Large-Scale Nonlinear Optimization
in Circuit Tuning

Andreas Wächter, Chandu Visweswariah, Andrew R. Conn

IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598, USA

Abstract

Circuit tuning is an important task in the design of custom digital integrated circuits such as high-performance microprocessors. The goal is to improve certain aspects of the circuit, such as speed, area, or power, by optimally choosing the sizes of the transistors. This task can be formulated as a large-scale nonlinear, nonconvex optimization problem, where function values and derivatives are obtained by simulation of individual gates. This application offers an excellent example of a nonlinear optimization problem, for which it is very desirable to increase the size of the problems that can be solved in a reasonable amount of time. In this paper we describe the mathematical formulation of this problem and the implementation of a circuit tuning tool. We demonstrate how the integration of a novel state-of-the-art interior point algorithm for nonlinear programming led to considerable improvement in efficiency and robustness. Particularly, as will be demonstrated with numerical results, the new approach has great potential for parallel and distributed computing.

Key words: large-scale nonlinear nonconvex programming, circuit tuning, transistor sizing, interior point method, filter method, line search

1 Introduction

Nonlinear programming (NLP), i.e., the numerical solution of optimization problems involving sufficiently smooth, but possibly nonconvex, objective and constraint functions, has been studied extensively since the 1960s. On the other hand, methods for linear programming have been *used* throughout engineering and operations research since the second World War. It has been relatively recent that the capabilities and computer implementations of algorithms for NLP allowed one to address practical nonlinear engineering applications with moderately accurate models. Thus one is now able to solve problems of the order of tens of thousands of variables and nonlinear constraints, or larger,

while retaining an acceptable level of robustness. Nevertheless, there is an inexorable demand for solving larger and more complex problems. In some cases it is essential to solve the problems in real time and/or repeatedly. Thus there is always concern for the computational bottlenecks of a particular problem. These could frequently be in the function evaluation (in extreme cases those might involve laboratory experiments, such as raising of a cereal crop, or the numerical simulation of very complex systems), or in the numerical analysis computations (such as solving very large linear systems), or in a combination of these.

The computational bottleneck when solving a nonlinear optimization problem may be due to the sheer size of the problem. For example, when the problem formulation comes from the discretization of partial differential equations (PDEs), the number of variables and constraints is usually proportional to the number of grid points, which in turn is often chosen as large as possible in order to obtain an accurate representation of the continuous PDE system [1]. In such an application, the objective and constraint functions are usually available in analytical form, so that their values and (first and second) derivatives can be computed quickly.

On the other extreme, even without waiting for a crop to mature, the main computational effort might lie in the computation of function values, e.g., when they are obtained by computer simulation of complex systems, and their values may be numerically noisy. In this case, the optimization problem could be relatively small (say, up to 100 variables) and derivative values of those functions may not be available, for example, because of legacy code. If possible, the derivatives should always be computed and used in optimization, however in those cases where they are not available and the problem is not too large, derivative-free optimization methods [2] are often extremely useful. In some cases other methods, such as simulated annealing or genetic algorithms may be the only suitable technique (for example if the problem is not continuous, highly nonlinear and is both noisy and uses discrete variables). However, we feel that one should use these only as a last resort when all else fails since they are both unreliable and very slowly converging.

Circuit tuning of custom circuits, the electrical engineering application that we consider in this paper, lies between those two extremes but fortunately, closer to the first than the second. The optimization problem formulation involves a relatively large number of variables (up to several $100,000$) and the involved functions are not given explicitly but are computed by simulation. In addition, circuit tuning is a good example of an important engineering application (critical in the design of custom computer chips), for which it is very desirable to increase the dimensionality of problems that can be solved in a reasonable amount of time.
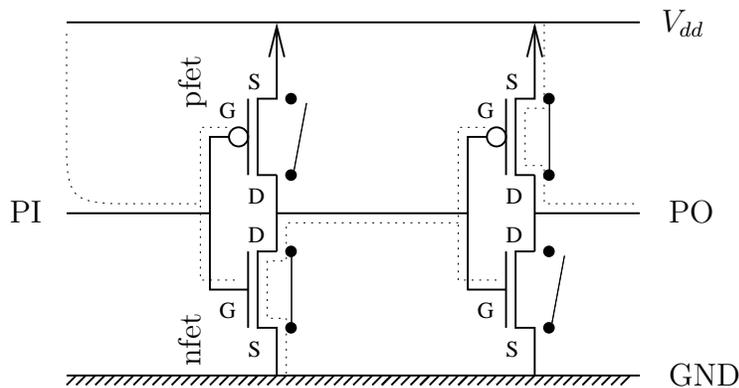
Fig. 1. Simple Example of Digital Circuit (2 Inverters)

In this paper we demonstrate how we achieved progress in this direction by using a state-of-the-art interior point method, replacing an earlier augmented Lagrangian, active-set code. The new method is more robust, requires less computation time and fewer function evaluations. Furthermore, we will present numerical results that show that the new approach has great potential to be used in a parallel/distributed computing environment, which we anticipate will lead to considerable additional reductions in computational time while increasing the scope of problems that can be addressed.

This paper is organized as follows: In Section 2, we introduce the application of circuit tuning, and present its mathematical formulation in Section 3. The subsequent section describes the software implementation, called EinsTuner. Its latest optimization engine, IPOPT, will be presented in Section 5. Numerical results will be discussed in Section 6. Finally, future work and conclusions are presented.

