High Performance Algorithms and Software for Nonlinear Optimization, pp. 1-000
G. Di Pillo and A. Murli, Editors
©2002 Kluwer Academic Publishers B.V.

Exploiting Optimality Conditions in Accurate Static Circuit Tuning

Chandu Visweswariah (chandu@watson.ibm.com)
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598, U.S.A.

Andrew R. Conn (arconn@watson.ibm.com)
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598, U.S.A.

Luís G. Silva (lgs@algos.inesc.pt)
IST/INESC
Lisbon, Portugal

Abstract

This paper describes, in a manner that is meant to be amenable to both mathematicians and engineers, the accurate optimization of transistor sizes on a static timing basis. Delays of individual gates (and gradients thereof) are obtained by transient (i.e. time-domain) simulation rather than simplified Elmore or analytical delay models. Slews (rise/fall times)[1] and their effects on delay are correctly taken into account. The optimization problem is stated in a form amenable to general-purpose nonlinear optimization. However, the size and inherent degeneracy of the resulting optimization problem make it difficult to solve. By considering the structure of the problem, optimality conditions are derived and conditions can be exploited to carry out the tuning more effectively and efficiently. Numerical results from the optimization of high-performance microprocessor circuits are presented. Further, an investigation of the viability and merits of an implementation of Lagrangian Relaxation in the same circuit optimization environment are detailed.

Keywords: Circuit optimization, Lagrangian relaxation, optimality conditions.

[1]The slew of a signal is the rate at which it changes. Some simplified models ignore the impact of input slew on the delay of a logic gate.

1 Introduction and motivation

The design of modern digital integrated circuits, which contain millions of transistors, is a complex task. One of the important design steps is circuit tuning or optimization. In the tuning step, the optimal size of each transistor is determined. Wider transistors generally lead to faster circuits, but consume more area and place a heavier burden on the previous stage of logic. Determining the optimal size for each transistor yields tremendous benefits, but is not an easy proposition.

Optimization of custom circuits is traditionally a manual, iterative, tedious and error-prone task. In contrast, automatic tuning improves performance and, perhaps more importantly, increases designer productivity. Circuit optimization is crucial in maximally exploiting silicon technology and making the right tradeoff choices. The availability of automated circuit tuning has had a profound impact on design methodology at IBM.

Ideally, one would be able to consider the entire chip, optimizing for performance, power, area, noise, layout considerations and other metrics. In practice, this is far too difficult a task. Therefore, we partition the chip into smaller macros that are tuned individually at the schematic level but with estimates of area and wire parasitics to aid the subsequent layout step. The circuit tuning task is then tackled by one of two complementary approaches, namely static and dynamic tuning.

Static tuning is a method wherein all paths through the circuit are considered simultaneously and each path is considered to be made up of simplified components that we might think of as the circuit building blocks or so-called channel connected components. Static timing analysis estimates the worst-case performance of the circuit irrespective of the input patterns applied to it. It does this by representing the circuit by a timing graph with nodes that map the physical circuit nodes and weighted directed edges that represent the worst-case delays between these nodes. Static tuning then finds the longest, or "critical" path(s) in the circuit by computing the graph theoretical delay of the network. These give rises to nonlinear optimization problems which are so large as to stress even the most advanced optimization packages. To make things worse, the problems exhibit a large amount of redundancy and degeneracy. To handle such problems, it is essential to exploit both structure and optimality conditions. Moreover, additional constraints like timing assertions, area, input loading, and rise and fall time limits are accommodated to render the tuning results useful.

By contrast, dynamic tuning inherently implies circuit optimization based on transient simulation of the underlying circuit. This requires careful and detailed specification of the optimization problem, but delivers more general capabilities in return. In particular, our own work in this area ([7], [8], [10] and [9]) addresses minimax and power optimization, and simultaneous transistor and wire tuning. Noise considerations, which are typically expressed as semi-infinite constraints, are mapped to integral equality constraints. This paper focuses on the static tuning approach.

There is a wealth of literature on solving the static optimization problem exactly. These problems have generally been formulated as nonlinear optimization problems [1, 2, 3] by introducing variables to represent the latest arrival time of logically correct signals at each node. Most previous approaches employ an Elmore or suitably chosen analytic delay model under which the static optimization is provably convex [4], and therefore any local solution is also a global solution. Unfortunately, for practical high-performance circuits designed in state-of-the-art technology, the accuracy of Elmore delay models is inadequate. Furthermore, modeling of slew (rise/fall time) and slew effects is crucial to maintaining good timing accuracy. Thus although these simplified delay models are convex, their accuracy is insufficient for production design and the guarantee of a global solution is essentially spurious.

Static optimization taking slew effects into account and using fast event-driven simulation for evaluating the timing characteristics of each channel-connected component (CCC — the simplified building blocks referred to above) was introduced in [5]. Unfortunately, such problems have a large number of variables and constraints even for modest-sized circuits. In addition, the formulation has inherent redundancy and degeneracy (as observed in [1, 3]), since arrival times and slews of non-critical signals can settle to one of several equally correct values at the solution. "Pruning" of arrival time variables [6] helps ameliorate some of the degeneracy and redundancy problems and significantly decreases the size of the optimization problem.

In this paper, the inherent structure of the static optimization problem is exploited further to derive optimality conditions on the Lagrange multipliers of both timing and slew constraints. The optimality conditions are used to effectively and efficiently solve relatively large static optimization problems. The resulting software has been used to tune numerous custom and synthesized macros of S/390 and PowerPC high-performance microprocessor circuits.

