GRADIENT-BASED OPTIMIZATION OF CUSTOM CIRCUITS… (the second generation)

Chandu Visweswariah
IBM Thomas J. Watson Research Center
Route 134 and Taconic
Yorktown Heights, NY 10598, U.S.A.
'Phone: +1-914-945-3124
FAX: +1-914-945-3124
E-mail: *chandu@watson.ibm.com*

Outline

- Introduction and motivation
- Static vs. dynamic tuning
- Gradient computation (SPECS)
- Dynamic tuning (JiffyTune)
- Static tuning (EinsTuner)
- Future work and conclusions

Acknowledgments

- SPECS
  - Payam Heydari
  - David Ling
  - Walt Molzen
  - Nam Nguyen
  - Bill Wright
  - Chagarn Whan
  - ...
- JiffyTune
  - Andy Conn
  - Paula Coulman
  - Ruud Haring
  - Chai Wah Wu
- EinsTuner
  - Andy Conn
  - Abe Elfadel
  - Peter O'Brien
  - Phil Strenski

Introduction & motivation

- Better (optimal) circuits
- Productivity
  - avoid tedious, error-prone, slow, manual tuning
  - explore tradeoffs
- Design re-use
  - particularly during design re-mapping
- Manufacturability

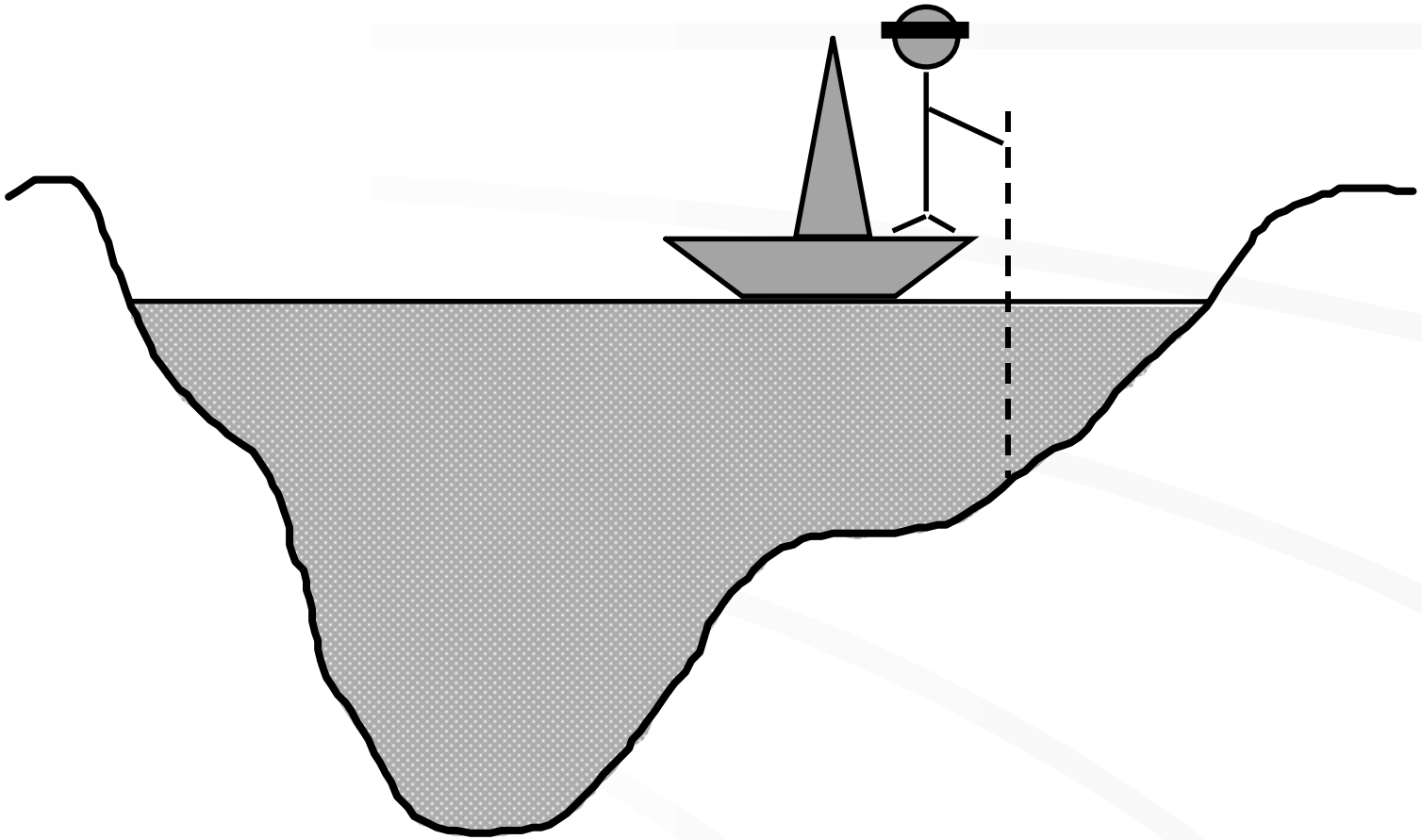
Static vs. dynamic tuning: *similarities*

- Continuous optimization only
- Applicable to custom circuits
- Use formal gradient-based nonlinear optimization
- Use incremental time-domain sensitivity computation
- Proof of convergence to a local minimum
- Wire tuning dovetails nicely

