Gate Sizing for Constrained Delay/Power/Area Optimization

Olivier Coudert

Abstract— **Gate sizing has a significant impact on the delay, power dissipation, and area of the final circuit. It consists of choosing for each node of a mapped circuit a gate implementation in the library so that a cost function is optimized under some constraints. For instance, one wants to minimize the power consumption and/or the area of a circuit under some user-defined delay constraints, or to obtain the fastest circuit within a given power budget. Although this technology-dependent optimization has been investigated for years, the proposed approaches sometimes rely on assumptions, cost models, or algorithms that make them unrealistic or impossible to apply on real-life large circuits. We discusse here a gate sizing algorithm (GS), and show how it is used to achieve constrained optimization. It can be applied on large circuits within a reasonable CPU time, e.g., minimizing the power of a 10000 nodes circuit under some delay constraint in 2 hours.**

Keywords— **Gate sizing, discrete constrained optimization, delay/power/area tradeoff**

I. Introduction

From the practical point of view, gate sizing consists of optimizing the power and/or area under some delay constraints, or optimizing the delay possibly under some power and/or area constraints. The constraints can also include library-specific design rules, such as maximum fanout load or maximum transition time.

People have studied gate sizing since the late 70's [27], [17]. Using a RC delay model [24], the delay and the area are expressed with posynomials[1], as is it done in TILOS [10]. Geometric programming or heuristics based greedy approaches can be used to solve such a posynomial formulation [10], [30], [29]. Linear programming is used in [2] thanks to a piecewise linear delay model. A convex programming formulation based on pseudo-posynomial is presented in [29], and is solved using an interior point method. Gate sizing has been formulated as a convex programming problem [29], and solved using an interior point method. Gate sizing has also been formulated as non-linear programming in [4], [14], [18], and solved with Lagrangian multipliers [26, pp. 60–74] [16], [9], [12]. Analytical delay/power/area models or continuous resizing have been used in [27], [15], [29], [3] to avoid facing the combinatorial explosion, or to fill the lack of first and second derivatives.

These approaches suffer from problems that make them difficult or unrealistic to be applied on real-life circuits with a discrete size library: (1) some methods cannot take into account a delay constraint, which is not acceptable for industrial designs; (2) some methods make crude assumptions on the optimality criterion, e.g., assuming that minimizing a weighted power and delay product is the best power/delay tradeoff, while the problem is about *constrained* optimization; (3) the cost models, especially for delay and power, are not realistic, or are over-simplified to fit a specialized optimization technique; (4) some methods continuously size the gates, with the idea of solving an easier problem and then projecting the continuous solution on a discrete solution. But projective methods does not necessarily yield a feasible solution (i.e., which meets the given constraints); (5) some methods assume that the objective function and/or the feasible region is convex, which does not hold with accurate delay and power model; (6) some methods are too CPU costly to be applied on circuits with more than 1000 gates.

This paper addresses gate sizing as defined above. With the accurate delay model, this makes gate sizing a non-linear, non-convex, constrained, discrete, optimization problem. Experiences show that it is not even unimodal, i.e., many local extrema exist. A first concern, addressed in Section 2, is the cost models used for delay, power, and area: how accurate are they, and how CPU expensive are they to evaluate and update? Section 3 presents a constraint free delay optimizer built on top of GS, a gate sizing based general purpose optimizer [5]. Section 4 shows how power optimization under delay constraints is done with GS. Finally, Section 5 presents and discusses experimental data, and shows in particular that GS exhibits better performances than the widely used greedy approach.

II. Cost Models

We shortly present here the models used to evaluate the area, power, and delay of a circuit. We also discuss how costly updating these measures is when a local modification (e.g., resizing a gate) is applied on a circuit.

A. Area

Since the area of each gate is known, and the wire and routing area can be statistically estimated from the characteristics of the circuit, the area A of the mapped circuit can be accurately estimated. Updating the area after resizing is straightforward.

B. Delay

The most accurate delay estimation is a simulation from differential equations, e.g., with SPICE. However it is too much CPU expensive. Fig. 1 illustrates a more abstract, still accurate, delay model. The time t needed for a signal to propagate from the inputs of a gate to the inputs of

Olivier Coudert is with Synopsys, Inc., 700 East Middlefield Road, Mountain View, CA 94043. Email: coudert@synopsys.com .