This paper includes a study of the use of Lagrangian relaxation. Dual methods in general and Lagrangian relaxation in particular have been suggested as efficient methods to solve static optimization problems [1, 3], and are indeed shown to be extremely efficient in the context of simple Elmore delay models when slews are ignored [11]. We have implemented Lagrangian relaxation in our more accurate environment and have found it to be slower than directly solving the primal problem with arrival time pruning. One potential benefit of Lagrangian relaxation, however, is that it provides a good lower bound to the solution of the original problem.

Section 2 briefly introduces the formulation of the static optimization problem whose optimality conditions are derived in Section 3. The exploitation of these optimality conditions is described in Section 4 and numerical results are presented in Section 5. The Lagrangian Relaxation implementation is discussed in Section 6 followed by conclusions.

2 Formulation

We assume that the circuit to be tuned consists of combinational parameterized cells, with one continuous parameter controlling the width of all the PFETs in the cell and one controlling those of all the NFETs [2]. Working with parameterized cells yields significant advantages during synthesis, tuning and automated layout. The formulation and optimality conditions derived below extend readily to full-custom transistor-level optimization and simultaneous transistor and wire sizing. The following notation is used in this paper.

$$
\begin{array}{rcl}
x^N & = & \mathrm{N-type\ transistor\ widths} \\
x^P & = & \mathrm{P-type\ transistor\ widths} \\
x & = & \mathrm{all\ transistor\ widths} \\
\beta & = & \beta \text{ ratios } (x^P/x^N) \\
AT & = & \mathrm{arrival\ times} \\
s & = & \mathrm{slews} \\
PO & = & \mathrm{set\ of\ primary\ outputs} \\
PI & = & \mathrm{set\ of\ primary\ inputs} \\
IN & = & \mathrm{set\ of\ internal\ nodes} \\
T & = & \mathrm{set\ of\ tunable\ transistors} \\
G & = & \mathrm{set\ of\ tunable\ gates} \\
RAT & = & \mathrm{required\ arrival\ times\ at\ POs} \\
 & & before \text{ capturing clock edge}^3 \\
A_{target} & = & \mathrm{area\ target} \\
L_{target} & = & \mathrm{input\ loading\ limits\ on\ PIs} \\
\beta_{lower} & = & \mathrm{lower\ bounds\ on\ } \beta \mathrm{\ ratio} \\
\beta_{upper} & = & \mathrm{upper\ bounds\ on\ } \beta \mathrm{\ ratio} \\
X_{lower} & = & \mathrm{lower\ bounds\ on\ } x \\
X_{upper} & = & \mathrm{upper\ bounds\ on\ } x
\end{array}
\tag{1}
$$

For the purposes of the description below we will ignore separate rising and falling arrival times and slews since they clutter the notation. In reality, each arrival time, slew, arrival time constraint and slew constraint has both a rising and falling version. Minimization of the cycle time of the circuit subject to area and other constraints is formulated as shown below. Note that z is an auxiliary variable introduced to represent the minimum cycle time of the circuit.

[2] Digital logic typically consists of an NFET "pull-down" tree made of n-type semiconductors and a complementary PFET "pull-up" tree made of p-type semiconductors. In the methodology considered herein one continuous parameter controls the width of all PFETS in the cell and one the NFETS.

[3] By convention, required arrival times are either expressed before a capturing clock edge or after a launching clock edge.

$$
\begin{array}{rrcll}
 & \min & & z & \\
 & x, z, AT, s & & & \\
\text{s.t.} & z & \geq & AT_i + RAT_i & i \in PO \\
\text{s.t.} & AT_j & \geq & AT_i + d_{ij}(x, s_i) & j \in (IN \cup PO), \\
 & & & & i \in fanin(j) \\
\text{s.t.} & s_j & \geq & s_{ij}(x, s_i) & j \in (IN \cup PO), \\
 & & & & i \in fanin(j) \\
\text{s.t.} & \sum_{i \in T} A_i x_i & \leq & A_{target} & \\
\text{s.t.} & \sum_{j \in T} L_{ij} x_j & \leq & L_{target_i} & i \in PI \\
\text{s.t.} & \beta_{lower_i} & \leq & x_i^P / x_i^N & i \in G \\
\text{s.t.} & x_i^P / x_i^N & \leq & \beta_{upper_i} & i \in G \\
\text{s.t.} & X_{lower} & \leq & x_i & i \in T \\
\text{s.t.} & x_i & \leq & X_{upper} & i \in T.
\end{array}
\tag{2}
$$

$A_i x_i$ is the contribution of gate i to the total area. $L_{ij} x_j$ is the contribution of transistor j to the loading of primary input i. Note that each of the d_{ij} and s_{ij} terms is a nonlinear function of input slew s_i, transistor widths of the relevant CCC and transistor widths of the fanout CCCs which constitute the loading capacitance. The implicit assumption is made above that the worst slew is propagated downstream rather than the slew corresponding to the last-arriving signal (see [5] for details). Arrival times and slews are variables of the problem. If the user desires, upper bounds can be placed on the slew variables. Timing and slew constraints are expressed above by traversing the timing graph. The formulation could easily be revised to minimize area subject to timing constraints, and all subsequent derivations in this paper would still be valid with minor changes.