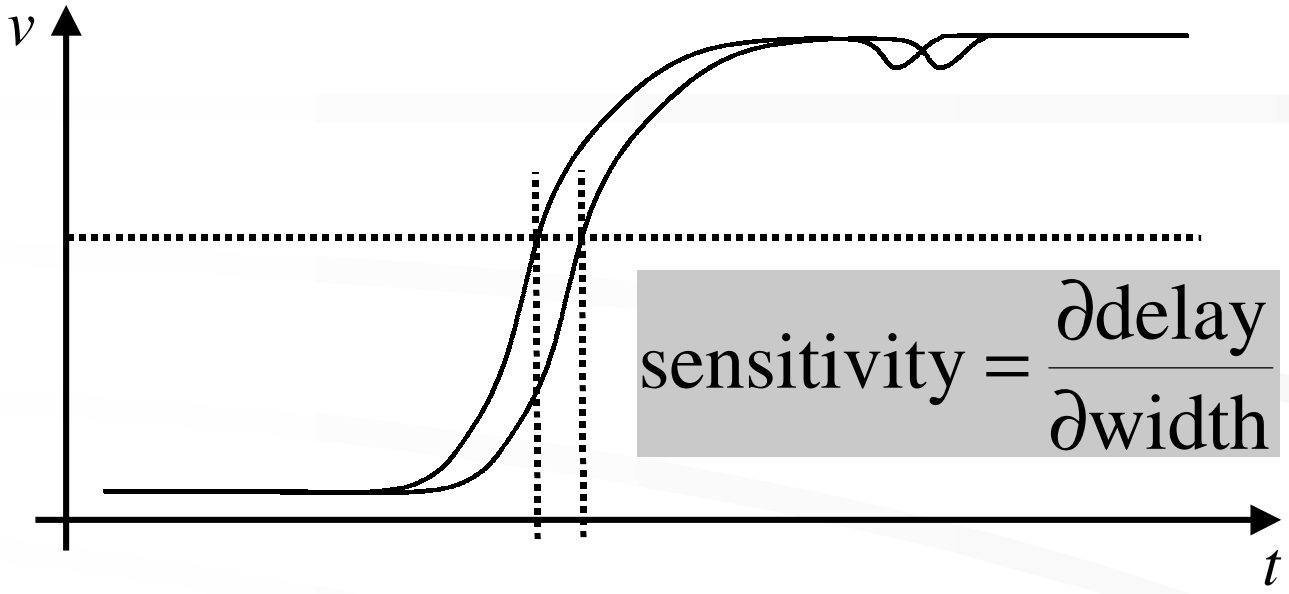
Static & dynamic tuning: *differences*

Dynamic (JiffyTune)	**Static (EinsTuner)**
Need input waveforms & careful specification of problem.	No input waveforms needed; problem easy to specify.
Considers only paths that are sensitized, measured and constrained; minimax allowed.	Considers all paths.
Not susceptible to false path problem.	Susceptible.
No inherent pessimism.	Pessimism inherent to static timing analysis.
Capacity of 20K FETs proven; can handle noise, power criteria.	Capacity unknown; no noise and power capabilities.
Typically applied to arithmetic & array circuits, library design.	Typically applied to control random-logic macros, but could apply to any circuitry.

Gradients/sensitivities essential



Part I: Gradient computation



- Incremental, accurate (to guide optimizer)
- Fast, millions computed per timing run
- Chain-rule and combine all aspects of device or wire model

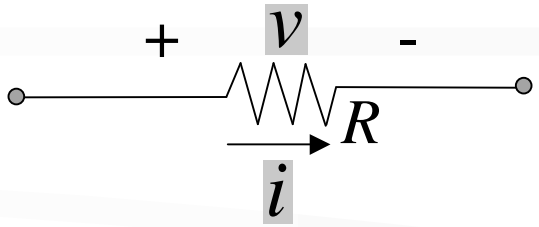
Direct and adjoint methods

$$\begin{array}{c} \\ p_1 \\ p_2 \\ \vdots \\ p_m \end{array} \begin{array}{c} \begin{array}{cccc} f_1 & f_2 & \cdots & f_n \end{array} \\ \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \dfrac{\partial f_i}{\partial p_j} & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} \end{array}$$

- Direct method: one row of Jacobian per sensitivity analysis
- Adjoint method: one column at a time
- Adjoint method incurs convolution overhead

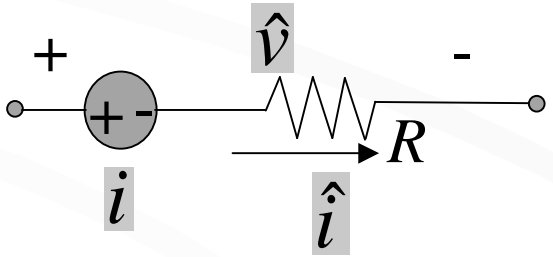
Direct method

$$v = iR$$



$$\frac{\partial v}{\partial R} = i + R\frac{\partial i}{\partial R}$$

$$\hat{v} = i + R\hat{i}$$

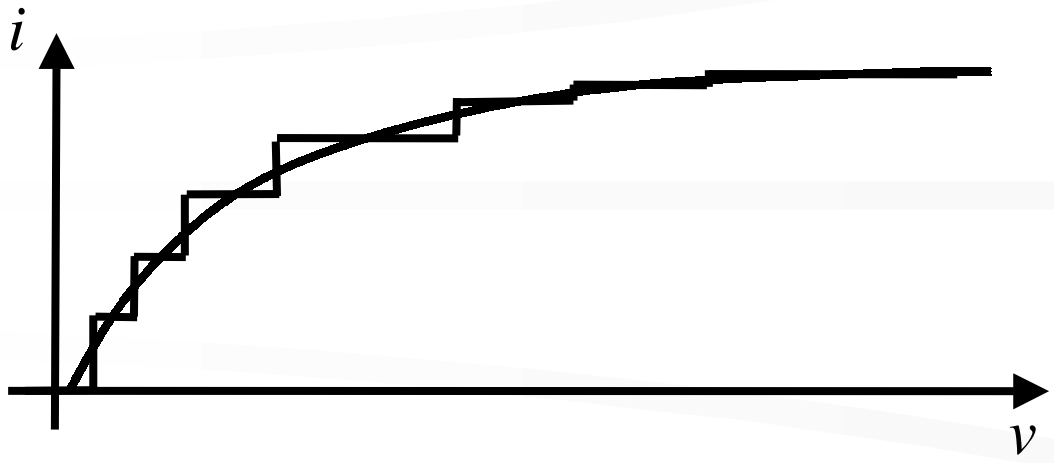


- Re-use LU factors of nominal solution

Adjoint method

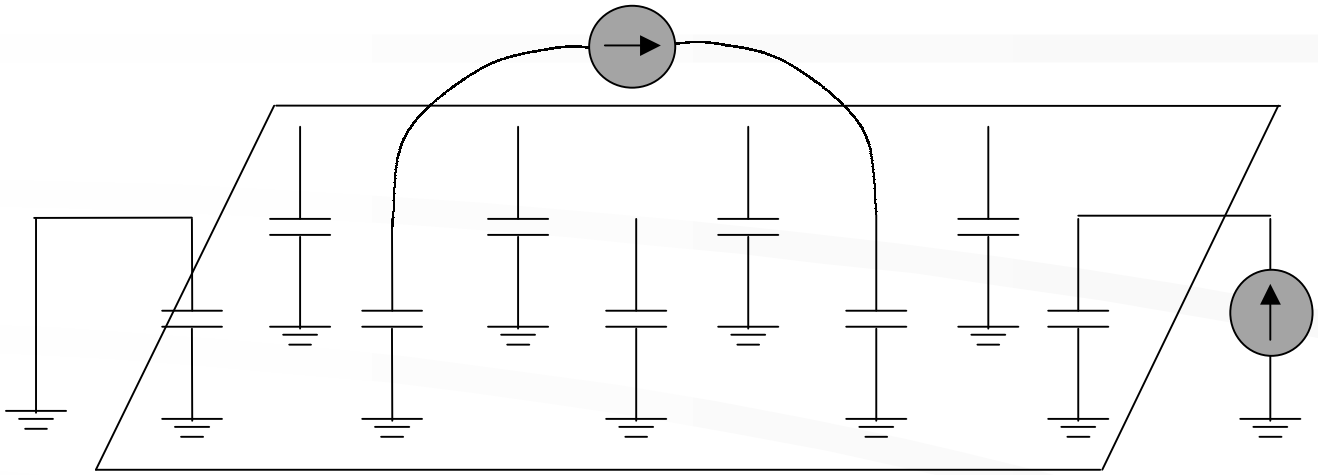
- Best expounded via Tellegen's theorem
- All parameters, one function per analysis
- Time and control are reversed
- Re-use LU factors of nominal circuit to solve adjoint circuit at each time point
- Convolve nominal and adjoint waveforms to obtain sensitivity
- Method of choice for large circuits!