[1]A posynomial is a polynomial with possibly negative exponent, e.g., $x^2 + 3x - 1/x + 8/x^3$

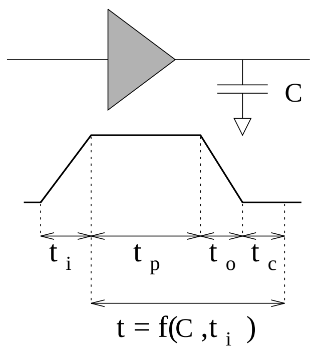

Fig. 1. Input slope and output load sensitive delay model.

the next gate is $t = t_p + t_o + t_c$, where t_p is the propagation time inside the gate, t_o is the output transition time, and t_c is the connection time spent in the wire. The delay t depends on the output load C, and on the input transition time t_i (the input transition time of a gate is the output transition time of its fanin gates). The reader is referred to [1], [32], [25], [21], [28], [19] for the presentation of some delay models. A study of different input transition time sensitive delay models shows that a table lookup approach is more accurate than multi-coefficient (linear, polynomial, or posynomial) approximations [19]. A table lookup based non-linear interpolation delay model is within 3% of SPICE, while 2 or 3-coefficient linear delay models can be off by as much as 20% from SPICE. We use such an accurate table lookup based non-linear delay model.

The designer specifies delay constraints between any two points of the circuit. Typically, he specifies the arrival time of the primary inputs and the required time of the primary outputs. Given a point n of the circuit, its *arrival time* $AT(n)$ is the time at which the signal is propagated from the primary inputs to n, and its *required time* $RT(n)$ is the time at which the signal must arrive to meet point-to-point delay constraints. Arrival and required times are defined for both rising and falling signals. They are computed by a forward and backward traversal of the circuit. The *slack* S of n is defined as $S(n) = RT(n) - AT(n)$. The set of points that have the minimal slack value is the *critical path* of the circuit, i.e., the slowest topological path. If the smallest slack is non negative, the delay constraints are met. The reader is referred to [8, pp. 225–289] for more details on delay computation, path sensitization, and false paths.

Sizing a gate affects the output load of its fanin gates, and the input transition time of its fanout gates. Consequently, the propagation times and transition times of its fanin gates and of its *transitive* fanout gates need to be recomputed. Moreover, resizing a gate can change the sensitivity of the paths, and therefore affects the slack of all the gates. In short, a single sizing can require a delay re-evaluation of the whole circuit, which is fairly expensive, even using incremental delay recomputation.

C. Power

The power P dissipated in a gate is:

$$P = P_{load} + P_{internal} + P_{leakage}.$$

The term P_{load}, the net dynamic power, is due to the charging and discharging of the output load of the gate. It depends on the toggle rate (number of transitions per unit of time) of the output net, and on the output load C. The term $P_{internal}$, the internal gate dynamic power, depends on the toggle rates, on the input transition time, and on the internal loads. The term $P_{leakage}$, the static leakage power, represents the static power dissipation in CMOS devices due to the leakage current. The toggle rates can be evaluated with expensive but accurate gate-level simulators [23], [7], or with fast but less accurate probabilistic methods [13], [7]. It is still difficult to take into account spatial and temporal correlation between the logical signals when dealing with sequential circuit [20], [31], [22]. The reader is referred to [23] for an overview on power estimation.

The toggle rate depends on how the signals propagate in the circuit. In particular, resizing a gate affects the *glitches*, i.e., the transitions due to signal races between two clock ticks. Measuring the glitches requires an event-driven simulation and an explicit knowledge of the inputs' waveforms. The later hypothesis is not realistic, and updating the amount of glitching with an event-driven simulation is far too costly. Thus we neglect the power dissipation due to glitching, and use a zero-delay model simulation to compute the toggle rates only once. In that case, updating the power after resizing is done efficiently.

III. Constraint Free Optimization

We now introduce GS, a general purpose gate sizing based optimization procedure [5]. We first show how constraint free optimization is done. We will show in the next section how GS is used for constrained optimization.

A. GS: General Purpose Optimization

Let us call a *move* a single gate resizing, which consists of replacing a node's gate g_0 with an equivalent gate g_1. Let us denote $\Delta Cost$ (call "gradient") the variation $Cost_1 - Cost_0$ of some cost function $Cost$ produced by this move.

As said in Section II, computing ΔA is straightforward. It is also the case for ΔP if one neglects second order components. On the other hand, computing ΔS can require a delay re-evaluation of the whole circuit. However, while a move can affect the slack of *every* nodes, its effect on the *slack gradients* decreases quickly in practice, approximatively geometrically by fanin and fanout level. For instance, a move that produces a slack variation of 1 on a node modifies the *gradients* of the immediate fanout gates of an order of 0.1, and of an order of 0.01 in the second level of fanout, etc. This enables us to use two heuristics:

(a) Instead of evaluating the gradient of a node *node* within the whole circuit, which is too computationally expensive in an iterative algorithm, we evaluate it

```
function  $GS(circuit, Cost, Fitness)$
$update$ = all the nodes of $circuit$;
$moves = \emptyset$;
loop {
    $old\_cost = Cost(circuit)$;
    foreach  $node \in update$  {
        Extract subcircuit $N$ around $node$;
        $node.move = 0$;
        $node.fit = Fitness(N)$;
        foreach  other gate $g$ implementing $node$ {
            $fit = Fitness(N[node \leftarrow g])$;
            if  $(fit > node.fit)$  {
                $node.move = g$;
                $node.fit = fit$;
            }
        }
        if  $(node.move \neq 0)$  {
            $moves = moves \cup \{node\}$;
        } else {
            $moves = moves - \{node\}$;
        }
    }
    $moved = ApplyMultiMove(circuit, Cost, moves)$;
    $update = PerturbedNodes(circuit, moved)$;
} until  $Converge(old\_cost, Cost(circuit), moved)$;
```

Fig. 2. The GS algorithm.

within a subcircuit N extracted around *node*, made of one or two transitive levels of fanin and fanout;

(b) After some moves has been performed, the gradient of a node *node* is re-computed only if *node* has been sufficiently perturbed, i.e., if one of its close neighbors has been resized.

We thus have a much cheaper gradient evaluation, and we avoid recomputing the gradients all the time. This CPU time vs. accuracy tradeoff has been experimentally validated.

Fig. 2 shows the gate resizing algorithm GS, which drives a circuit *circuit* to a local minimum of a cost function *Cost*. GS uses a fitness function *Fitness* to grade moves, the greater, the better. A good choice for constraint free optimization consist of taking $-\Delta Cost$ as the fitness function, since it picks the move that decreases *Cost* the most on a subcircuit. The set *update* contains the nodes whose fitness needs to be computed, and *moves* is the set of possible moves. For every node of *update*, the best fitness *node.fit* and its associated move *node.move* is computed w.r.t *Fitness*, using a subcircuit for the evaluation, as explained in (a). Then the function *ApplyMultiMove* takes the set *moves* of all candidate moves and determines a *multiple* move made of the maximal ordered subset of *moves* that minimizes *Cost*. This can be done in several way: along the descent direction [9]; by conjugation of directions [9]; by looking at the maximal subset of *moves* that minimizes *Cost*. The set *moved* of nodes that have actually been resized is returned, from which *PerturbedNodes*

```
function  $MaximizeSlack(circuit, S)$;
$best\_slack = -\infty$;
$GS(circuit, -S, \Delta S)$;
while  $S(circuit) > best\_slack$  {
    $best\_slack = S(circuit)$;
    $best\_solution = GatesOf(circuit)$;
    $GS(circuit, -TS, \Delta TS)$;
    $GS(circuit, -S, \Delta S)$;
}
$GatesOf(circuit) = best\_solution$;
```

Fig. 3. Delay optimization.

derives the new set of nodes whose fitness needs to be recomputed for the next iteration, as explained in (b). This process is iterated until some convergence criterion is met.

B. Constraint Free Delay Optimization

The problem here consists of maximizing the smallest slack of the design. The sensitivity of circuit's delay to resizing motivates a global optimization process as opposed to a local greedy search. Also delay optimization can encounter several local extrema because of the non-convexity of the delay model, which motivates a method that avoids suboptimal solution.

Let $S(circuit)$ be the smallest slack in the circuit *circuit*, and $TS(circuit)$ be the sum of the slacks of all its nodes. Fig. 3 shows the delay optimization procedure. An optimization and a perturbation step are iterated until no more improvement is found. The optimization step consists of maximizing the slack. The perturbation step is used to get out of the local minimum and look for another, potentially better, local minimum, and is indeed another optimization step, namely maximizing TS. Its effect is to *globally* speed up *every* nodes, so that the conflicts between the critical paths are *relaxed*, and the next maximization of $S(circuit)$ can achieve a better result[2].

GS cannot guarantee the optimality of the result since the cost function is not unimodal. To validate GS, we compared it with a greedy approach and three other non-linear optimization techniques [6]. This comparison shows that GS consistently beats the other methods for a comparable or smaller CPU time.

IV. Delay Constrained Power Optimization

We now show how delay constrained power optimization is done with GS. Delay constrained area optimization is done in a similar way. Delay optimization under a power constraint can be done using a dichotomous search based on delay constrained power optimization, which shrinks a delay interval solution until it is small enough.

Fig. 4 illustrates three constrained optimization techniques. Each figure shows a delay constraint as a vertical line that separates the feasible region (on the left) from

[2]We tried several perturbation functions, e.g., guarded randomization, (un)guarded sum of outputs' slacks maximization, but none of them were as good as maximizing $TS(circuit)$.