3 Optimality conditions

It is convenient to rewrite the greater-than constraints of (2) as less-than constraints. By doing so, the Lagrangian of the above problem can be written as

$$
\begin{array}{l}
\mathcal{L}(x, z, AT, s, \lambda) = \\
z + \sum_{i \in PO} \lambda_i^{PO} (AT_i + RAT_i - z) \\
+ \sum_{j \in (IN \cup PO), i \in fanin(j)} \lambda_{ij}^{IN} (AT_i + d_{ij}(x, s_i) - AT_j) \\
+ \sum_{j \in (IN \cup PO), i \in fanin(j)} \lambda_{ij}^{SLEW} (s_{ij}(x, s_i) - s_j) \\
+ \lambda^{AREA} (\sum_{i \in T} A_i x_i - A_{target}) \\
+ \sum_{i \in PI} \lambda_i^{LOADING} (\sum_{j \in T} L_{ij} x_j - L_{target_i}) \\
+ \sum_{i \in G} \lambda_i^{\beta LOWER} (\beta_{lower_i} x_i^N - x_i^P) \\
+ \sum_{i \in G} \lambda_i^{\beta UPPER} (x_i^P - \beta_{upper_i} x_i^N) \\
+ \sum_{i \in T} \lambda_i^{XLOWER} (X_{lower} - x_i) \\
+ \sum_{i \in T} \lambda_i^{XUPPER} (x_i - X_{upper}).
\end{array}
\tag{3}
$$

The first-order optimality conditions are that all the constraints are satisfied, all multipliers are non-negative (since we are dealing exclusively with less-than constraints), complimentary slackness is satisfied (for each constraint either the multiplier is zero or the constraint is tight) and

$$\begin{array}{rcll} \nabla_z \mathcal{L} & = & 0 & \\ \nabla_{AT_i} \mathcal{L} & = & 0 & \forall i \in (IN \cup PO) \\ \nabla_{s_i} \mathcal{L} & = & 0 & \forall i \in (IN \cup PO) \\ \nabla_{x_i} \mathcal{L} & = & 0 & \forall i \in T. \end{array} \tag{4}$$

By taking advantage of (4) above and exploiting the topology of the network, we obtain the following optimality conditions. Optimal Lagrange multipliers are indicated by a * superscript. The relations below are obtained by focusing on a single arrival time or slew variable, writing all the terms of $\mathcal{L}$ in which it occurs, and then differentiating all such terms with respect to the variable being considered and setting the sum to zero.

$$\begin{array}{rcl} \sum_{i \in PO} \lambda_i^{PO^*} & = & 1 \\ \sum_{k \in fanin(i)} \lambda_{ki}^{IN^*} & = & \sum_{j \in fanout(i)} \lambda_{ij}^{IN^*} \ \forall i \in IN \\ \lambda_j^{PO^*} & = & \sum_{i \in fanin(j)} \lambda_{ij}^{IN^*} \ \forall j \in PO \\ \sum_{k \in fanin(i)} \lambda_{ki}^{SLEW^*} & = & \sum_{j \in fanout(i)} \lambda_{ij}^{SLEW^*} \frac{\partial s_{ij}}{\partial s_i} \\ & & + \sum_{j \in fanout(i)} \lambda_{ij}^{IN^*} \frac{\partial d_{ij}}{\partial s_i} \ \forall i \in IN \\ \sum_{k \in fanin(i)} \lambda_{ki}^{SLEW^*} & = & 0 \ \forall i \in PO. \end{array} \tag{5}$$

The optimality conditions on the arrival time constraint multipliers were first used in [2], but exploitation of the conditions on the slew constraint multipliers is new. The optimality conditions have several implications. The sum of the "downstream" arrival time multipliers at each node of the directed timing graph must equal the sum of the "upstream" multipliers. A similar relation is true for the slew multipliers, but in terms of the slew sensitivities of all the downstream nonlinear slews and delays. Note that the sensitivities are known once the simulation of the relevant CCCs has been completed. Also, a slew or arrival time constraint of a non-critical gate will have a zero multiplier.

All the multipliers for arrival time and slew constraints can be computed by a single forward and backward traversal of the timing graph. In the forward traversal, the arrival times and slews at each node are computed, as in a regular timing analysis. Downstream slew and arrival time multipliers are maintained for each node. Except for the downstream arrival time multiplier of the sink node which is initialized to 1.0, all the rest are initialized to zero. Then a backward traversal of the timing graph

begins. For each node visited, the *tight* upstream arrival time and slew constraints are assigned equal multipliers so as to add up to the correct downstream value. As soon as a multiplier is assigned, that value is added (straightforwardly for arrival time constraints, and in conjunction with delay and slew sensitivities for slew multiplier constraints) to the downstream multiplier of the upstream node of each tight constraint. Thus locally optimal multipliers are efficiently and easily determined by two graph traversals. In the case where a single path is critical, the multipliers can be determined uniquely. When there are multiple equally critical paths, a reasonable (but not necessarily good) guess for the multipliers is rapidly obtained by the graph traversal procedure.

4 Implementation details

The ideas of the preceding sections have been incorporated in a software tool called EinsTuner, which consists of several components.

The timing of each CCC is conducted by using SPECS [12, 13], which is a fast, transistor-level, event-driven simulator that uses reasonably accurate but simplified device models. SPECS also provides the gradients of each delay and slew measurement with respect to input slew, output load and transistor widths by using the adjoint method of sensitivity computation [14].