2 Circuit Tuning

A combinational digital circuit essentially implements a logical function, i.e., given Boolean values (by convention, "0" is represented by a low voltage and "1" by a high voltage) at its primary inputs (PIs), it "computes" the corresponding output values at its primary outputs (POs). To simplify the exposition in this paper we will restrict ourselves to CMOS (Complementary Metal Oxide Semiconductor) circuits, in which the transistors act as simple switches. The transistors are fabricated in two flavors: PFETs (P-type Field-Effect Transistors), which are "on" with a "0" on the gate, and NFETs (N-type Field Effect Transistors) which are turned on when a "1" is applied on the gate. These transistors are interconnected with wires in an integrated circuit to realize the required Boolean function.

A very simple example circuit is shown in Figure 1, where we see two inverters

3

in a row. In the depicted situation we assume that we have a "1" at the primary input (i.e., it has a high voltage commonly referred to as V_{dd}). Since in that case high voltage is applied to the gate G of the (upper) PFET in the first stage, this transistor will act as an open switch, i.e., no current can flow from its source S, to its drain D, whereas its complementary (lower) NFET will act as a closed conducting switch. Hence, the gates of the transistors in the second inverter will be connected to ground (GND), i.e., they obtain a "0" signal, and consequently act oppositely to the transistors in the first inverter, so that the primary output will again obtain a "1" signal.

In circuit tuning we are interested in the dynamic behavior of transistor networks. A change in the input signals leads to a change in the output signals, but since various transistor and wire parasitic capacitances have to charge or discharge to reach their new state, there is some delay (for current technologies on the order of 10^{-11} seconds) before the output signals achieve their correct new values. This delay depends on the sizes of the transistors. The larger a transistor is, the more current will have to flow through the upstream transistors in order to charge or discharge the gate of this transistor, but on the other hand this transistor itself can conduct more current and (dis)charge the next transistor stage and wires downstream more quickly. On the other hand, a larger transistor requires more area on the chip and generally consumes more power. Circuit tuning deals with the question of how to optimally choose the transistor sizes in order to optimize some aspect of the circuit (such as input-output delay or power consumption), while at the same time considering certain constraints (such as a limit on the area).

Today's computer chips contain very large digital circuits (with up to several hundred million transistors), and the tuning of the entire chip at once is impossible due to this vast size. For this and other design reasons, the chip is divided into functional units. The functional units are in turn divided into smaller "macros," which are designed and tuned separately and concurrently. These macros are currently typically of the order of up to tens of thousands of transistors. Even this reduced size is often too large to be optimized within a reasonable amount of time (say, 12 hours) using today's approaches, so that many transistor sizes are marked as untunable and only those believed to be critical are changed in the tuning process. This of course usually leads to suboptimal solutions, and there is a high incentive to be able to increase the number of transistors that can be optimized at once. In this paper we describe how this goal can be achieved by improving the optimization algorithm and exploiting parallel computing.
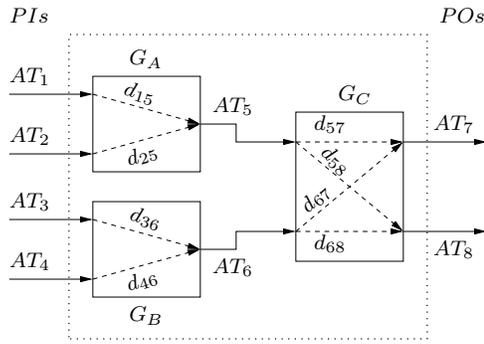
Fig. 2. Simple circuit

3 Mathematical Formulation

Given a complex digital circuit, we can decompose it into "basic logical gates," so-called Channel Connected Components (CCCs), such as NAND and NOR gates, which typically consist of only a few transistors. Consider the network example of Figure 2, consisting of three simple CCCs (G_A, G_B, and G_C). With each wire [1] i in the network we associate an *arrival time* AT_i, which is defined as the earliest point in time (relative to some reference time such as the clock cycle initiation) at which it is guaranteed that the signal (i.e., voltage) in this wire has its correct final value after an arbitrary realistic change at the primary inputs.

In each CCC we have delays, d_{ij}, between inputs i and outputs j, which denote the time required to pass a signal change at i to j (we neglect delay due to wires for simplicity). In the above example we can therefore express the propagation of arrival times [2] by

$$AT_5 = \max\{AT_1 + d_{15}, AT_2 + d_{25}\}, \quad AT_6 = \max\{AT_3 + d_{36}, AT_4 + d_{46}\},$$
$$AT_7 = \max\{AT_5 + d_{57}, AT_6 + d_{67}\}, \quad AT_8 = \max\{AT_5 + d_{58}, AT_6 + d_{68}\},$$

or more generally,

$$AT_j = \max\{AT_i + d_{ij} : i \in \text{input}(j)\}, \tag{1}$$

where input(j) denotes the set of wires that are inputs to the CCC at which j is an output.

A delay d_{ij} is a nonlinear function of the transistor sizes in the CCC, w_j, as well as the size of the transistors in the next CCC(s), $w_{\text{next}(j)}$, which have to be driven by output j. In addition, d_{ij} depends on the waveform of the incoming signal at i, particularly on how quickly it changes from one state to the other. In practice, we approximate the waveform as a piecewise linear

---

[1] For simplicity, we use the word wire here for each set of directly connected wires.
[2] Analogously to PERT procedures.