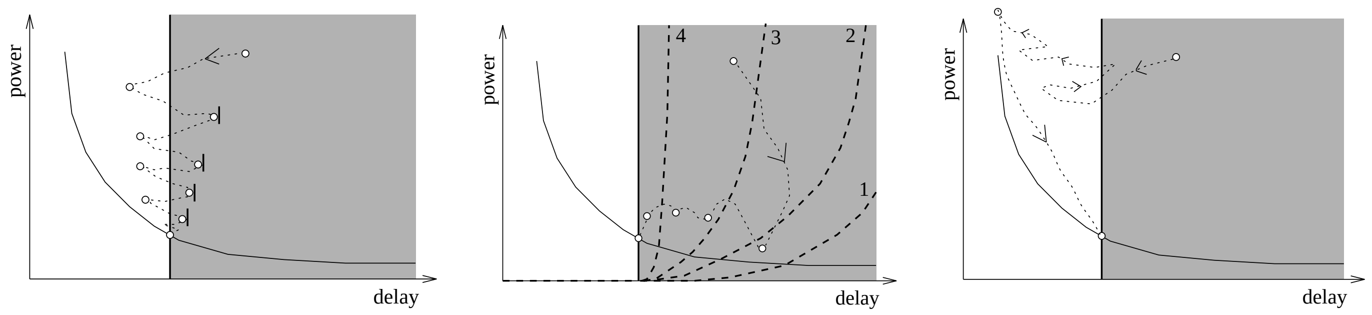

Fig. 4. Ping-pong, penalty function, and relaxation based constrained optimization.

the infeasible region (on the right, in gray). The optimal circuit, i.e., the circuit that minimizes the power consumption and meets the delay constraint is the intersection of this line with the optimal power/delay curve. The goal is to find a trajectory that drives an initial configuration to the optimal configuration.

Ping-ponging goes back and forth from the infeasible to the feasible region. First, one drives the circuit to the feasible region. Then, one minimizes the power with a loose delay constraint. This procedure is iterated and the delay constraint is made tighter at each iteration until it becomes the required delay constraint. One problem is how to tighten the delay constraint to obtain good quality results. Overall, although effective, ping-ponging is too much CPU time consuming, and so can be applied on small circuit only (< 500 nodes).

A penalty function based optimization method consists of adding to the objective function a penalty term *penalty*, which measures how much the constraints are violated. It is zero in the feasible region, and becomes greater when one goes deeper in the infeasible region. One iterates the minimization of $P + penalty$, and sharpen *penalty* at each iteration. The dashed curves in Fig. 4 show the penalty functions of the first four iterations. This constrained optimization method is very effective. However, it is also very difficult to tune (i.e., determining the penalty functions), and is too CPU intensive to be applied on large circuits.

We have chosen a relaxation method, illustrated on the right of Fig. 4. It consists of first, minimizing the constraint (i.e., maximizing the slack S), and then relaxing slowly the configuration within the feasible region. Fig. 5 shows the corresponding procedure. First, the delay is optimized[3]. Second, one minimizes the power P while enforcing the delay constraints by minimizing *Pcns* defined as:

$$Pcns(circuit) \quad = \quad \begin{cases} +\infty & \text{if } S(circuit) < 0 \\ P & \text{otherwise.} \end{cases}$$

The fitness function *Relax* is:

$$\begin{aligned} Relax &= \epsilon \qquad \text{if } \Delta P > 0 \text{ or } S_0 + \Delta S < 0 \\ Relax &= (\epsilon - \alpha \Delta P) \cdot \phi(\frac{\Delta S}{\epsilon + S_0}), \qquad \text{where} \end{aligned}$$

[3]If the delay constraints cannot be met, they are restated so that the power is optimized for the best found delay.

```
function $MinimizePowerUnderDelayCns(circuit, S)$;
$MaximizeSlack(circuit, S)$;
$GS(circuit, Pcns, Relax)$;
```

Fig. 5. Power optimization under delay constraints.

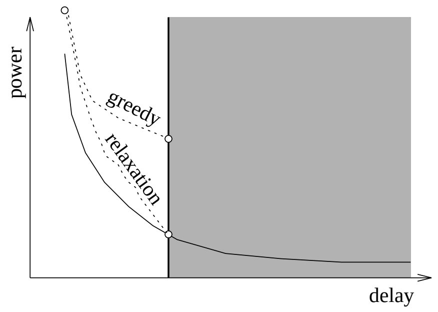

Fig. 6. Relaxation vs. Greedy

$$\phi(x) \quad = \quad \begin{cases} 1 + x & \text{if } x \geq 0 \\ \frac{1}{1-x} & \text{otherwise.} \end{cases}$$

In this formulation, S_0 is the current slack of the node under inspection. The small constant ϵ and the normalization constant α are precomputed according to the characteristics of the initial circuit. *Relax* balances the gain in power with a delay dependent function ϕ that acts as a benefit/penalty function. It takes into account how much power and slack is won or lost, and on how critical the node is. Fig. 7 shows how the fitness evolves w.r.t ΔP and ΔS for more and more critical nodes (i.e., for S_0 decreasing to zero).