To solve the nonlinear optimization problem, LANCELOT [15, 16, 17], a general-purpose package is employed. LANCELOT uses an augmented Lagrangian merit function and trust-region approach. It has been customized in several ways to render circuit tuning more efficient. For example, two-step updating [18], failure recovery and special considerations for dealing with numerical noise [5] have been implemented.

Pruning is used to reduce the number of arrival time variables and is best illustrated by an example. Consider a fragment of a circuit shown in Fig. 1, where the blocks represent CCCs. The timing constraints in which the arrival time AT_3 appears are

$$\begin{array}{rcl} AT_3 & \geq & AT_1 + d_{13} \\ AT_3 & \geq & AT_2 + d_{23} \\ AT_4 & \geq & AT_3 + d_{34} \\ AT_5 & \geq & AT_3 + d_{35}. \end{array} \tag{6}$$

Slews and separate rising/falling arrival times have been omitted for simplicity. An equivalent set of constraints is

$$\begin{array}{rcl} AT_4 & \geq & AT_1 + d_{13} + d_{34} \\ AT_4 & \geq & AT_2 + d_{23} + d_{34} \\ AT_5 & \geq & AT_1 + d_{13} + d_{35} \\ AT_5 & \geq & AT_2 + d_{23} + d_{35}. \end{array} \tag{7}$$

It can be seen that by choosing

$$AT_3 = \max(AT_1 + d_{13}, AT_2 + d_{23}), \tag{8}$$

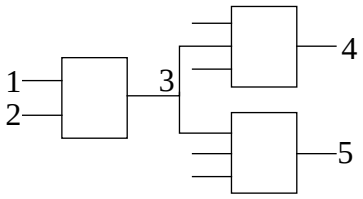

Figure 1: Fragment of a circuit.

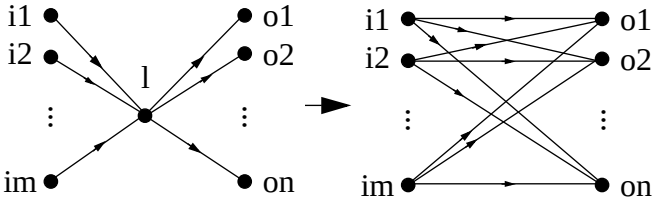

Figure 2: Basic graph manipulation.

the set of constraints in (7) is equivalent to the set of constraints in (6). By this simple manipulation, we went from a situation in which we had 5 arrival time variables and 4 constraints to a new situation with 4 arrival time variables and 4 constraints – a reduction in the dimensionality of the problem. Of course, each constraint now has extra terms. In fact, the summations of block delays can be thought of as path delays. However, the number of CCC simulations is unchanged, and having additional nonlinear terms in a constraint is not harmful to the optimizer we use since it exploits *group partial separability* ([15], page 109) and the overall number of distinct terms does not increase. For example, even though d_{13} appears twice in (7) but only once in (6), the value, gradient and approximate Hessian are computed only once. Moreover, the effective dimension of the two subspaces in which we are doing quasi-Newton updates are unchanged. We observe that pruning in this way is possible only because the arrival times do not appear nonlinearly in the original constraints. In graphical terms consider a segment shown on the left side of Fig. 2. The node l will be pruned. This node has m fanins (m edges are incident on this node), and n fanouts (n edges originate at this node). The timing constraints for this graph segment are shown below.

$$\begin{array}{rcll} AT_l & \geq & AT_{ij} + d_{ij,l} & \forall j \in 1, 2, \ldots, m \\ AT_{ok} & \geq & AT_l + d_{l,ok} & \forall k \in 1, 2, \ldots, n. \end{array} \tag{9}$$

We now consider eliminating or pruning AT_l from the above equations, to obtain

$$\begin{array}{rcll} AT_{ok} & \geq & AT_{ij} + d_{ij,l} + d_{l,ok} & \forall j \in 1, 2, \ldots, m, \\ & & & \forall k \in 1, 2, \ldots, n. \end{array} \tag{10}$$

The constraints in the above equation are shown graphically on the right side of

Fig. 2. As before, it can be seen that by choosing

$$AT_l = \max(AT_{ij} + d_{ij,l}) \ \forall j \in 1, 2, \ldots, m \tag{11}$$

the "pruned" set of constraints is equivalent to the original set of constraints. Details are given in [6].

The optimality conditions (5) can easily be rederived taking pruning into account. The timing arcs incident at each node of the pruned timing graph represent sub-path delays rather than just the delay through one CCC. Nonetheless, the basic upstream and downstream conditions hold as in (5).

After the first iteration of LANCELOT, all the multipliers are computed according to the optimality conditions derived in the previous section. When more than one upstream constraint is tight, each is given an equal multiplier value so that the total sums to the required downstream value. Clearly this may not be the optimal choice but given no further information it seems to be the obvious one. In subsequent successful iterations of LANCELOT, multipliers that were previously non-zero but whose corresponding constraints are not tight are reset to zero, and the merit function of LANCELOT recomputed. Again this seems to be the obvious choice given that we know that at optimality the multipliers corresponding to constraints that are not tight must be zero. As the optimization progresses, the non-zero multipliers increasingly correspond to the critical slew and timing constraints, thereby naturally guiding the optimizer to focus on the critical sections. At the start of each new major iteration, provided sufficient feasibility has been obtained, all multipliers are updated by the usual first-order mechanism of LANCELOT. Because of the nature of our additional multiplier updates in minor iterations of LANCELOT, after a finite number of such iterations the multipliers within any major iteration are fixed and the original convergence theory of LANCELOT applies.