5

function, and define the *slew* as the time required to switch from the level at 10% to that at 90% (see Figure 3), or vice versa for a falling signal. The larger the slew of the incoming signal at i, the longer will be the delay d_{ij}. In the problem formulation we therefore also associate with each wire i a slew s_i, and with each input/output pair $i \in \text{input}(j)$ an output slew function, S_{ij}, which like d_{ij} depends on w_j, $w_{\text{next}}(j)$, and s_i. As a pessimistic but safe estimate we propagate worst slews in a manner analogous to (1) as

$$s_j = \max\{S_{ij} : i \in \text{input}(j)\}. \tag{2}$$

Assume now that we want to minimize the overall delay of the circuit, i.e., the worst arrival time at all primary outputs, subject to an area constraint and simple bounds on the transistor sizes and slews. This can be formulated as the following optimization problem:

$$\min_{AT,s,w} \ \max\{AT_i : i \in PO\},$$
$$\text{s.t.} \ AT_j = \max\{AT_i + d_{ij}(w_j, w_{\text{next}(j)}, s_i) : i \in \text{input}(j)\},$$
$$s_j = \max\{S_{ij}(w_j, w_{\text{next}(j)}, s_i) : i \in \text{input}(j)\},$$
$$\sum_i k_i w_i \leq A,$$
$$w^L \leq w \leq w^U, \quad s^L \leq s \leq s^U,$$

where A is the fixed area target, and the area requirement of the circuit design is approximated as a linear combination of the individual transistor sizes[3]. Arrival times at primary inputs are given.

The difficulty with this problem formulation is that the objective function and some of the constraints are non-differentiable, because of the max operator, whereas most efficient optimization algorithms, in particularly the algorithms used in our implementation, require sufficiently smooth objective and constraint functions. Therefore, we introduce an auxiliary variable z and reformulate the problem as

$$\min_{AT,s,w,z} \ z, \tag{3a}$$
$$\text{s.t.} \ z \geq AT_i \qquad\qquad\qquad i \in PO, \tag{3b}$$
$$AT_j \geq AT_i + d_{ij}(w_j, w_{\text{next}(j)}, s_i) \qquad i \in \text{input}(j), \tag{3c}$$
$$s_j \geq S_{ij}(w_j, w_{\text{next}(j)}, s_i) \qquad\quad i \in \text{input}(j), \tag{3d}$$
$$\sum_i k_i w_i \leq A, \tag{3e}$$
$$w^L \leq w \leq w^U, \quad s^L \leq s \leq s^U. \tag{3f}$$

Since we assume that the functions d_{ij} and S_{ij} are monotonically increasing in the slew variables, s_i, a local solution of one formulation corresponds to a

---

[3] AT, s and w denote appropriately dimensioned vectors with entries AT_i, etc.

local solution of the other one in the sense that the objective functions are identical, as well as all transistor sizes w.

Additionally, we may have to ensure that the input capacitance at a primary input $i \in PI$, which is a function of the transistor sizes in the CCCs having i as input, does not become too large. This is necessary in order to guarantee that the circuit under consideration can actually be driven, for example, by a preceding macro. We therefore add the constraints

$$C_i(w) \leq C_i^{\max}, \tag{4}$$

for each $i \in PI$, where $C_i(w)$ is an analytic expression for the total capacitance with upper bound $C_i^{\max}$. Finally, to balance rise and fall times and guard against electrical noise events, we need to ensure that the ratio of the sizes of complementary PFET and NFET transistors in each CCC, k, stays within a certain range given by $[\beta_k^L, \beta_k^U]$,

$$\beta_k^L \leq \frac{\frac{1}{\sum_j w_{k,j}^{\text{NFET}}}}{\frac{1}{\sum_j w_{k,j}^{\text{PFET}}}} \leq \beta_k^U. \tag{5}$$

The overall mathematical formulation for the delay minimization problem is then the nonlinear optimization problem (3)-(5), where the nonlinear functions d_{ij} and S_{ij} are obtained by simulation of the individual CCCs, as described in the next section. In practice, other problem formulations, such as minimizing the area requirement subject to a bound on the overall delay, or a weighted combination of these goals, are also considered.

For simplicity, the description above neglected the fact that the delay and slew functions are different for rising and falling signals. Our problem formulation therefore takes account of arrival times and slews for both falling and rising signals. Furthermore, it is possible for a designer to group the sizes of certain transistors for layout reasons. The production software accommodates delays in wires, non-complementary topologies, sequential circuits and many other such complications that are beyond the scope of this paper.

4 EinsTuner

Over the past few years, the circuit tuning tool **EinsTuner** has been developed at IBM as a joint effort between the Research and Electronic Design Automation departments. It allows designers of custom circuits to conveniently define a tuning problem using a graphical user interface (in the **CADENCE** schematic environment, with which they were already accustomed), execute the numerical optimization procedure, and examine the results. It has had
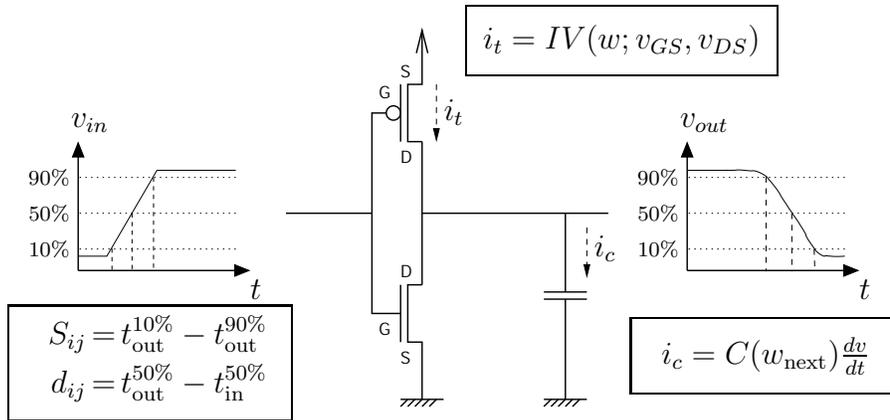
Fig. 3. Simulation of a gate