This optimization method is based on two ideas. (1) Optimizing the delay gives plenty of alternatives for power optimization, i.e., going far away from the infeasible region makes power minimization less likely to be trapped in a local minimum. (2) The power optimization is done within the feasible region by relaxing the delay constraints using a penalty/benefit function, as opposed for instance to a greedy method that resizes as many non-critical nodes as possible to their minimal power. Such a greedy method can give low quality results for the following reason: resizing a few nodes to their local minimal power too "quickly" creates critical paths that prevent most of the other nodes

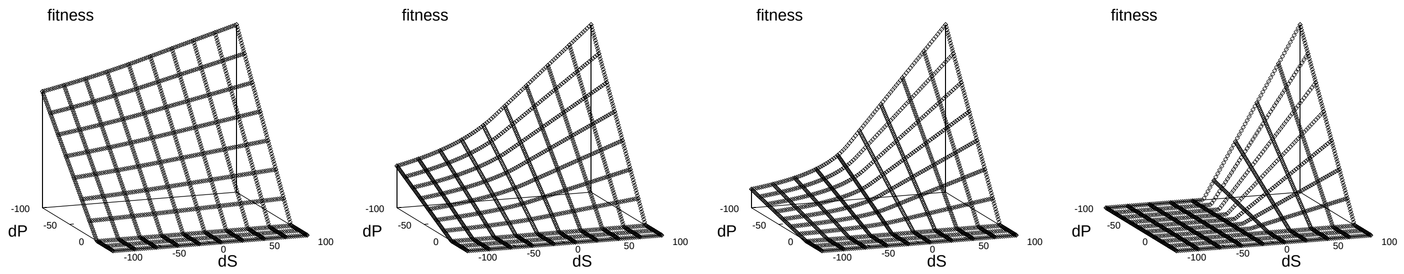

Fig. 7. From left to right, fitness for more and more delay critical nodes.

from being resized and saving more power. Fig. 6 illustrates this behavior. Greedy targets a minimum power configuration without taking into account how tight the constraint is, until its trajectory eventually hits the delay constraint line and gets stuck at this point, which can be far from the optimal configuration.

V. Experimental Results

We first compared GS to a greedy approach on 92 circuits mapped for delay. Table I shows some of these circuits, and summarizes the results obtained for delay optimization. GS found a better delay than greedy on 86 examples. The **%availability** measure is defined as follows. The available improvement is the difference between the initial delay and the best delay found by any algorithm. Algorithms earn a score based on how much of that available improvement they find, e.g., an algorithm that achieves all of the improvement on a circuit is awarded a 100%, and 0% if it does not find any improvement. **%availability** is the average of these scores over the 92 examples. Not only GS beats out the greedy approach, but also it is overall faster, especially when the circuit is large (> 3000 nodes).

We then let GS and a greedy approach optimize the power for different delay constraints, starting from a circuit in an optimal delay configuration. Fig. 8 shows some of the resulting optimal power/delay curves. It clearly demonstrates that a greedy approach can get stuck in a local minimal far from the optimal point, as explained in Section 5. Not only greedy is trapped in local minimums, but is also very dependent on the starting point: some of the curves produced by the greedy approach are locally increasing, and exhibit chaotic behaviors. Overall, GS's optimal power/delay curve is always better than greedy's one. The difference ranges typically between 1% and 7.5%, on average 5.2%, and is over 25% for some examples. GS beats out greedy for a CPU time only 10% larger.

Under a delay constraint equal to the initial delay, GS achieves a power saving of 11% on average, up to more than 30% on some large examples mapped on a rich library. The average CPU time is 1400 seconds for a 3400 nodes circuit, to be compared with an order of 1400 seconds for a 800 nodes circuit in [10], an order of 30000 seconds for a 800 nodes circuit in [29], and an order of 45000 seconds for a 1300 nodes circuit [11]. As shown by Table II, GS optimizes the power of fairly large circuits in a reasonable CPU time.

TABLE II
Power optimization under delay constraints.

Circuit.lib	**nodes**	**ave**	**%P**	**CPU**
*C*3540.*cbc*7_*hd*	757	3.723	11.4%	80
*C*3540.*ibm*_*m*5*l*	850	7.681	31.8%	276
*pair.ibm*_*m*5*l*	1349	7.652	27.9%	248
*C*7552.*ibm*_*m*5*l*	1377	6.691	20.4%	328
*C*7552.*cbc*7_*hd*	1505	4.412	7.1%	118
*des.cbc*7_*hd*	2579	3.943	8.5%	510
*des.ibm*_*m*5*l*	2789	7.042	30.5%	1523
*sgi*_*flat.cb*60*hd*230*d*	3995	3.799	13.3%	162
*bobbie.lca*300*k*	4575	5.181	5.3%	205
*F*642925.*cbc*7_*hd*	11447	4.497	7.8%	5793
*tandem.cmos*_*cba*	15061	4.315	13.4%	8622
*tandem.cb*60*hd*230*d*	16128	3.858	13.1%	7514
*tandem.ibm*_*m*5*l*	16181	7.907	57.3%	9242

%P is the power saving due to gate sizing. The **CPU** time is in seconds on a 60 MHz SuperSparc (85.4 SpecInt).

