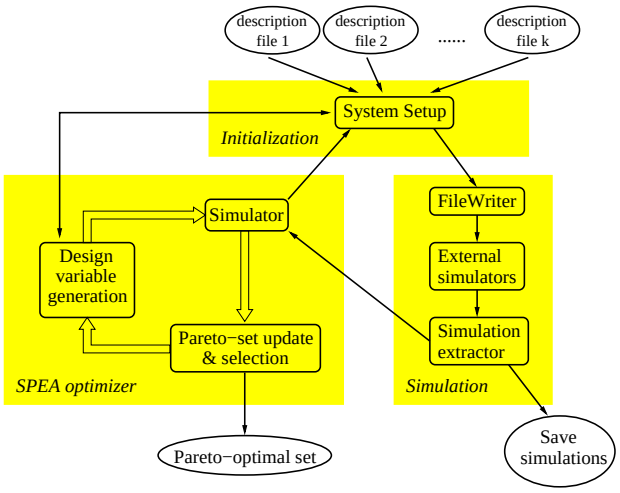

Figure 2. Algorithm framework.

If (5) and (6) are not met, then the Pareto optimal performance trade–off curve (2D–hypersurface) contains points corresponding with dominated points in the performance space of a lower hierarchical building block.
So, in order to use lower–level Pareto–optimal surfaces, a lower–level solution moving towards its Pareto–optimal surface, should result in a movement of its corresponding higher–level solution towards the higher–level Pareto-optimal surface. As shown in the above, for both a minimization problem at the two hierarchical levels, the derivatives of the objective functions considered should be positive. For the performance parameters 'power' and 'area' for example, which are most typically used objectives in analog synthesis, this requirement is obviously met. As an other example, for the calculation of the Signal-to-Noise Ratio (SNR), calculated with the high–level simulator depicted in [4], only the position of poles, zeros and the gain of the composing integrators is used from the lower level. Because the relationship between the gain of the different integrators and the SNR isn't as the requirements above state, gain should not be included as an objective at the integrator transistor level; it can be handled as a constraint. If on the other hand certain lower–level performance parameters do not influence higher–level objectives, then they can be included in the list of lower–level objectives as well. In the example of section 4 it will be shown that for performance parameters at higher and lower hierarchical levels that meet the requirements stated in the above derivation, significant benefit can be derived from using the lower–level Pareto–optimal surface instead of lower–level simulations.

3.3. The optimization framework

The hierarchical multi–objective synthesis method has been prototyped in a software framework. The framework contains three major fragments (Figure 2). In the **initialization part**, the complete hierarchically decomposed system is set up. As described in section 3.1, a description file has to

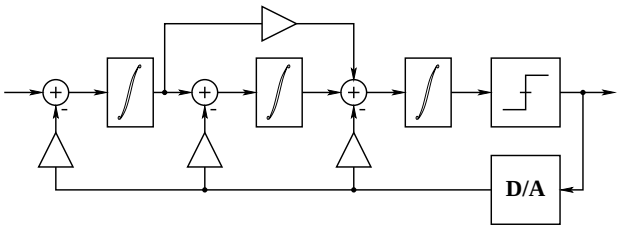

Figure 3. A single–loop 3^{th} order Delta–Sigma modulator architecture.

be given for the complete system. This description file is then analyzed in the *System Setup* module and description files for the subsystems are systematically included during setup. The second major part, the **simulation part**, consists of a module (*FileWriter*) that uses the information extracted from the description files to generate a new input file for each externally used simulator. The generation of such an input file is based on a template file which contains variables for which values have to be substituted. These values are extracted from lower–level building blocks or given as design parameters during simulation startup. After writing the input files, the different external simulators are executed. When they are finished, a *Simulation Extractor* module extracts the needed performance and constraint variable values from the simulator outputs. All simulations are stored in files for later use (see sections 3.1 and 3.2). If on the other hand a flag is set to use previously stored simulations or a set of Pareto–optimal solutions, then no file is written and no external simulators are called. The Pareto–optimal solutions are stored in an ordered set to make it possible to mutate to nearby optimal solutions. The third fragment, implements the **SPEA optimizer** as discussed in section 2.2. In each optimization run, the design variables which can not be extracted from lower levels are generated by the MOEA for each subsystem in the hierarchy. Values for the design variables are chosen within their boundaries given in the description files. The complete systems performance variables are then determined by a call to the *simulator*. The externally stored Pareto set is updated and by assigning fitness values, individuals are selected to be used for mating (section 2.2) and generating the next set of design variables.