Finally, the tool provides several practical enhancements to users such as a Cadence interface with graphical back-annotation, grouping facilities, tunability commands and support for timing assertions. The graphical interface provides the circuit designer with a "point-and-click" environment. The back-annotation involves a "snap-to-grid" to account for the fact that variables such as transistor widths are discrete (i.e., manufactured at particular sizes) although the optimization treats them as continuous. The resulting compromise is not significant.

5 Numerical results

The results reported here are those we had in a prototype code that has evolved into our present tuning tool, EinsTuner, that is continually being upgraded. EinsTuner is more general in many ways than the prototype (and where relevant we have tried to indicate this below).

EinsTuner was used to optimize a number of circuits at constant area and constant input loading. The results are presented in Table 1. "Real" circuits are designated

Table 1: Table of numerical results.

Name	# FETs	Start (ps)	End (ps)	%	# Its.	CPU (s)
inv3	6	132	122	7.6	61	5.1
c17	28	200	159	20.5	20	7.8
a3_3	34	326	259	20.6	41	21.5
a_graph	34	302	260	13.6	30	38.1
f_adder	46	316	295	6.6	41	21.5
s390-1	72	278	251	9.5	40	87.1
s390-2	102	385	324	15.7	67	990.5
s390-3	154	1573	1553	11.1	26	236.8
ppc-1	824	580	511	12.0	37	1818
s390-4	882	311	228	26.7	20	532.8
c8	584	740	597	19.3	53	1004
ppc-2	880	635	500	21.2	67	3687
c432	960	1476	1281	13.2	66	2768
ppc-3	1554	626	545	12.8	32	6156
s390-5	1400	950	588	38.1	42	7110
c880	1582	1618	1418	12.4	132	9225
s390-6	1892	1445	1401	12.1	47	13820
ppc-4	2726	1275	1062	23.7	67	53380
c1355	2180	1250	1164	6.9	18	2954
c499	2216	1272	1153	9.4	38	4391
c2670	2796	1167	1016	12.9	40	8575
c3540	5164	2336	2101	10.1	51	12246

with a "ppc" or "s390" prefix to denote macros from PowerPC and S/390 microprocessors, respectively. The original macro names are not shown for confidentiality purposes. The remaining benchmarks are "artificial" (e.g., ISCAS-'85 circuits). The start points on the real designs were provided by the designer, and the circuits had been hand-tuned to different extents before running EinsTuner. The artificial benchmarks were sized with a gain-based sizing after library-mapping to obtain a start point for the optimization. The "start" and "end" columns represent the minimum cycle time (the variable z of Section 2) before and after tuning. Some of this cycle time can consist of assertions (arrival times at the inputs and required arrival times at the outputs), so percentage improvement is computed as the ratio of the improvement in the cycle time to the pre-tuning critical path delay without any assertion offset. The sixth column shows the number of optimization iterations required (which is the same as the number of times the functions need to be evaluated using simulation) and the last column shows the run time. All tests were run on various models of IBM Risc/System 6000 workstations by distributing the runs on a pool of heteroge-

Table 2: Table of high-performance S/390 microprocessor results.

Name	# Gates	Gain
1	22	10.4%
2	24	16.6%
3	24	11.6%
4	41	15.2%
5	128	15.9%
6	128	17.1%
7	162	10.0%
8	42	6.0%
9	183	18.2%
10	295	14.5%
11	317	9.5%
12	487	15.8%
13	487	18.8%

neous machines. Healthy delay improvements averaging 15% are observed, even on previously tuned circuits, and the run times for circuits up to 5,000 transistors was manageable (4 hours or less, except for 15 hours in one case). The largest circuit tuned by the prototype EinsTuner contains about 8,000 transistors and requires 880 MB of memory and a day of CPU time.

In addition to the results in the above table, ten macros and three sub-macros of the instruction unit of a high-performance S/390 microprocessor were tuned with EinsTuner. These results are given in Table 2. The macros included several flavors of 64-bit comparators, 40-bit wide OR, 56-bit incrementer with carry-gating, branch hit comparison logic, and so on. Most of these circuits were in the 3,000 to 5,000 transistor range. EinsTuner consistently gained 20% or more on the critical path delay of these circuits, enabling the unit to meet timing requirements. The circuits were typically first tuned at the schematic level with estimated parasitics, and then again in extracted mode after the placement and routing. Again gain is defined as cycle time improvement divided by initial tunable part of the critical path delay without offset. As before, without offset implies that assertions are discounted in computing the portion of the delay actually spent in the macro being tuned.

For synthesized random logic macros, latches were cut out by moving assertions to the inner boundaries of source and sink latches, and modeling the driver and loading of latches by appropriately sized untunable gates. The resulting combinational circuits were tuned with EinsTuner. A suite of 10 synthesized macros of a PowerPC microprocessor was benchmarked with and without taking advantage of the optimality conditions. The critical path delay was improved on average 15.7% at constant area

Table 3: Table of results for synthesized macros of a PowerPC microprocessor.