VI. Conclusion

Gate sizing is a technology dependent optimization that affects the quality of the final circuit. However, many of the approaches that have been explored are limited by the assumptions they rely on, or by the size of the circuit. We have presented GS, a general purpose gate resizing based optimization method. GS trades fitness accuracy against CPU time, which makes it possible to apply on large circuits. We have shown how constrained optimization, for example power minimization under delay constraint, is achieved with GS, using perturbations, multiple moves, and relaxations techniques, as opposed to single-move greedy approaches. This method produces better quality results on large circuits within a reasonable CPU time.

References

[1] D. Auvergne, N. Azemard, D. Deschacht, M. Robert, "Input Waveform Slope Effects in CMOS Delays", *IEEE Jour. on Solid-State Circuits*, **25**-6, pp. 1588–1590, Dec. 1990.
[2] M. Berkelaar, J. Jess, "Gate Sizing in MOS Digital Circuits with Linear Programming", Proc. of *EDAC'90*, pp. 217–221, 1990.
[3] M. Borah, R. M. Owens, M. J. Irwin, "Transistor Sizing for

TABLE I

GS VS. GREEDY FOR DELAY OPTIMIZATION.

Example					**Greedy**		**GS**	
circuit	**library**	**nodes**	**ave**	**space**	**%S**	**CPU**	**%S**	**CPU**
*C*2670	*cmos_cba*	674	1.95	10^{192}	6.93	7	7.14	16
*C*6288	*cmos_cba*	1040	2.59	10^{399}	16.19	91	16.20	96
*C*5315	*cmos_cba*	1001	3.62	10^{432}	2.88	6	4.20	39
pair	*cmos_cba*	1202	4.19	10^{610}	5.60	33	5.36	74
*C*3540	*ibm_m5l*	812	7.41	10^{620}	0.10	56	1.19	164
*C*7552	*cmos_cba*	1358	4.16	10^{695}	2.38	39	3.53	92
*C*6288	*lca*300*k*	1727	3.91	10^{730}	23.98	1216	24.75	891
des	*cb*60*hd*230*d*	2687	3.31	10^{1132}	5.67	349	7.23	440
*F*642925	*cb*60*hd*230*d*	11692	2.56	10^{3530}	25.55	2023	26.59	827
*F*642925	*cbc*7*_hd*	11479	3.09	10^{4088}	11.66	2918	11.89	814
*F*642925	*cmos_cba*	11408	3.28	10^{4472}	26.50	1084	27.17	664
*F*642925	*lca*300*k*	14136	3.65	10^{4708}	33.43	8604	33.31	1496
tandem	*cmos_cba*	15061	3.25	10^{6057}	24.93	5182	27.45	524
*F*642925	*ibm_m5l*	20225	3.59	10^{6896}	45.03	14177	48.47	8826
*F*642925	*ibm_m5l*	21174	3.64	10^{7375}	54.40	30813	57.43	13579
tandem	*ibm_m5l*	16181	5.91	10^{9484}	45.77	5468	48.21	2049
	minimum	674	1.91	10^{165}	0.00	6	0.18	13
	maximum	21174	7.39	10^{9484}	54.39	30813	57.45	13579
	average	3384	4.16	10^{2095}	8.72	1142	9.51	519
	%availability				72.21%		99.97%	

Each **circuit** is mapped for delay on a **library**. The table gives the number of **nodes** of the circuits, the **ave**rage number of different sizes per node, and the size of the configuration **space**. The table reports the delay improvement (**%S**) produced by **Greedy** and **GS**. The **CPU** time is in seconds on a 60 MHz SuperSparc (85.4 SpecInt).

Minimizing Power Consumption of CMOS Circuits under Delay Constraint", Proc. of *Int'l Symp. on Low Power Design*, pp. 167–172, April 1995.

[4] M. A. Cirit, "Transistor Sizing in CMOS Circuits", Proc. of *24th DAC*, June 1987.

[5] O. Coudert, "Gate Sizing: a General Purpose Optimization Approach", Proc. of *ED&TC'96*, Paris, France, March 1996.

[6] O. Coudert, R. Haddad, S. Manne, "New Algorithms for Gate Sizing: A Comparative Study", Proc. of *33rd DAC*, Las Vegas, June 1996.

[7] DesignPower, *Starter Kit*, Synopsys, Inc.