4. Experimental results

In this section we will illustrate the presented methodology by generating the Pareto–optimal performance–hypersurface of a third–order single–loop continuous–time delta–sigma modulator (Figure 3). We chose a simple hierarchical decomposition of one subblock for the $\Delta\Sigma$ converter: the integrator subblock is represented by a highspeed folded–cascode OTA (Figure 4), implemented in a 0.25 μm CMOS technology. For the high–level simulation of the $\Delta\Sigma$, the Model of Computation from [4] was implemented in the MATLAB^{TM} programming language. At transistor–level, HSpice was used.

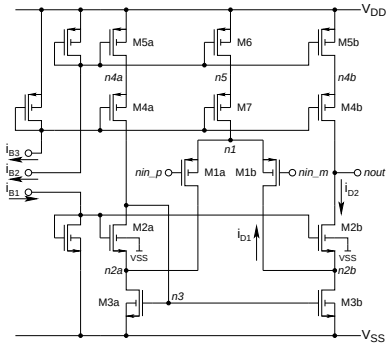

Figure 4. CMOS folded–cascode operational transconductance amplifier.

The information contained in the description file of the $\Delta\Sigma$ modulator is given below:

- The performances chosen for the exploration are power consumption [W], silicon area [m^2], and signal–to–noise ratio (SNR) [dB].
- The design variables at the modulator level are the over–sampling ratio (OSR), the input–signal frequency (f_0), the sampling frequency (f_s), and the input amplitude (A_0). Also the poles, zeros and gain of the integrators are needed but they get extracted from the lower level during the simulation.
- No constraint variables were set for the modulator.

For the description file of the OTA, the following information is given:

- The performances are power [W], area [m^2], UGF [Hz], phase margin (PM) [$^\circ$], noise density [V^2/Hz], slew rate [V/s]. They can all be extracted from the HSpice output file.
- The design variables include all independent gate-source and drain-source voltages, transistor widths (operating–point driven formulation [3]), and biasing currents, and the integrating capacitance. The supply voltages are fixed to Vdd $= 1.25V$ and Vss $= -1.25V$.
- Constraints are drain–source and gate–source voltages to keep the transistors into saturation. Also the poles and zeros are given in the list of constraints, but without constraints, just to make sure they are determined for use at $\Delta\Sigma$ level.

To demonstrate the advantage of using hierarchical performance information in the form of Pareto–optimal hypersurfaces, two methods are compared here. One (**A**), where the design space of the $\Delta\Sigma$ modulator combines all design spaces of all the integrators and its own, and another method (**B**) where first a Pareto–optimal solution set is generated for the integrators and then this set is used during optimization of the $\Delta\Sigma$ level.

Table 1. Comparison of the integrators Pareto–optimal fronts using $\mathcal{C}$ metrics.

	Int_1	Int_2	Int_3
Int_1	$\mathcal{C}_{\succeq} = 1.00$ $\mathcal{C}_{\succ} = 0.00$	$\mathcal{C}_{\succeq} = 0.12$ $\mathcal{C}_{\succ} = 0.10$	$\mathcal{C}_{\succeq} = 0.16$ $\mathcal{C}_{\succ} = 0.14$
Int_2	$\mathcal{C}_{\succeq} = 0.33$ $\mathcal{C}_{\succ} = 0.31$	$\mathcal{C}_{\succeq} = 1.00$ $\mathcal{C}_{\succ} = 0.00$	$\mathcal{C}_{\succeq} = 0.26$ $\mathcal{C}_{\succ} = 0.25$
Int_3	$\mathcal{C}_{\succeq} = 0.20$ $\mathcal{C}_{\succ} = 0.18$	$\mathcal{C}_{\succeq} = 0.13$ $\mathcal{C}_{\succ} = 0.11$	$\mathcal{C}_{\succeq} = 1.00$ $\mathcal{C}_{\succ} = 0.00$