an important impact on the final design of IBM's recent microprocessors. As demonstrated earlier [3], an average gain of 15% was obtained compared with carefully hand-tuned circuits. More importantly, it allows designers to concentrate on higher-level tasks such as finding the best topology, exploring "what-if" scenarios, and examining trade-offs, and to leave the tuning effort to the optimizer, thus boosting designers' productivity.

The computation of the values and gradients of the delay and slew functions in EinsTuner is based on time-domain simulation of the CCCs, which can be modeled as a system of differential and algebraic equations (DAEs) for voltage and current (see Figure 3). This DAE system is integrated for given transistor sizes, with the appropriate input, i, excited (based on the slew s_i). The corresponding delays d_{ij} and S_{ij} are obtained from the waveforms at the outputs. For each CCC, such a simulation is performed for all possible input-to-output transitions. The set of such transactions is determined by logical state analysis software.

The simulation is performed by the fast event-driven simulator SPECS [4]. SPECS uses tables as models for the various device i-v characteristics (IV), which define the current flows through the device depending on the voltages on the inputs. Specialized integration techniques, event-driven simulation, and simplified device models enable SPECS to be 70x faster than AS/X (an IBM-internal SPICE-like simulator) at a relative stage timing error of 5% or less. The key feature of SPECS exploited in EinsTuner is a mature gradient computation capability that has been extensively used previously in a dynamic tuner [5]. SPECS computes incremental time-domain sensitivity information by both the adjoint and direct methods. The adjoint method is used in EinsTuner. The sensitivity of a single time-domain measurement (such as delay, slew, power or noise) can be computed with respect to any number of parameters (such as transistor widths, input slews or fanout capacitances) in a single adjoint analysis. Each adjoint analysis is a small incremental computational overhead on the nominal simulation. Various measures were taken to attempt to minimize the numerical noisiness of the data provided by the simulator.

In its previous releases, EinsTuner used the nonlinear optimization code LAN-CELOT [6] in order to solve the problem (3)-(5). LANCELOT is based upon an augmented Lagrangian trust-region method, using conjugate gradients and an active-set approach for the computation of the trial steps. Since the values and gradients of the constraint functions d_{ij} and S_{ij} are obtained by simulation and are therefore computationally expensive, several efforts were made to make LANCELOT more aggressive, such as a two-step updating scheme for slack and minmax variables [7]. In addition, a preprocessing step ("pruning") has been developed that produces a smaller but equivalent problem formulation by removing appropriate nodes in the timing graph [8]. Finally, as we already noted, the functions d_{ij} and S_{ij} are only computed within some accuracy due to the iterative procedure of the simulator, and certain precautions had to be taken to handle this numerical noise.

While LANCELOT has been relatively successful in solving tuning problems up to a certain size (about 80,000 variables including slack variables), larger problems could not be solved, either because the optimization method failed or the computation time became too long for practical purposes. Moreover, LANCELOT is a robust but rather old code. We therefore investigated the benefits of using a different optimization algorithm, which is presented in the next section.

5   IPOPT

In this section we give an overview of the nonlinear optimizer IPOPT [4] [9] that has recently been integrated into EinsTuner as the default tool for solving the optimization problem (3)-(5). To simplify notation, we describe the method for the problem formulation

$$\min_{x \in \mathbb{R}^n} f(x), \tag{6a}$$
$$\text{s.t. } c(x) = 0, \tag{6b}$$
$$x \geq 0, \tag{6c}$$

where the objective function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ and the equality constraints $c : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ with $m < n$ are assumed to be twice continuously differentiable. General inequality constraints can be handled by means of slack variables, i.e., "$g(x) \geq 0$" is written as "$g(x) - s = 0, s \geq 0$."

IPOPT implements a line-search filter interior point (or barrier) algorithm exploiting any sparsity of the derivatives. In contrast to active-set methods (such as LANCELOT), interior point methods do not suffer directly from the

---

[4]   An open source version of IPOPT is available at `http://www.coin-or.org/Ipopt`

combinatorial complexity of actively deciding what subset of the inequality constraints (6c) is binding at the solution. Since increased computation time in LANCELOT for large problems seemed to be due to this identification task, IPOPT was a promising candidate for improving the efficiency.