[8] S. Devadas, A. Ghosh, K. Keutzer, *Logic Synthesis*, McGraw-Hill, 1994.

[9] A. V. Fiacco, G. P. McCormick, *Nonlinear Programming*, John Wiley & Sons, NY, 1968.

[10] J. P. Fishburn, A. E. Dunlop, "TILOS: a Posynomial Programming Approach to Transistor Sizing", Proc. of *ICCAD'85*, pp. 326–328, Nov. 1985.

[11] J. P. Fishburn, "LATTIS: an Iterative Speedup Heuristics for Mapped Logic", Proc. of *29th DAC*, pp. 488–491, June 1992.

[12] R. Fletcher, *Practical Methods of Optimization*, John Wiley & Sons, 1987.

[13] A. Ghosh, S. Devada, K. Keutzer, J. White, "Estimation of Average Switching Activity in Combinational and Sequential Circuits", Proc. of *29th DAC*, Anaheim CA, June 1992.

[14] K. S. Hedlund, "AESOP: A Tool for Automated Transistor Sizing", Proc. of *24th DAC*, pp. 114–120, 1987.

[15] B. Hoppe, G. Neuendorf, D. Schmitt-Landsiedel, "Optimization of High-Speed CMOS Logic Circuits with Analytical Models for Signal Delay, Chip Area and Dynamic Power Dissipation", *IEEE Trans. on CAD*, **9**-3, pp. 236–246, March 1990.

[16] H. W. Kuhn, A. W. Tucker, "Nonlinear Programming", *2nd Berkeley Symposium on Mathematical Statistics and Probability*, J. Neyman Ed., Univ. of California Press, 1951.

[17] C. M. Lee, H. Soukup, "An Algorithm for CMOS Timing and Area Optimization", *IEEE Jour. of Solid-State Circuits*, **19**-5, pp. 781–787, Oct. 1984.

[18] D. P. Marple, "Transistor Size Optimization in the Tailor Layout System", Proc. of *26th DAC*, pp. 43–48, 1989.

[19] A. Martinez, "Automated Library Characterization and Timing Model Accuracy Issues when Interfacing to Different CAD Tools", Hewlett-Packard Company.

[20] J. Monteiro, S. Devadas, B. Lin, "A Methodology for Efficient Estimation of Switching Activity in Sequential Logic Circuits", Proc. of *31st DAC*, pp. 12–17, June 1994.

[21] Motorola HDC Series Design Manual.

[22] F. N. Najm, "Transition Density, a Stochastic Measure of Activity in Digital Circuits", Proc. of *28th DAC*, pp. 644–649, San Francisco CA, June 1991.

[23] F. N. Najm, "A Survey of Power Estimation Techniques in VLSI Circuits", *IEEE Trans. on VLSI Systems*, **2**-4, pp. 446–455, Dec. 1994.

[24] P. Penfield, J. Rubinstein, "Signal Delay in RC Tree Networks", Proc. of *2nd Caltech VLSI Conference*, pp. 269–283, March 1981.

[25] R. W. Phelps, "Advanced Library Characterization for High Performance ASIC", Texas Instruments, Inc.

[26] S. S. Rao, *Optimization: Theory and Applications*, Wiley Eastern Ld., 1978.

[27] A. E. Ruehli, P. K. Wolff, G. Goertzel, "Analytical Power/Timing Optimization Technique for Digital System", Proc. of *14th DAC*, pp. 142–146, June 1977.

[28] T. Sakurai, A. R. Newton, "Delay Analysis of Series-Connected MOSFET Circuits", *IEEE J. of Solid-State Cir.*, **26**-2, pp. 122–131, Feb. 1991.

[29] S. S. Sapatnekar, V. B. Rao, P. M. Vaidya, S. M. Kang, "An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Optimization", *IEEE Trans. CAD*, **12**-11, pp. 1621–1634, Nov. 1993.

[30] J. M. Shyu, A. Sangiovanni-Vincentelli, J. P. Fishburn, A. E. Dunlop, "Optimization-Based Transistor Sizing", *IEEE Jour. of Solid State Circuits*, **23**-2, pp. 400–409, April 1988.

[31] C.-Y. Tsui, M. Pedram, A. M. Despain, "Exact and Approximate Methods for Calculating Signal and Transition Probabilties in FSMs", Proc. of *31st DAC*, pp. 18–23, June 1994.

[32] G. Zewi, U. Barkai, Z. Becker, J. Ben-Simon, E. Kadar, "An Accurate Slope-Dependent Delay Model", Proc. of *TAU'90*, Haifa, Israel, 1990.