To compare the two solution sets A and B, we selected the Coverage metric $\mathcal{C}(A,B)$ from [9]. This metric represents the amount of individuals in population B that are dominated by an individual in population A divided by the total amount of individuals in B. So, $\mathcal{C}(A,B) = 1$ means that all individuals in B are dominated by or equal to individuals in A. $\mathcal{C}(A,B) = 0$ means that none of the individuals in B are dominated. In accordance to the definitions from [11], the following relations between objective vectors (z^i) are used: $z^A \in A$ strictly dominates $z^B \in B$ ($z^A \succ z^B$) if all objective values in z^A are better than in z^B, z^A dominates z^B ($z^A \succeq z^B$) if all objective values in z^A are better or equal than in z^B. Using these relations, a distinction is made between $\mathcal{C}_{\succ}(A,B)$ and $\mathcal{C}_{\succeq}(A,B)$. For example, if $0 < \mathcal{C}_{\succeq}(A,B) < 1$ and $0 < \mathcal{C}_{\succeq}(B,A) < 1$ then both sets contain solutions which dominate solutions in the other set. But if for the same sets $\mathcal{C}_{\succ}(A,B) = 0$ and $\mathcal{C}_{\succ}(B,A) > 0$ it can be concluded that only B contains solutions which strictly dominate solutions in A, thus B is better then A in relation to approximating the Pareto–optimal front.

First an optimization was started to find an approximation of the Pareto–optimal set for all 3 integrators. Keep in mind that this should normally only be done once, because the resulting set can be used for all three integrators and for future design as well. This is only done here to have different simulation data when we use method **A**. Table 1 shows that the fronts don't dominate each other. Method **A** consists of high–level optimization while using all generated simulation results stored while generating the lower–level Pareto–optimal surfaces. Method **B** uses only final the Pareto–optimal sets of the integrators.

The total amount of optimization generations at $\Delta\Sigma$ level was limited to 100. While a population limit of 100 individuals was set, the external Pareto set was not limited. For method **A**, a total of about 6300 good individuals were generated and an other 12200 individuals were discarded because they didn't meet the specs (e.g. transistors in saturation). For method **B**, a total of about 5400 good individuals were generated and about 12000 were discarded.

Figure 5 compares the solution sets from method **A** and **B** during optimization. The curves in the figure show the values of the four coverage metrics ($\mathcal{C}_{\succeq}(A,B)$, $\mathcal{C}_{\succ}(A,B)$, $\mathcal{C}_{\succeq}(B,A)$, $\mathcal{C}_{\succ}(B,A)$) as defined before. The results are better then expected actually. We wanted to show that the use of only the

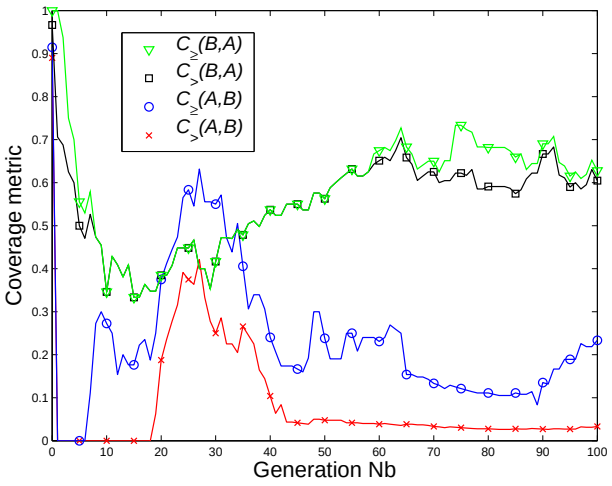

Figure 5. Comparison of $\Delta\Sigma$ Pareto–optimal fronts generated by method A and B, at different generations in the optimization process starting from the initial population, using the coverage metric $\mathcal{C}$.