Fast simulation with SPECS



- Simplified device models
  - piecewise constant *i*-*v* tables
  - lumped, linear, grounded capacitors
- Event-driven simulation
- Specialized integration for steady-state
- 70x SPICE; 5% relative timing accuracy

Sensitivity computation in SPECS



- Sensitivity and adjoint circuits consist of disconnected capacitors
- At event times, charge transfer occurs
- Event-driven solution and convolution

Sensitivity performance

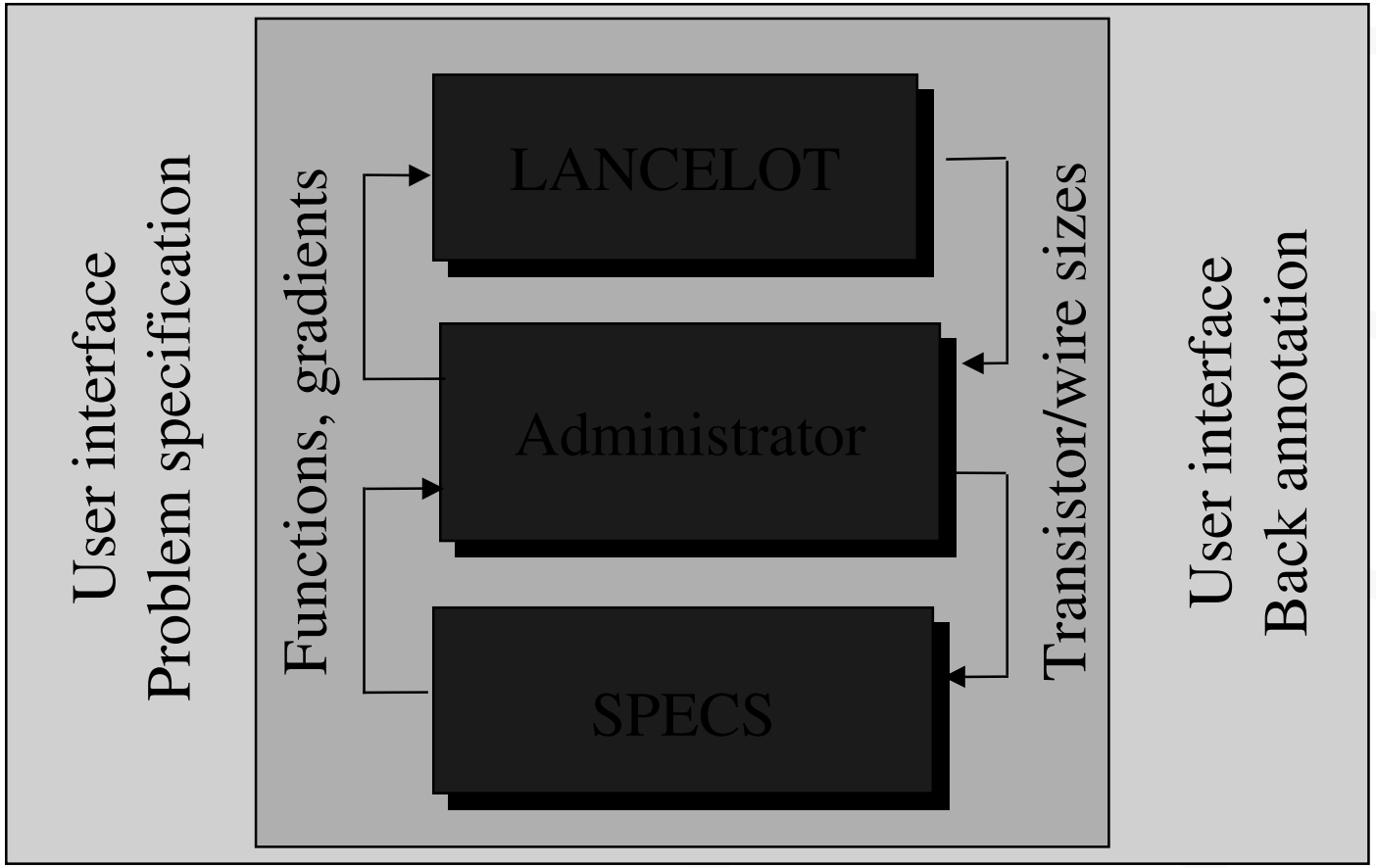
- IOmux circuit (LPM chip): 6,900 FETs, 770 RC wires, 57 functions, 6,114 parms., 348,498 gradients

	Direct method	Adjoint method	?
Gradient computation CPU time (s)	39,041 (10.8 hours)	635.4 (10.6 minutes)	?
Total run time (s)	39,223	827.1	?
Run time overhead per sens. circuit solution	3.3% (6,114 solutions)	5.8% (57 solutions)	?
Overhead per gradient	0.06%	0.001%	?
Run time per gradient	112 ms	1.82 ms	?

Part II: JiffyTune [ICCAD '96, TCAD '98]

- Given
  - an initially sized working circuit schematic
  - input waveforms
  - a precise statement of the tuning problem
- Determine
  - optimal transistor and wire sizes
- While supporting…
  - flexible objective functions and constraints, DFM
  - ratio-ing and grouping
  - delay, rise/fall time, power, area, noise, minimax

JiffyTune: software organization



Features of JiffyTune at a glance

- Simulation (fast simulator SPECS)
  - adjoint Lagrangian gradient computation
  - noise optimization
- Optimization (nonlinear optimizer LANCELOT)
  - customization
  - two-step updates
  - failure recovery
- User interface (Cadence)
  - front-end and back-annotation

Adjoint Lagrangian [ICCAD '97]

- LANCELOT's merit function is

$$\Phi = f(x) + \sum_i \lambda_i c_i(x) + \frac{1}{2\mu} \sum_i [c_i(x)]^2$$

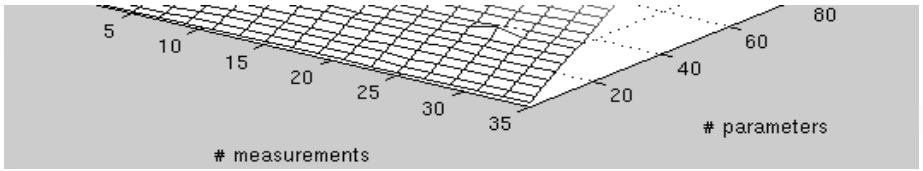
- Key observation: merit function is a scalar
- All the gradients of any scalar can be computed by means of a single adjoint analysis!
- In practice, apply scaled versions of all adjoint excitations at once