In the following we will describe the algorithm in some detail. As a barrier method, IPOPT solves a sequence of *barrier problems*

$$\min_{x \in \mathbb{R}^n} \varphi_\mu(x) := f(x) - \mu \sum_{i=1}^n \ln(x^{(i)}), \tag{7a}$$

$$\text{s.t. } c(x) = 0, \tag{7b}$$

for a decreasing sequence of *barrier parameters*, μ_l, with $\lim_{l\to\infty} \mu_l = 0$ [10]. Equivalently, we may understand this as applying a homotopy method to the first order optimality conditions of (7) (the primal-dual equations)

$$\nabla f(x) + \nabla c(x)\lambda - z = 0, \tag{8a}$$

$$c(x) = 0, \tag{8b}$$

$$XZe - \mu e = 0, \tag{8c}$$

with the homotopy parameter μ [11,12]. Here, $\lambda \in \mathbb{R}^m$ are the Lagrangian multipliers for the equality constraints (6b), and $z \in \mathbb{R}^n$ corresponds to the Lagrangian multipliers for the bound constraints (6c) in the limit as $\mu \to 0$. We further use the notation $X := \mathrm{diag}(x)$, $Z := \mathrm{diag}(z)$, and e for the vector of all ones of the appropriate dimension. Note that the optimality conditions for the original problem (6) are the equations (8) for $\mu = 0$ together with "$x, z \geq 0$."

Similarly to the algorithm proposed in [13], IPOPT computes an approximate solution to the nonlinear system (8) for a *fixed* value of μ, then decreases the barrier parameter, and iterates on the next barrier problem from the approximate solution of the previous one. In the following, we will describe how the approximate solution for (7) is computed for a fixed value of the barrier parameter.

IPOPT applies a damped Newton's method to the primal-dual equations (8). Given an iterate (x_k, λ_k, z_k) with $x_k, z_k > 0$, the search direction $(d_k^x, d_k^\lambda, d_k^z)$ is obtained from the linearization of (8), namely

$$\begin{bmatrix} W_k & \nabla c(x_k) & -I \\ \nabla c(x_k)^T & 0 & 0 \\ Z_k & 0 & X_k \end{bmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \\ d_k^z \end{pmatrix} = -\begin{pmatrix} \nabla f(x_k) + \nabla c(x_k)\lambda_k - z_k \\ c(x_k) \\ X_k z_k - \mu e \end{pmatrix}, \tag{9}$$

where W_k denotes the Hessian, or some approximation of it, for the Lagrangian

10

function
$$\mathcal{L}(x, \lambda, z) := f(x) + c(x)^T \lambda - z. \tag{10}$$
Instead of solving the linear system (9) directly, IPOPT obtains the equivalent solution by first solving the smaller linear system

$$\begin{bmatrix} W_k + \Sigma_k & \nabla c(x_k) \\ \nabla c(x_k)^T & 0 \end{bmatrix} \begin{pmatrix} d_k^x \\ \lambda_k^+ \end{pmatrix} = - \begin{pmatrix} \nabla \varphi_\mu(x_k) \\ c(x_k) \end{pmatrix}, \tag{11}$$

where $\Sigma_k := X_k^{-1} V_k$, and then computing d_k^λ from

$$d_k^\lambda = \lambda_k^+ - \lambda_k \tag{12}$$

and d_k^z from

$$d_k^z = \mu X_k^{-1} e - z_k - \Sigma_k d_k^x. \tag{13}$$

Having computed search directions from (11), (12), and (13), we now have to decide how far we want to go in these directions in order to obtain the next iterate

$$x_{k+1} := x_k + \alpha_k d_k^x, \quad \lambda_{k+1} := \lambda_k + \alpha_k d_k^\lambda, \quad z_{k+1} := z_k + \alpha_k^z d_k^z,$$

with step sizes $\alpha_k, \alpha_k^z \in (0, 1]$. Note that we allow a different step size in the z variables from that taken for the other variables.

Since we know that x and z are positive at an optimal solution of the barrier problem (7), we maintain this property for all iterates. Consequently, we apply the *fraction-to-the-boundary* rule

$$\bar{\alpha}_k := \max \left\{ \alpha \in (0, 1] : x_k + \alpha d_k^x \geq (1 - \tau)x_k \right\}, \tag{14a}$$
$$\bar{\alpha}_k^z := \max \left\{ \alpha \in (0, 1] : z_k + \alpha d_k^z \geq (1 - \tau)z_k \right\}, \tag{14b}$$

for a parameter $\tau \in (0, 1)$, usually close to 1, and choose $\alpha_k^z := \bar{\alpha}_k^z$. The step size $\alpha_k \in (0, \bar{\alpha}_k]$ for the remaining variables is then determined by a backtracking line search procedure exploring a decreasing sequence of trial step sizes, $\alpha_{k,l} = 2^{-l} \bar{\alpha}_k$, with $l = 0, 1, 2, \ldots$, using a variant of Fletcher and Leyffer's filter method [14].