Pareto–optimal performance information doesn't abundantly deteriorate the quality of the Pareto–optimal set (solution set of method **B** shouldn't be strictly dominated by the solution set of method **A**). As can be concluded from figure 5, method **B** even results in solution sets that dominate the solution sets of method **A**. The randomly generated initial population generated in method **B** is already slightly dominating the initial population in method **A**. This can be explained by the fact that the 'power' and 'area' performance parameters meet the requirements depicted in subsection 3.2, and because in method **B** only Pareto–optimal lower–level values are used the higher–level values of 'power' and 'area' are already Pareto–optimal.
Over the 100 generations, the solution set from method **B** almost always dominates the solution set of method **A**. For later generation however $\mathcal{C}_{\succeq}(A,B)$ and $\mathcal{C}_{\succ}(A,B)$ curves are slightly rising again and $\mathcal{C}_{\succeq}(B,A)$ and $\mathcal{C}_{\succ}(B,A)$ curves are decreasing. So, the solution sets of method **A** start to cath up with the solution sets of method **B**. Eventually both solutions should be at least as good because all the Pareto–optimal lower–level solutions used in method **B** are also contained in the lower–level solution set used by **A**. This only takes more generation runs.

5. Conclusions

In this paper, an efficient methodology was presented for the generation of Pareto–optimal performance tradeoff surfaces of complex systems. The objective was to use hierarchical performance information from lower hierarchical subblocks in the form of Pareto–optimal performance hypersurfaces. It was shown that under certain requirements, these lower–level Pareto–optimal fronts can indeed be used to extract performance information of the subblocks avoiding time–consuming transistor–level simulations. Advantages of this approach are the limited dimension of the high–level design space, the storage memory reduction by modeling the Pareto–optimal performance trade–off surface, the significant speedup of the optimization algorithm, while the quality of the eventually generated high–level performance trade–off is maintained. The surfaces generated by this methodology can be used by an analog designer to a priori choose a performance configuration that best meets the given specifications and hereby immediately determines the values of all design parameters of all the subsystems.

References

[1] K. Francken and G. Gielen. A high–level simulation and synthesis environment for $\Delta\Sigma$ modulators. *IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems*, 22(8):10049–1061, Aug. 2003.

[2] G. Gielen and R. A. Rutenbar. Computer–aided design of analog and mixed–signal integrated circuits. *Proceedings of the IEEE*, 88(12):1825–1852, Dec. 2000.

[3] F. Leyn, G. Gielen, and W. Sansen. An Efficient DC Root Solving Algorithm with Guaranteed Convergence for Analog Intergrated CMOS Circuits. In *Proceedings IEEE/ACM International Conference on Computer Aided Design*, pages 304–307, 1998.

[4] E. Martens and G. Gielen. A Model of Computation for Continuous–Time $\Delta\Sigma$ Modulators. In *Proceedings Design Automation and Test in Europe Conference*, pages 162–167, 2003.

[5] F. Medeiro, B. Pérez-Verdú, A. Rodríguez-Vázquez, and J. L. Huertas. A vertically–integrated tool for automated design of $\Sigma\Delta$ modulators. *IEEE Journal of Solid–State Circuits*, 30(7):762–772, July 1995.

[6] R. A. Rutenbar, G. Gielen, and B. A. A. Antao. *Computer–Aided Design of Analog Integrated Circuits and System*. John Wiley & Sons, 2002.

[7] D. A. Van Veldhuizen. *Multiobjective Evolutionary Algorithms: classifications, analysis, and new innovations*. PhD thesis, Air Force Institute of Technology, Wright–Patterson AFB, USA, June 1999.

[8] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, Swiss Federal Institute of Technology, Zurich, Switzerland, 1999.

[9] E. Zitzler, K. Deb, and L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, 2000.

[10] E. Zitzler and L. Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *Transactions on Evolutionary Computation*, 3(4):257–271, Nov. 1999.

[11] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *Transactions on Evolutionary Computation*, 7(2):117–132, Apr. 2003.