Sensitivity performance

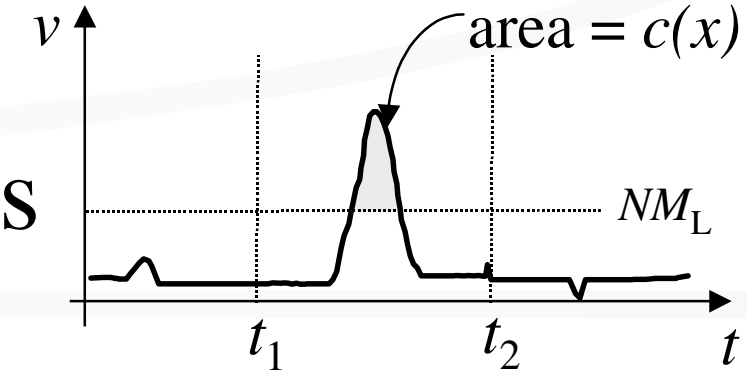
- IOmux circuit (LPM chip): 6,900 FETs, 770 RC wires, 57 functions, 6,114 parms., 348,498 gradients

	Direct method	Adjoint method	Adjoint Lagrangian
Gradient computation CPU time (s)	39,041 (10.8 hours)	635.4 (10.6 mins.)	21.9 (21.9 secs.)
Total run time (s)	39,223	827.1	213.6
Run time overhead per sens. circuit solution	3.3% (6,114 solns.)	5.8% (57 solns.)	11.4% (1 soln.)
Overhead per gradient	0.06%	0.001%	0.00003%
Run time per gradient	112 ms	1.82 ms	0.0063 ms 63 μs!

Complexity analysis

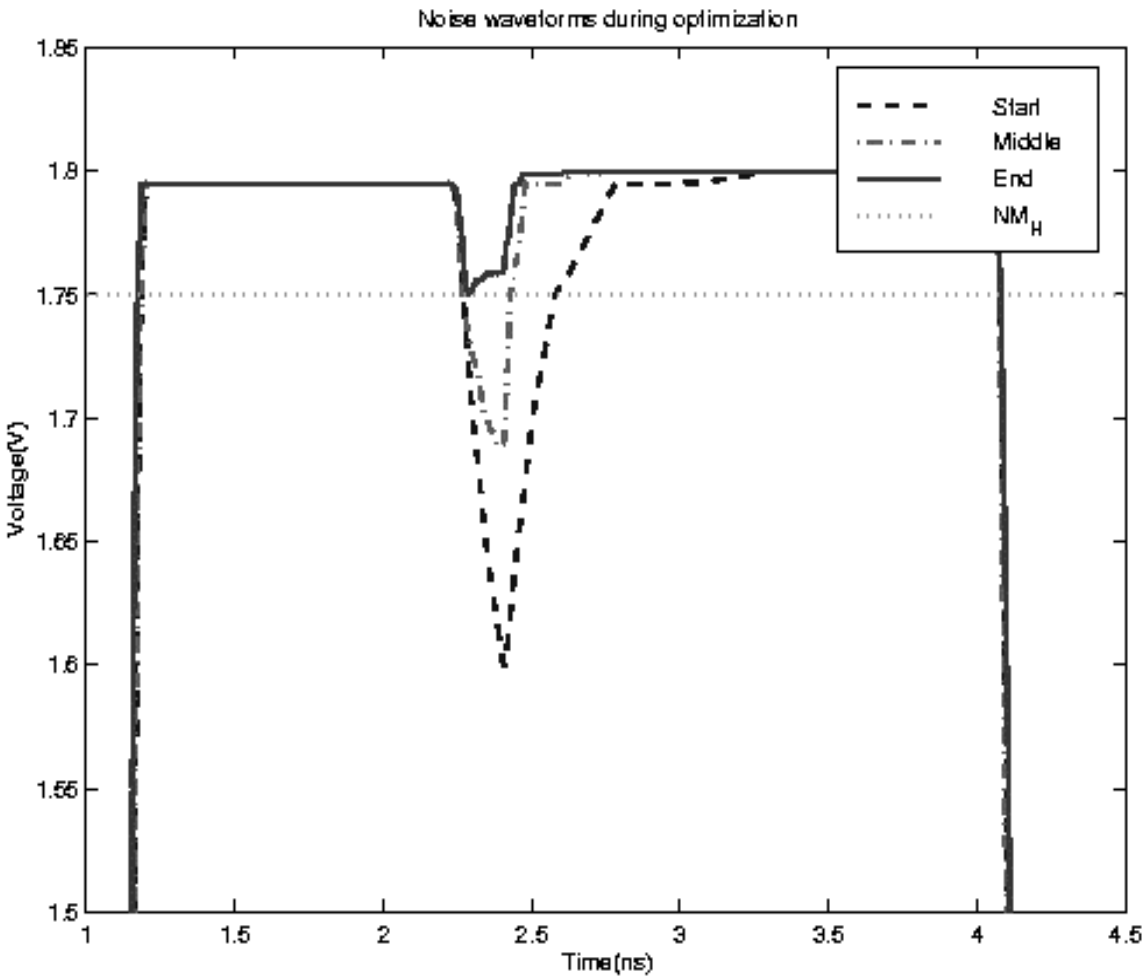


Noise considerations



- Trick: remap infinite number of constraints to a single integral constraint $c(x) = 0$
- In adjoint Lagrangian mode, any number of noise constraints almost for free!
- General (constraints, objectives, minimax)
- Tradeoff analysis for dynamic library cells