The main idea behind filter methods is to consider the objectives of minimizing the constraint violation $\theta(x) := \|c(x)\|$ and minimizing the barrier function $\varphi_\mu(x)$, where a certain emphasis is given to the first measure to guarantee convergence to a feasible point. Figure 4 depicts a projection of $\mathbb{R}^n$ into the $(\theta(x), \varphi_\mu(x))$ half-plane. Each point in the original optimization space, such as the optimal solution x_* or the current iterate x_k, corresponds to a point in that graph, e.g., $(\theta(x_k), \varphi_\mu(x_k))$. In order to decide whether a trial point, $x_k + \alpha_{k,l} d_k^x$, should be accepted as the next iterate x_{k+1}, we first check whether
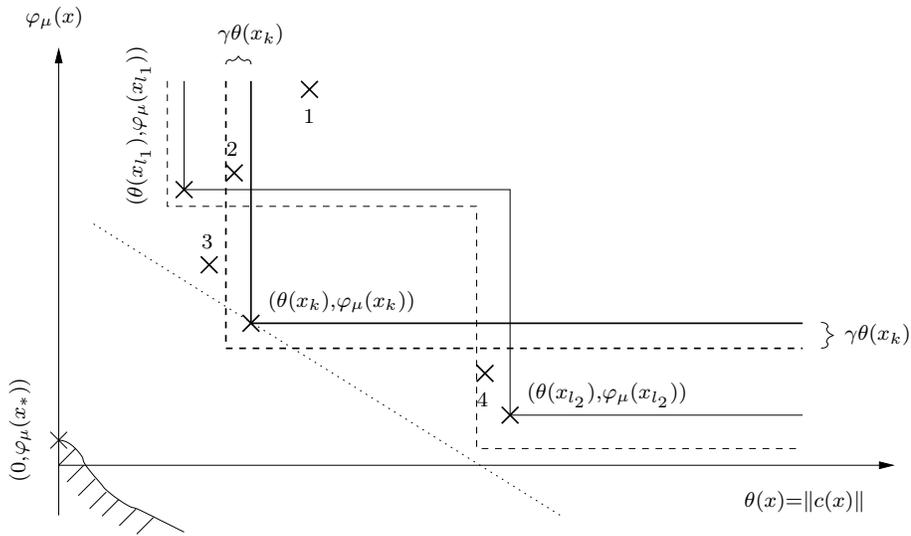
11

Fig. 4. The filter line-search method

it sufficiently improves one of the two measures θ or φ_μ compared with their values at x_k. In the example of Figure 4, a trial point corresponding to "1" would not be acceptable, since it worsens both measures. Also "2" would be rejected, even though it improves the infeasibility somewhat, but the decrease is insufficient (sufficient decrease is here defined by the margin depicted by the two dashed lines whose intersection is closest to $(\theta(x_k), \varphi_\mu(x_k))$). The trial point corresponding to "3" would be acceptable.

A few necessary safeguards have to be added to this simple procedure:

- If the current iterate is (almost) feasible but not sufficiently optimal, the sufficient-decrease condition with respect to x_k, as just described, is replaced by enforcing sufficient decrease in the barrier function φ_μ only.
- In order to avoid cycling, the (θ, φ_μ)-pairs corresponding to previous iterates that define a certain envelope (x_{l_1} and x_{l_2} in our example) are stored in a *filter*, and a trial point is only acceptable if it leads to a sufficient decrease relative to all those points. In the example, a trial point corresponding to "4" would be rejected since it does not sufficiently improve θ and φ_μ w.r.t. x_{l_2}.
- It may happen that no trial step size $\alpha_{k,l}$ exists which produces an acceptable point. If this situation is detected, the algorithm switches to a *feasibility restoration phase* whose purpose is to minimize the infeasibility (ignoring the objective function) until either a new acceptable iterate is obtained, or the infeasibility can no longer be reduced, e.g., if the problem (6) is (locally) infeasible.

The filter line search procedure implemented in IPOPT is formally stated and analyzed in [15]. The advantage of the filter method over previous techniques, such as a single merit function approach, is that it is usually less conservative

and allows one to take larger steps. (In Figure 4, points acceptable to an exact penalty function will have to lie below a straight line like the slanted dotted line.) Furthermore, the restoration phase safeguards the method from certain unnecessary failures, such as those described in [16,17].

Since in EinsTuner only first derivatives of the constraint functions (specifically d_{ij} and S_{ij}) are available from the gate simulation, we obtain the Hessian matrix, W_k, by a quasi-Newton method. This allows the approximation of the second derivatives of the Lagrangian Hessian based on gradient evaluations that have to be performed in any case. More precisely, we use the limited-memory BFGS method [18], so that the Hessian W_k used in (11) is of the form

$$W_k = \sigma_k I + U_k V_k^T, \tag{15}$$

for some suitable $\sigma_k > 0$ and $U_k, V_k \in \mathbb{R}^{n \times p}$. We use $p = 40$ in our implementation.

The computationally most expensive part of the optimization (not including the function and derivative computations) is overwhelmingly the solution of the linear system (11), which is of order up to $n + m = O(100, 000)$ equations. To exploit the sparsity of the problem, we note that this system together with (15) can be written as

$$\left(\begin{bmatrix} \sigma_k I + \Sigma_k & \nabla c(x_k) \\ \nabla c(x_k)^T & 0 \end{bmatrix} + \begin{bmatrix} U_k \\ 0 \end{bmatrix} \begin{bmatrix} V_k^T & 0 \end{bmatrix} \right) \begin{pmatrix} d_k^x \\ \lambda_k^+ \end{pmatrix} = - \begin{pmatrix} \nabla \varphi_\mu(x_k) \\ c(x_k) \end{pmatrix}$$

which may be solved using the Sherman-Morrison-Woodbury formula [19]. This only requires the factorization of the first matrix followed by $p + 2$ back-solves along with the factorization of a dense small matrix of size $p \times p$. This first matrix is very sparse, since $\sigma_k I_k + \Sigma_k$ is a diagonal matrix, and the constraint gradients $\nabla c(x_k)$ typically have only a few nonzero elements. In EinsTuner, the sparse parallel direct linear solver WSMP [20] is used for the factorization of this matrix. WSMP is a robust, high-performance, easy-to-use serial and parallel package for the direct solution of general and symmetric sparse linear systems. It is currently being extended to include iterative solution methods.