Name	# Gates	# Transistors	Gain
1	372	1538	19.2
2	546	2192	14.3
3	314	1247	23.5
4	250	1009	11.4
5	47	174	4.4
6	200	859	15.6
7	94	383	24.8
8	50	194	17.6
9	511	2071	13.6
10	230	912	10.8

and constant input loading. Without taking advantage of optimality conditions, the quality of the solutions was worse, and the average improvement was only 9.5%. The consistency and robustness of the software has also improved due to the exploitation of the optimality conditions. It is important to understand that in the presence of numerical noise and degeneracy, it is not always possible to solve the optimization problem as accurately as we would like; taking advantage of the optimality conditions enables us to converge faster and more effectively. These results are given in Table 3.

6 Lagrangian relaxation

Lagrangian relaxation has been suggested in the electrical engineering literature as an efficient method of solving the static optimization problem [1, 3, 11]. Mathematically, it is well known that such an approach depends critically on being able to solve the inner problem rapidly. We have implemented Lagrangian relaxation in the EinsTuner environment as an option to test the viability of such an approach. This section describes the implementation and results.

We consider the first three sets of constraints of (2), i.e., the arrival time and slew constraints, to be "complicating constraints" and relax them into the objective function. The area constraint, input loading constraints and simple bounds on the

transistor widths are not relaxed. Thus we obtain the following reformulated problem.

$$\begin{array}{lllll} & \max & \min & \mathcal{L}_{relaxed}(x, z, AT, s, \lambda) & \\ & \lambda & x, z, AT, s & & \\ \text{s.t.} & \sum_{i \in T} A_i x_i & \leq & A_{target} & \\ \text{s.t.} & \sum_{j \in T} L_{ij} x_j & \leq & L_{target_i} & i \in PI \\ \text{s.t.} & \beta_{lower_i} & \leq & \beta_i & i \in G \\ \text{s.t.} & \beta_i & \leq & \beta_{upper_i} & i \in G \\ \text{s.t.} & X_{lower} & \leq & x_i & i \in T \\ \text{s.t.} & x_i & \leq & X_{upper} & i \in T, \end{array} \tag{12}$$

where $\mathcal{L}_{relaxed}$ is the Lagrangian consisting of the original objective function and the complicating constraints

$$\begin{array}{l} \mathcal{L}_{relaxed}(x, z, AT, s, \lambda) = z \\ + \sum_{i \in PO} \lambda_i^{PO} (AT_i + RAT_i - z) \\ + \sum_{j \in (IN \cup PO), i \in fanin(j)} \lambda_{ij}^{IN} (AT_i + d_{ij}(x, s_i) - AT_j) \\ + \sum_{j \in (IN \cup PO), i \in fanin(j)} \lambda_{ij}^{SLEW} (s_{ij}(x, s_i) - s_j). \end{array} \tag{13}$$

Substituting the optimality conditions λ^* derived in (5), we get

$$\begin{array}{l} \mathcal{L}_{relaxed}(x, s) = \sum_{i \in PO} \lambda_i^{PO^*} RAT_i \\ + \sum_{j \in (IN \cup PO), i \in fanin(j)} \lambda_{ij}^{IN^*} d_{ij}(x, s_i) \\ + \sum_{j \in (IN \cup PO), i \in fanin(j)} \lambda_{ij}^{SLEW^*} (s_{ij}(x, s_i) - s_j). \end{array} \tag{14}$$

Observe that $\mathcal{L}_{relaxed}$ is independent of z and the arrival time variables AT [11]. Also, as optimality is approached, many arrival time and slew multipliers corresponding to off-critical paths will gravitate towards zero. Thus the inner optimization problem with the objective function (14) is solved essentially in the transistor width variables, a subset of the slews, and possibly slack variables introduced by the nonlinear optimizer for the inequalities that were not relaxed.

Lagrangian relaxation was implemented in EinsTuner, using LANCELOT to solve the inner minimization problem. The outer maximization was conducted by taking a subgradient step exactly as suggested in [11]. The step size chosen was k/n where k is a constant and n the iteration counter. This sequence of step sizes tends to zero as n goes to ∞, while the sum of the steps diverges, both requirements for convergence. Once the multipliers are updated with a subgradient step, they will in general not satisfy the optimality conditions. However, the smaller inner subproblem assumes that the multipliers satisfy these conditions. To overcome this difficulty, the results of the subgradient update are projected onto the nearest point on the hyperplane representing the optimality conditions, as in [11]. The projection is carried out subject to simple bounds (e.g., multipliers corresponding to less-than constraints must be positive). During the projection, in contrast to the method used to update multipliers in the direct solution of the primal problem, criticality or tightness of constraints is

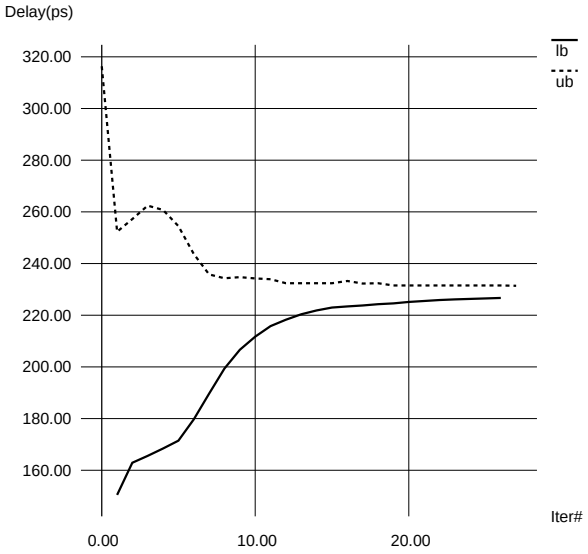

Figure 3: Upper and lower bounds plotted against iteration count.

not taken into account to prevent the non-zero multipliers from bouncing around from one critical path to another.