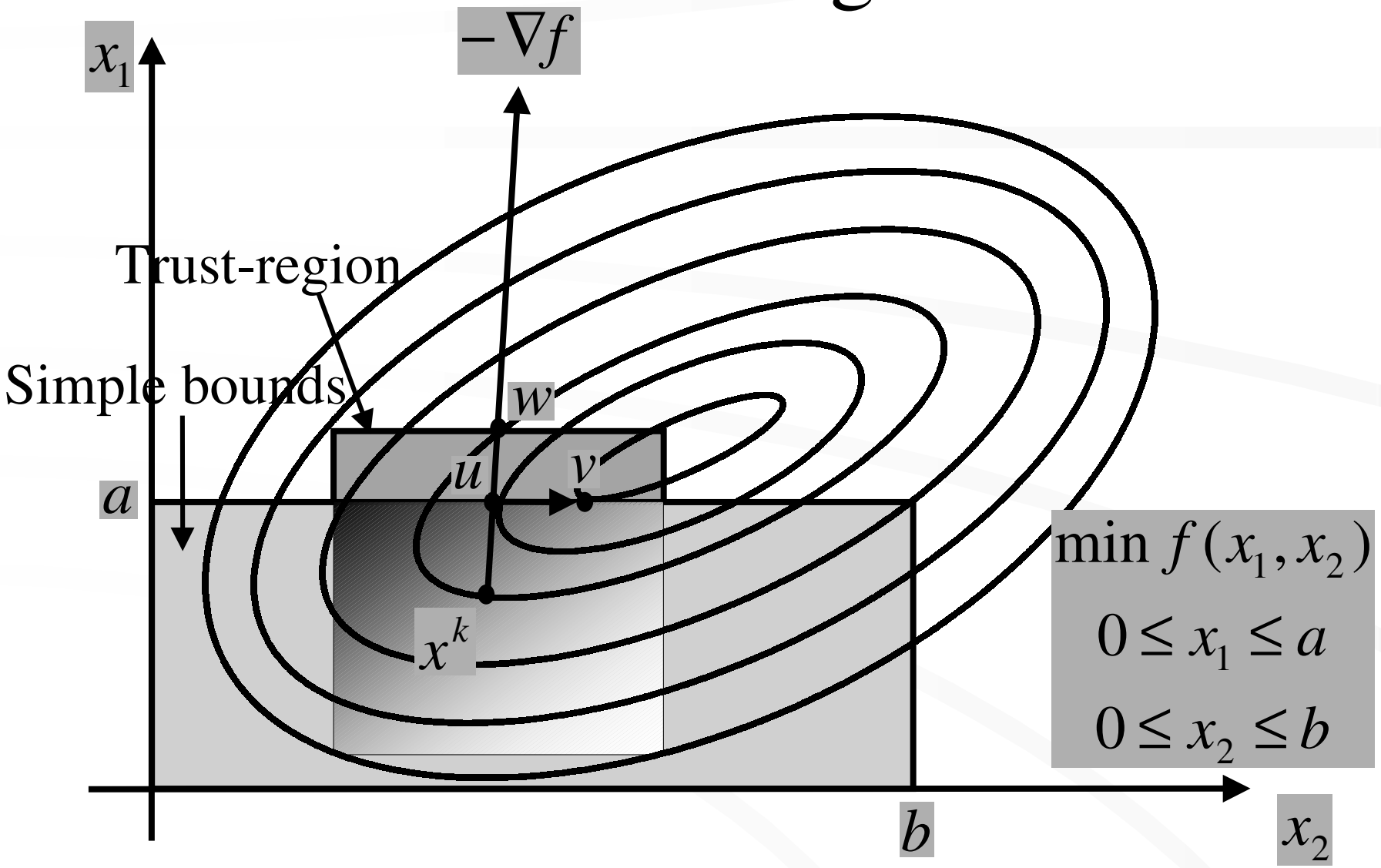
Noise considerations



Nonlinear optimization (LANCELOT)

- Large-scale, general-purpose, nonlinear optimizer
- Trust-region algorithm
- Augmented Lagrangian merit function
- Especially efficient for large problems with nonlinear constraints
- Handles simple bounds easily and effectively by means of projections
- Many options available

LANCELOT algorithm



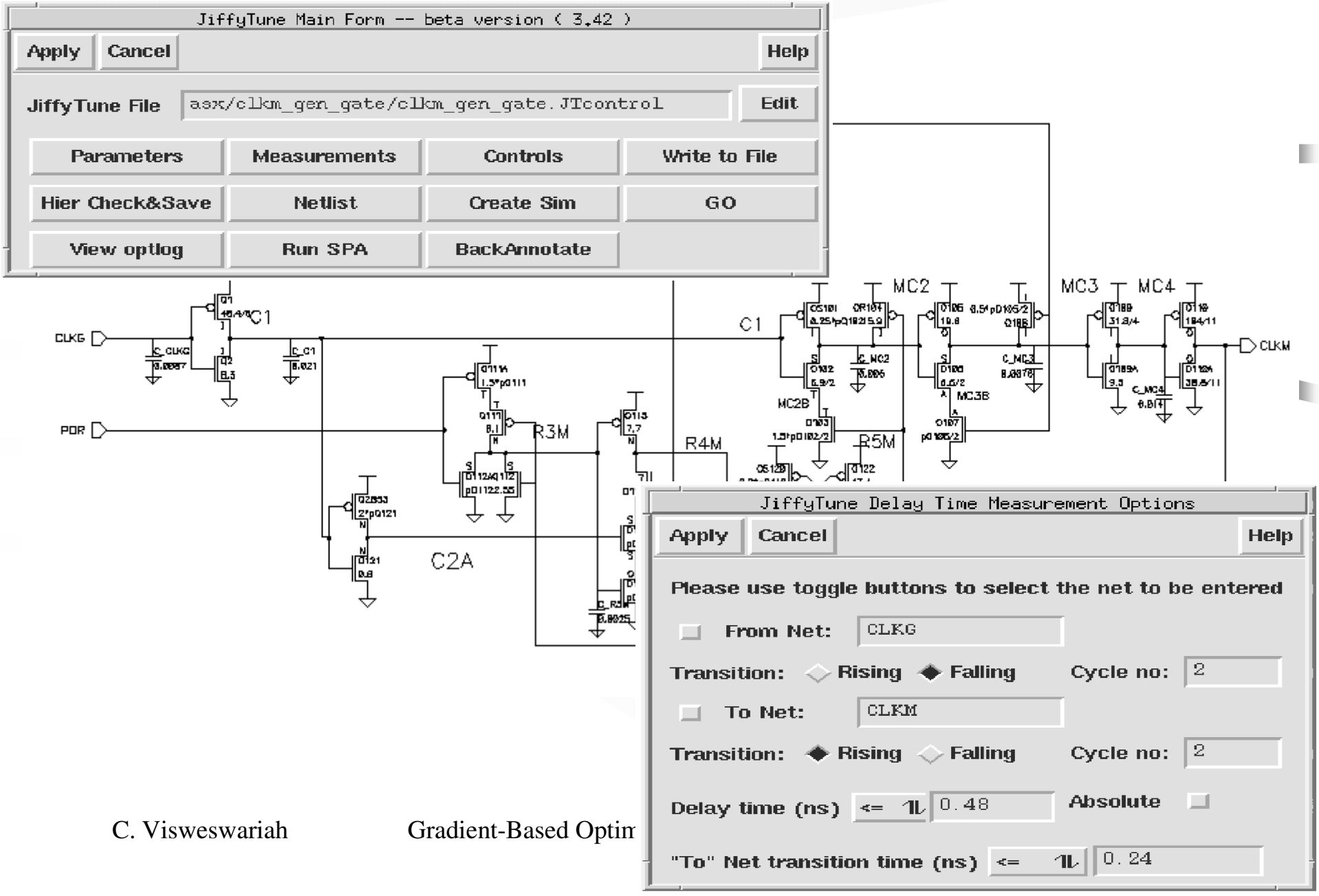
Customization of optimizer

- Choice of tolerances and stopping criteria to deal with noisy data
- Two-step updates [submitted to SIAM J. Opt.]
- Failure recovery
- Close communication between simulator and optimizer for adjoint Lagrangian mode
- Specialized Hessian rank-one updates