6   Numerical Results

We evaluated the performance of the new optimization engine in EinsTuner on 59 real-world test cases from high-performance microprocessor chip design. The sizes of the resulting nonlinear optimization problems (3)-(5) vary between $n = 1,261$ and $n = 161,701$ variables (including slacks), with about half of

the problems having more than $n = 10,000$ variables. The largest circuit has $71,022$ transistors, of which $19,576$ are independently tunable. The total computation time for tuning this circuit amounts to $122.5h$ and $21.5h$ for the old and new optimization engine, respectively, on a Risc System/6000 model SP_thin-260.

The optimization problems cannot be solved to a very high degree of accuracy, due to numerical noise in the functions d_{ij} and S_{ij}. However, after the optimization procedure has terminated, the transistor sizes are snapped to a discrete grid of manufacturable quantities, and therefore a highly accurate solution is not required.

Since the test examples stem from realistic circuit tuning problems, some of them are posed infeasibly. Small infeasibilities can be handled with certain heuristics (such as slightly relaxing the constraints). In 4 of the 59 instances, IPOPT was not able to find a feasible point. LANCELOT failed to find a feasible point for the same problems, as well as in four additional cases.

Due to the (normally desirable) relatively early termination of the optimization, as well as the postprocessing step, it is sometimes difficult to compare the solutions obtained by the two methods. However, inspection of the minimized delays of the circuits and the infeasibility reported to the designer after the postprocessing step shows that IPOPT overall provides better solutions. Considering the 51 test cases that were solved by both methods, IPOPT required on average only 74.1% of the number of function evaluations (simulations of all CCCs) and 46.9% of the total CPU time compared with LANCELOT[5]. Moreover, IPOPT seems to be considerably more robust and less sensitive to the scaling of the problem formulation, based upon observations in production use beyond the test cases reported here.

Beside improved efficiency, IPOPT offers another important related advantage. Figures 5 and 6 shows the distribution of the CPU time for the individual benchmark runs (ordered by increasing problem size), both for LANCELOT and IPOPT. It can be seen that (especially for the larger problems) a considerable fraction of the CPU time in a LANCELOT run is consumed by the optimization algorithm, whereas for IPOPT the greatest part of the computational time is spent in the gate simulation. Since the simulation of the individual CCCs can be performed independently from each other once the transistor widths and slews are proposed by the optimizer, this can easily be exploited in a parallel or a distributed computational framework. Such an implementation would considerably speedup the overall IPOPT runs. In addition, the remaining time in IPOPT is, for the most part, consumed by the solution of the linear system (11), which can also be performed in parallel using WSMP. In contrast to this,

---

[5] Here, we report total CPU time, i.e., IPOPT does not gain any advantage from the fact that it uses a parallel linear solver.
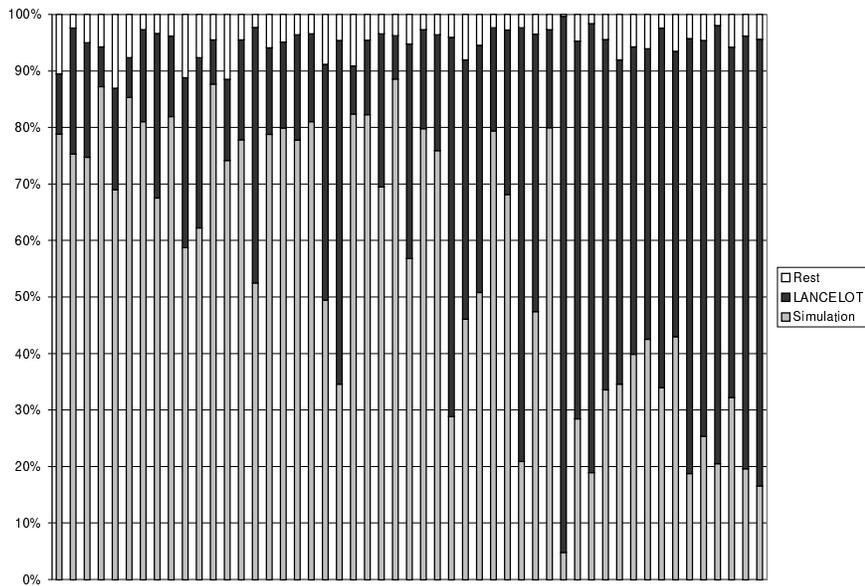
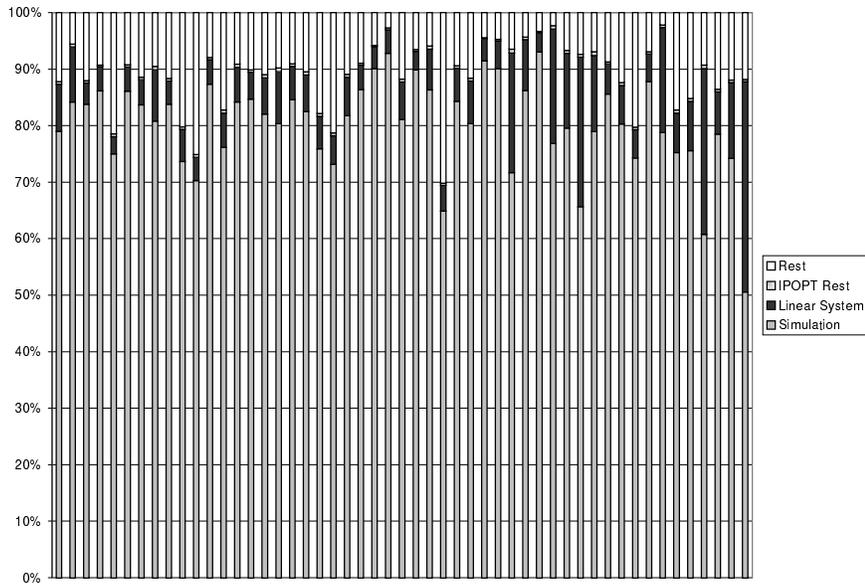Fig. 5. CPU time distribution for LANCELOT

Fig. 6. CPU time distribution for IPOPT

since LANCELOT follows an active-set approach and therefore has to solve a sequence of linear systems, it is more difficult, if not impossible to parallelize the previous optimization engine efficiently, even if the linear system solves are obtained by a parallel direct solver instead of by a conjugate gradient method.