The choice of step size and projection method are crucial to obtaining an efficient implementation. For example, if k is too large, the multipliers display oscillatory behavior and converge very slowly.

One well-known and interesting feature of Lagrangian relaxation is that it yields a lower and upper bound to the original problem solution at each iteration [20]. The objective function of the relaxed problem when minimized for a given set of multipliers is a lower bound on the original problem and the objective function of the original problem an upper bound. Fig. 3 shows a plot of the lower and upper bounds obtained as a function of iteration count during the optimization of a full adder circuit. The availability of the bounds gives us an intuitively appealing termination criterion which is to stop when the lower and upper bounds agree to within a tolerance. In practice, as we shall see, this criterion is only useful for relatively small problems.

Table 4 shows numerical results comparing EinsTuner with and without Lagrangian relaxation; "w/o" indicates without, "w/" indicates with Lagrangian relaxation. The number of iterations without Lagrangian relaxation, outer iterations with Lagrangian relaxation and total number of inner iterations with Lagrangian relaxation are shown in the third, fourth and fifth columns, respectively. While the runs were conducted on various models of IBM Risc/System 6000 workstations, the two runs corresponding to each benchmark circuit were on the same machine so that CPU times can be compared. The two versions produced the same solution in all cases. The stopping criterion was either that the upper and lower bound agreed to within 5 ps, or all the multipliers converged to within 0.01. Generous tolerances and several schemes were used to improve the performance of the Lagrangian relaxation algorithm. Despite

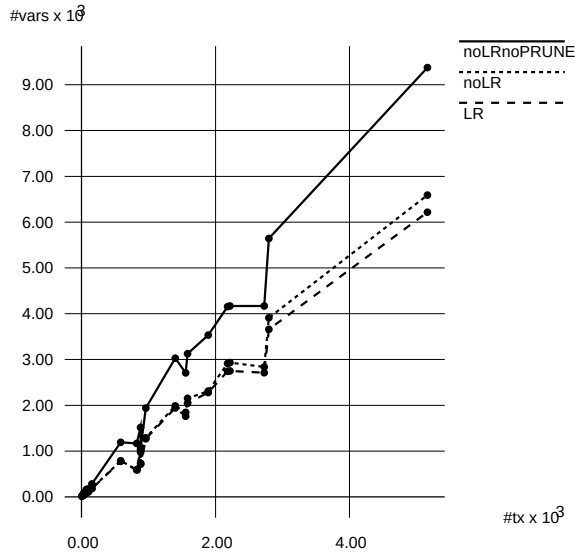

Figure 4: Number of variables versus problem size.

that, as can be seen from the table, Lagrangian relaxation does not yield a CPU advantage, and in fact the run times rapidly get much worse than solving the original problem without relaxation. We attribute this lack of improvement to two factors.

- Because of arrival time pruning, the number of variables in the inner problem is not much smaller than the original problem, so there is not much gain from the relaxation. In other words, pruning has already gained in a mathematically appealing fashion much of the dimensionality reduction that the special structure of the problem (also exploited in the Lagrangian relaxation) offers. Fig. 4 shows the number of variables in the optimization problem with and without Lagrangian relaxation. The difference is marginal. Without arrival time pruning, however, as the figure shows, the difference would be much greater. Further, in [11], slews were not considered, so the inner problem only had transistor width variables, and the inner problem could be solved very efficiently due to the simple delay modeling. In our context, solving the inner problem is almost as computationally intensive as solving the original problem; however, it has to be repeated until the multipliers converge.

- The sub-gradient update of the multipliers in the outer maximization yields at best linear convergence and thus convergence can be very slow particularly on large problems. During early iterations, the inner problem is solved crudely to reduce the computational burden.

We would argue that in spite of the results in [11], a Lagrangian relaxation approach is essentially undesirable because of the difficulty, alluded to above, of solving the inner subproblems when more reasonable models than Elmore ones are used. So although it might be preferable to use bundle methods, see for example [19], we have

Table 4: Numerical results with Lagrangian relaxation.

Name	# FETs	Iterations			CPU (s)	
		w/o	w/	w/ $\sum$	w/o	w/
inv3	6	61	2	36	5.1	5.3
a3_3	34	41	17	215	21.5	138.2
a_graph	34	30	28	303	38.1	434.7
f_adder	46	41	16	182	21.5	121.1
s390-1	72	40	34	109	87.1	479.7
s390-2	102	67	61	471	990.5	8426
s390-3	154	26	29	131	236.8	1502
ppc-1	824	37	117	1181	1818	57430
s390-4	882	20	156	386	532.8	14553
c8	584	53	194	1372	1004	37068

not implemented such an approach. Bundle methods, better step size choices and better methods of projecting the multipliers on the optimality hyper-plane can be investigated and may yield improvements, but in our opinion Lagrangian relaxation is highly unlikely to be superior to solving the pruned primal problem directly. In conclusion, it would appear that for realistic nonlinear delay models, since the inner iterations are not much cheaper, and the outer iterations converge too slowly, Lagrangian relaxation is not a useful approach.