User interface

- Good tools are useless without a good human interface
- Cadence interface developed by actual custom designer
- Visual and interactive in familiar environment
- Allows problem specification by pointing and clicking; back-annotates tuning results
- Stores tuning criteria as attributes of the schematic for design re-use

User interface



User experience

- Chips that have used JiffyTune
  - 630fp and 630fp+ (Austin)
  - LPM chip (Yorktown)
  - HARP DASD controller (Yorktown)
  - BlueFlame (Poughkeepsie/Yorktown)
- About 500 schematics tuned
- Two case studies will be described

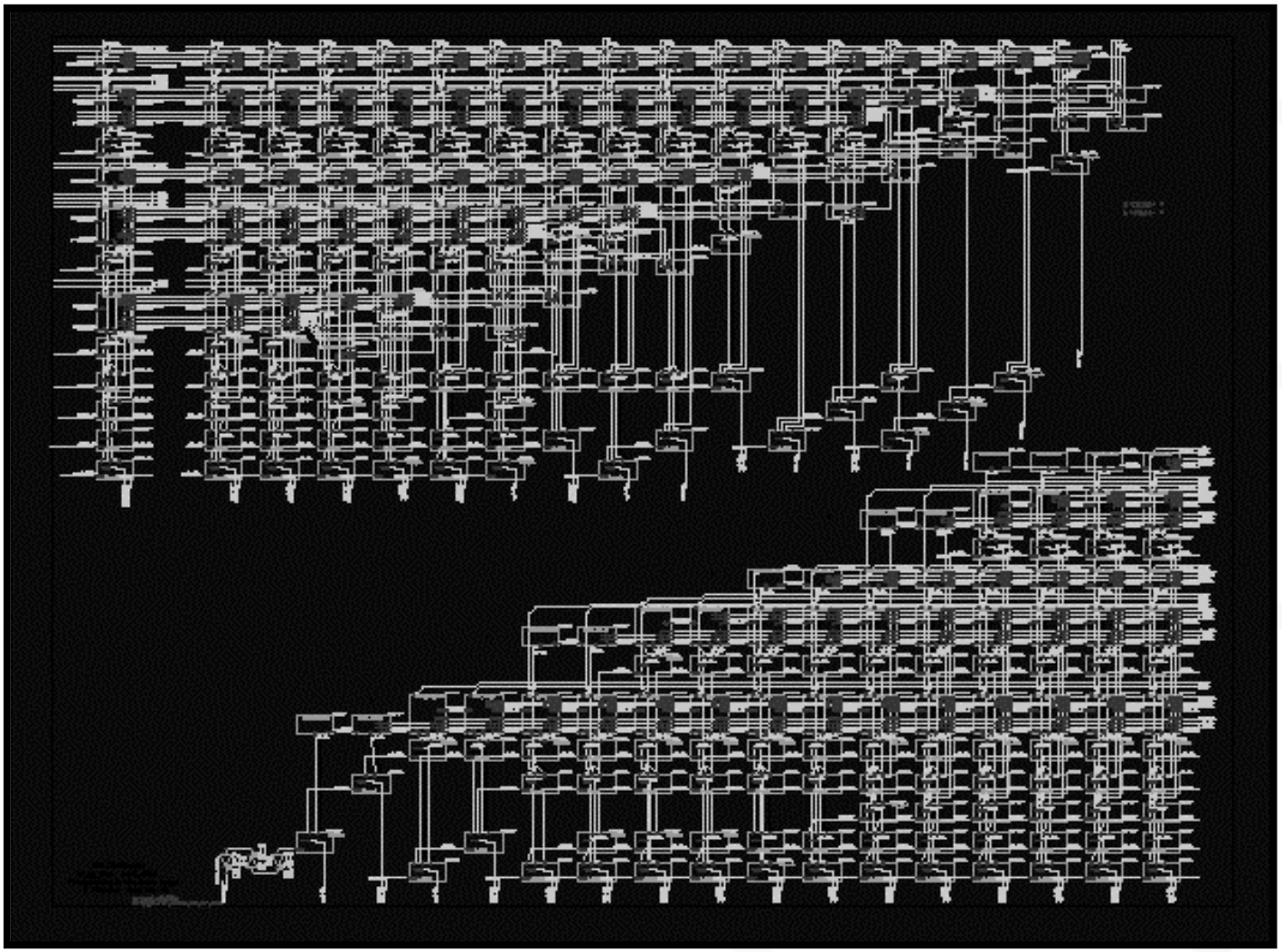
Case study #1: Booth encoder

- Booth encoder for BlueFire is a tricky circuit (large fanouts, plenty of pass gates, many nasty paths, tight delay constraints)
- “Wouldn’t want to guess device sizes...” (Peter Cook, Yorktown)
- Tuned: 13-way minimax + area constraint
- Savings of 28% realized in delay
- JT runs were done in an hour (would take several days with PathMill + hand-tuning)

Case study #2: compressor tree

- 8:2 compressor for 16x32 multiplier in BlueFire FXU, designed by Bruce Fleischer, Yorktown
- Initially, topology selections were made and cells were hand-tuned based on SPICE
- JiffyTune application #1 improved delay 6.4%
- JiffyTune application #2 after topology changes gained a further 6.1%
- Second phase of tuning completed in a day; ease of use exploited