7   Conclusions

Tuning of digital integrated circuits such as microprocessors is a very important task in the design of computers. Even though optimization approaches

based on nonlinear optimization techniques have been very successful in the past, there remains a high incentive to be able to tune larger circuits in less time. In this paper, we showed that the integration of a line-search filter interior point method IPOPT into the circuit tuning tool EinsTuner was able to improve robustness and efficiency significantly over the previous method. An additional benefit is the possibility of obtaining promising speedup via parallel computing. This will be explored in our future research.

Acknowledgments

References

[1] V. Akcelik, G. Biros, O. Ghattas, Parallel multiscale Gauss-Newton-Krylov methods for inverse wave propagation, in: Proceedings of SC2002, The SCxy Conference series, ACM/IEEE, Baltimore, Maryland, 2002.

[2] A. R. Conn, K. Scheinberg, Ph. L. Toint, Recent progress in unconstrained nonlinear optimization without derivatives, Mathematical Programming 79 (3) (1997) 397–414.

[3] A. R. Conn, I. M. Elfadel, W. W. Molzen, Jr., P. R. O'Brien, P. N. Strenski, C. Visweswariah, C. B. Whan, Gradient-based optimization of custom circuits using a static-timing formulation, in: Proc. Design Automation Conference, 1999, pp. 452–459.

[4] C. Visweswariah, R. A. Rohrer, Piecewise approximate circuit simulation, IEEE Transactions on Computer-Aided Design of ICs and Systems 10 (7) (1991) 861–870.

[5] A. R. Conn, P. K. Coulman, R. A. Haring, G. L. Morrill, C. Visweswariah, C. W. Wu, JiffyTune: circuit optimization using time-domain sensitivities, IEEE Transactions on Computer-Aided Design of ICs and Systems 17 (12) (1998) 1292–1309.

[6] A. R. Conn, N. I. M. Gould, Ph. L. Toint, LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A), Springer Verlag, 1992.

[7] A. R. Conn, L. N. Vicente, C. Visweswariah, Two-step algorithms for nonlinear optimization with structured applications, SIAM Journal on Optimization 9 (4) (1999) 924–947.

[8] C. Visweswariah, A. R. Conn, Formulation of static circuit optimization with reduced size, degeneracy and redundancy by timing graph manipulation, IEEE International Conference on Computer-Aided Design (1999) 244–251.

[9] A. Wächter, An interior point algorithm for large-scale nonlinear optimization with applications in process engineering, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, USA (January 2002).

[10] A. V. Fiacco, G. P. McCormick, Nonlinear Programming: Sequential Unconstrained Minimization Techniques, John Wiley, New York, USA, 1968, reprinted by SIAM Publications, 1990.

[11] R. H. Byrd, G. Liu, J. Nocedal, On the local behavior of an interior point method for nonlinear programming, in: D. F. Griffiths, D. J. Higham (Eds.), Numerical Analysis 1997, Addison–Wesley Longman, Reading, MA, USA, 1997, pp. 37–56.

[12] N. I. M. Gould, D. Orban, A. Sartenaer, Ph. L. Toint, Componentwise fast convergence in the solution of full-rank systems of nonlinear equations, Tech. Rep. TR/PA/00/56, CERFACS, Toulouse, France, revised May 2001 (1999).

[13] R. H. Byrd, M. E. Hribar, J. Nocedal, An interior point algorithm for large-scale nonlinear programming, SIAM Journal on Optimization 9 (4) (1999) 877–900.

[14] R. Fletcher, S. Leyffer, Nonlinear programming without a penalty function, Mathematical Programming 91 (2) (2002) 239–269.

[15] A. Wächter, L. T. Biegler, Global and local convergence of line search filter methods for nonlinear programming, Tech. Rep. CAPD B-01-09, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA, USA (August 2001).

[16] A. Wächter, L. T. Biegler, Failure of global convergence for a class of interior point methods for nonlinear programming, Mathematical Programming 88 (2) (2000) 565–574.

[17] M. Marazzi, J. Nocedal, Feasibility control in nonlinear optimization, in: A. DeVore, A. Iserles, E. Suli (Eds.), Foundations of Computational Mathematics, Vol. 284 of London Mathematical Society Lecture Note Series, Cambridge University Press, 2001, pp. 125–154.

[18] R. H. Byrd, J. Nocedal, R. B. Schnabel, Representations of quasi-Newton matrices and their use in limited memory methods, Mathematical Programming 63 (4) (1994) 129–156.

[19] J. Nocedal, S. Wright, Numerical Optimization, Springer, New York, NY, USA, 1999.

[20] A. Gupta, Recent advances in direct methods for solving unsymmetric sparse systems of linear equations, ACM Transactions on Mathematical Software 28 (3) (September 2002) 301–324.