7 Conclusions

This paper described the optimal sizing of digital CMOS circuits on a static timing basis. Optimality conditions were derived which relate Lagrange multipliers corresponding to arrival time and slew constraints. By taking advantage of these relations, the static optimization problem can be solved efficiently and effectively. Numerical results on industrial microprocessor circuits up to 5,000 transistors in size were demonstrated with the prototype. Depending on the quality of the start point, an improvement of 15% or more was often achieved on previously tuned circuits. Our current largest benchmark problem has nearly 33,000 variables and more than 28,000 nonlinear constraints. The largest circuit tuned to date by EinsTuner contains 140,109 variables (including slacks) and 110,767 constraints (all but one nonlinear) and 28,907 free variables at the solution. It required 1.4 GB of memory and 4 days, 8 hours of CPU time. The macro contained 12,167 CCCs and 47,748 transistors.

Lagrangian Relaxation was implemented in this same framework, but in contrast to previous work with simplified Elmore delay models, was found to be of limited value. Especially in the presence of arrival time pruning, the inner sub-problem is not that much smaller than the original optimization problem; however, it has to be

solved repeatedly until the multipliers corresponding to relaxed constraints converge. The one benefit of Lagrangian relaxation is that it naturally yields an increasingly refined estimate of the lower bound of the circuit's delay as the optimization proceeds.

8 Acknowledgments

The authors would like to gratefully acknowledge useful discussions with Prof. Martin D-F. Wong of the University of Texas at Austin and Prof. Charlie C-P. Chen of the University of Wisconsin at Madison. Members of the EinsTuner team including Ee Cho, David Ling, Walt Molzen, Ruud Haring, Phil Strenski, Abe Elfadel, Peter O'Brien, Greg Northrop, Pat Williams, Mike Henderson and Katya Scheinberg provided invaluable help with discussions, advice, device models, designer support, the Cadence interface, the SPECS API, optimization routines and so on, without which this work would simply not be possible. Greg Norththrop provided us with the results for Table 2 and Phil Strenski the results of Table 3. We are grateful to Ruud Haring for his helpful suggestions on the manuscript.

References

[1] M. D. Matson and L. A. Glasser, "Macromodeling and optimization of digital MOS VLSI circuits," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. CAD-5, pp. 659–678, October 1986.

[2] D. Marple, "Transistor size optimization in the Tailor layout system," *Proc. 1989 Design Automation Conference*, pp. 43–48, June 1989.

[3] A. Srinivasan, K. Chaudhary, and E. S. Kuh, "RITUAL: A performance driven placement algorithm for small cell ICs," *IEEE International Conference on Computer-Aided Design*, pp. 48–51, November 1991.

[4] J. P. Fishburn and A. E. Dunlop, "TILOS: A posynomial programming approach to transistor sizing," *IEEE International Conference on Computer-Aided Design*, pp. 326–328, November 1985.

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

[6] C. Visweswariah and 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*, pp. 244–251, November 1999.

[7] A. R. Conn, P. K. Coulman, R. A. Haring, G. L. Morrill and C. Visweswariah, "Optimization of custom MOS circuits by transistor sizing," *IEEE International Conference on Computer-Aided Design*, pp. 174–180, November 1996.

[8] A. R. Conn, C. Visweswariah, R. A. Haring and C. W. Wu, "Circuit optimization via adjoint Lagrangians," *IEEE International Conference on Computer-Aided Design*, pp. 281–288, November 1997.

[9] C. Visweswariah, R. A. Haring and A. R. Conn, "Noise considerations in circuit optimization," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, pp. 679–690, vol. 19, June 2000.

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

[11] C.-P. Chen, C. N. Chu, and D. F. Wong, "Fast and exact simultaneous gate and wire sizing by Lagrangian Relaxation," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. 18, July 1999.

[12] C. Visweswariah and R. A. Rohrer, "Piecewise approximate circuit simulation," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. 10, pp. 861–870, July 1991.

[13] C. Visweswariah and J. A. Wehbeh, "Incremental event-driven simulation of digital FET circuits," *Proc. 1993 Design Automation Conference*, pp. 737–741, June 1993.

[14] P. Feldmann, T. V. Nguyen, S. W. Director, and R. A. Rohrer, "Sensitivity computation in piecewise approximate circuit simulation," *IEEE Transactions on Computer-Aided Design of ICs and Systems*, vol. 10, pp. 171–183, February 1991.

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

[16] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, "Global convergence of a class of trust region algorithms for optimization with simple bounds," *SIAM Journal on Numerical Analysis*, vol. 25, pp. 433–460, 1988. See also same journal, pp. 764–767, volume 26, 1989.

[17] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, "A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds," *SIAM Journal on Numerical Analysis*, vol. 28, no. 2, pp. 545–572, 1991.

[18] A. R. Conn, L. N. Vicente, and C. Visweswariah, "Two-step algorithms for nonlinear optimization with structured applications," *SIAM Journal on Optimization*, vol. 9, pp. 924–947, September 1999.

[19] C. Lemaréchal and J. Zowe, "A condensed introduction to bundle methods in nonsmooth optimizations," in *Algorithms for continuous optimization: the state of the art* (E. Spedicato, ed.), no. 434 in NATO ASI Series C: Mathematical and Physical Sciences, (Dordrecht, The Netherlands), pp. 357–382, Kluwer Academic Publishers, 1994.

[20] G. L. Nemhauser and L. A. Wolsey, *Integer and combinatorial optimization*. New York: John Wiley and Sons, 1988. Chapter 6.