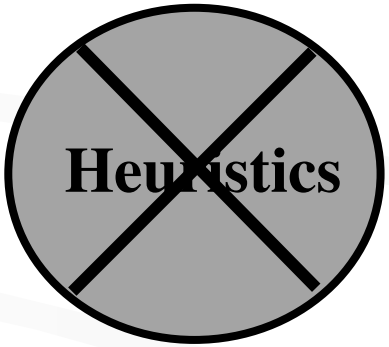
16x32 multiplier



8:2 compressor tree cross-section

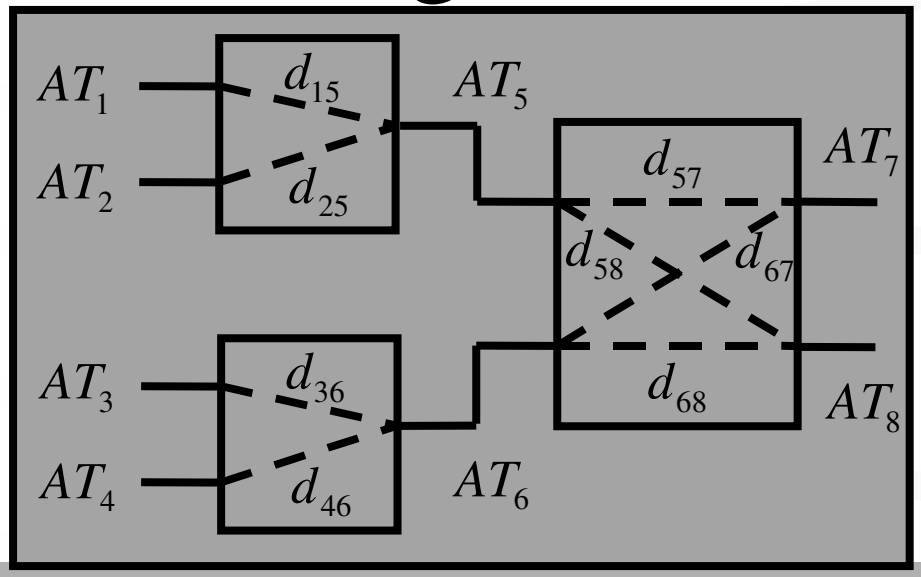
Static tuning - a formal approach

- Most approaches are heuristic (TILOS, AMPS)
  - no proof of convergence
  - not gradient-based



- Our approach (EinsTuner)
  - a formal gradient-based optimization technique
  - proof of convergence
  - ability to tune custom circuits
  - accurate (based on real simulation)

Static tuning formulation



$$\min[\max(AT_7, AT_8)]$$

$$\text{s.t.} \quad AT_7 = \max[AT_5 + d_{57}, AT_6 + d_{67}]$$

$$\text{s.t.} \quad AT_8 = \max[AT_5 + d_{58}, AT_6 + d_{68}]$$

$$\text{s.t.} \quad AT_5 = \max[AT_1 + d_{15}, AT_2 + d_{25}]$$

$$\text{s.t.} \quad AT_6 = \max[AT_3 + d_{36}, AT_4 + d_{46}]$$

Minimax optimization

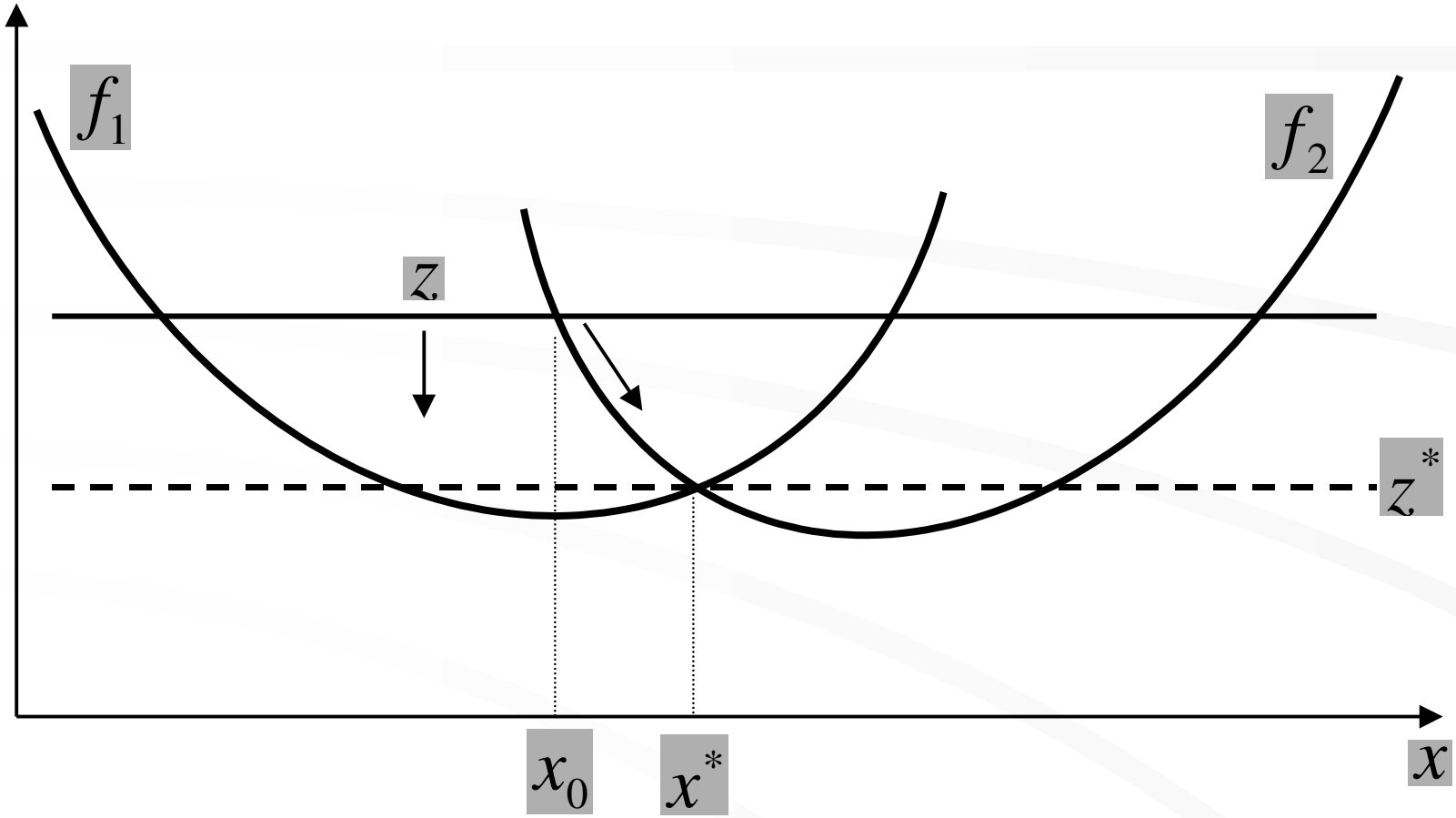
$$\min_{x}[\ \max\{f_1(x),\ f_2(x)\}\ ]$$

- This problem is not smooth!
- Re-map to:

$$\min_{x,z}\ z$$
$$\text{s.t. } z \geq f_1(x)$$
$$\text{s.t. } z \geq f_2(x)$$

- This trick converts a minimax problem into a standard twice continuously differentiable nonlinear optimization problem

Minimax optimization



Remapped problem formulation

$$\min[\max(AT_7, AT_8)] \Rightarrow \begin{cases} \min z \\ \text{s.t. } z \geq AT_7 \\ \text{s.t. } z \geq AT_8 \end{cases}$$

$$AT_7 = \max[AT_5 + d_{57}, AT_6 + d_{67}] \Rightarrow \begin{cases} AT_7 \geq AT_5 + d_{57} \\ AT_7 \geq AT_6 + d_{67} \end{cases}$$

$$AT_8 = \max[AT_5 + d_{58}, AT_6 + d_{68}] \Rightarrow \begin{cases} AT_8 \geq AT_5 + d_{58} \\ AT_8 \geq AT_6 + d_{68} \end{cases}$$