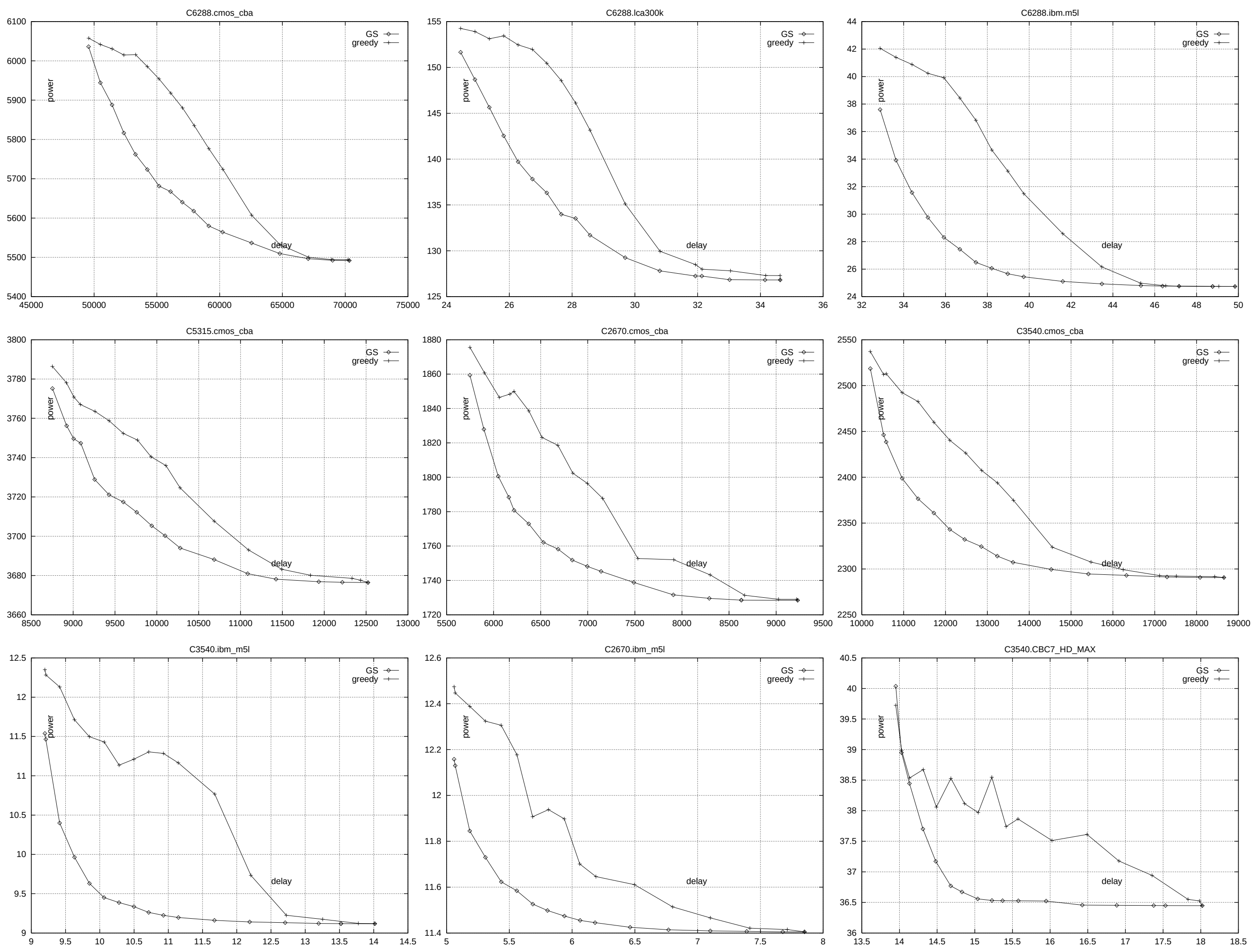

Fig. 8. Optimal power/delay produced by GS and greedy.

Olivier Coudert received his engineering and M.S. degrees in computer sciences and applied mathematics from Ecole Centrale de Paris (Paris, France) in 1987, and his Ph.D. degree in computer sciences from Ecole Nationale Supérieure des Télécommunications (Paris, France) in 1991. From 1988 to 1993 he was at Bull Research Center, where he worked on formal verification of sequential systems, reliability analysis, and logic synthesis. From 1993 to 1994 he was at DEC Paris Research Labs, where he worked on logic synthesis, symbolic computation, formal verification, and real-time software compilation. He joined Synopsys in 1994, where he has been working on logic synthesis, low power optimization, formal verification, and combinatorial optimization. He has (co)authored more than 40 international publications. His current interests include formal verification, combinatorial and symbolic optimization, and algebraic computation.
Dr. Olivier Coudert served on the technical program committee for several international conferences on CAD, including DAC, ICCAD, ED&TC, EDAC, and CAV. He is a member of the editorial board for *Formal Methods in System Design*, Kluwer Pub. He received a best paper award at ICCD'90, DAC'92, ED&TC'96, and DAC'96.