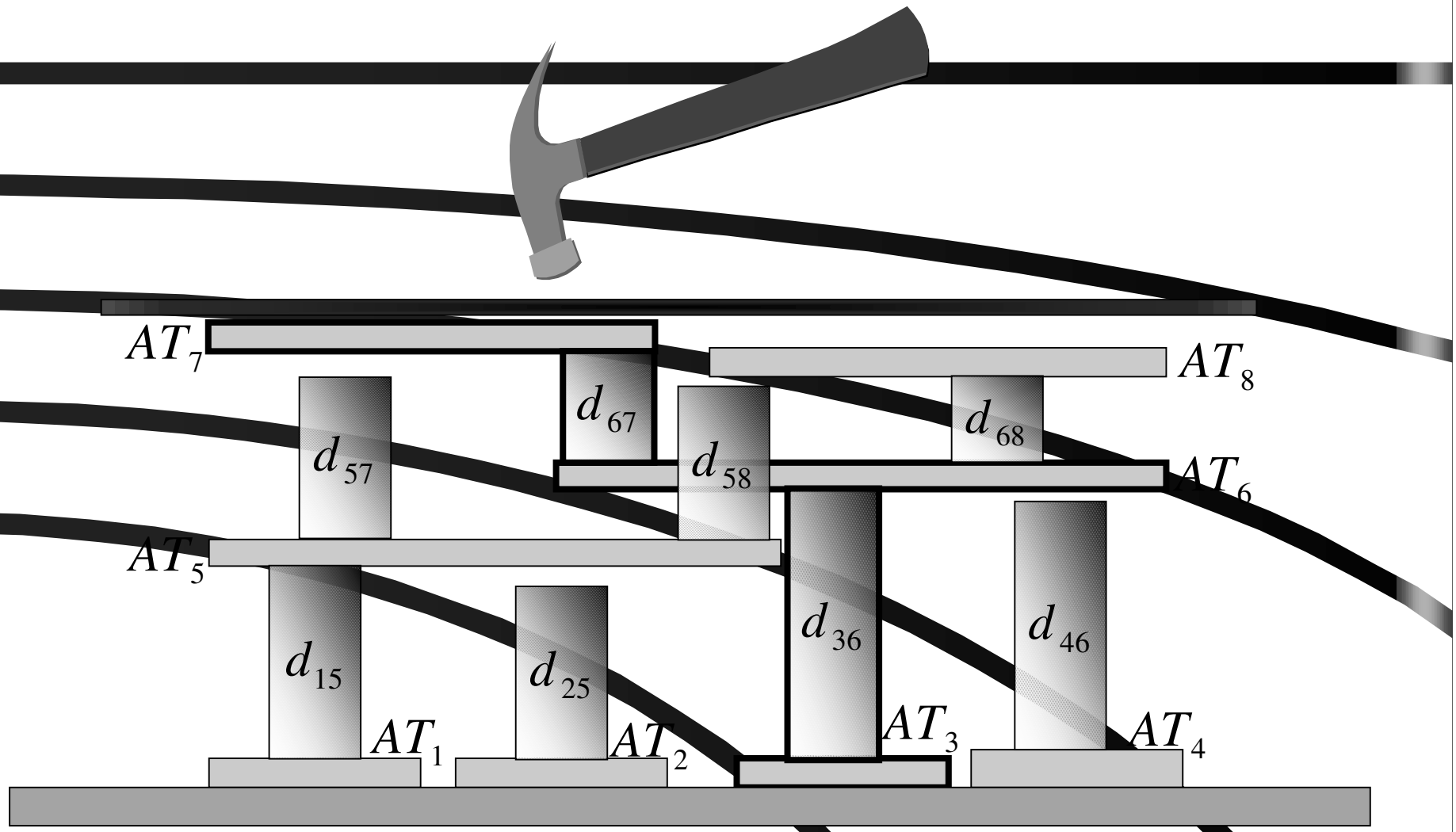
$$AT_5 = \max[AT_1 + d_{15}, AT_2 + d_{25}] \Rightarrow \begin{cases} AT_5 \geq AT_1 + d_{15} \\ AT_5 \geq AT_2 + d_{25} \end{cases}$$

$$AT_6 = \max[AT_3 + d_{36}, AT_4 + d_{46}] \Rightarrow \begin{cases} AT_6 \geq AT_3 + d_{36} \\ AT_6 \geq AT_4 + d_{46} \end{cases}$$

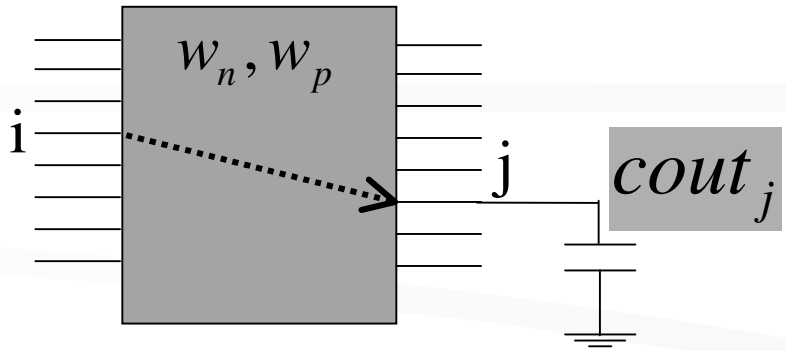
Reformulated problem

- *ATs* are variables of the problem
- At the solution
  - original problem is solved because by definition z is minimized and the constraints are feasible
  - *ATs* on critical path are correct because these constraints are tight
  - d_{ij}'s are functions of transistor widths

Brick and planks analogy



Constraint generation



$$AT_j^r \geq AT_i^f + d_{ij}^r(w_n, w_p, cout_j, slew_{in}^f)$$

$$AT_j^f \geq AT_i^r + d_{ij}^f(w_n, w_p, cout_j, slew_{in}^r)$$

$$slew_j^r \geq s_{ij}^r(w_n, w_p, cout_j, slew_{in}^f)$$

$$slew_j^f \geq s_{ij}^f(w_n, w_p, cout_j, slew_{in}^r)$$

- Each delay, slew depends on one input slew

Mechanics of tuning

- Divide circuit into CCCs
- Traverse topology, generate objective functions & constraints
- Feed problem to optimizer (LANCELOT)
- LANCELOT requests function, gradient evaluation for specific CCCs (SPECS via an API)
- Run optimization to convergence
- Check feasibility
- Run final static timing analysis

Aids to convergence

- Initialization of multipliers and variables
- Scaling, choice of units
- Choice of simple bounds on arrival times, z
- Reduction of noise
- Failure recovery
- Two-step updates
- Reduction of dimensionality
- Exploiting of group partial separability
- Tuning of LANCELOT to be aggressive

Results: experimental procedure

- Step 1: run heuristic gain-based sizing with gain set to 4.0
- Step 2: determine area at the end of the heuristic sizing
- Step 3: run tuning with heuristic sizing as start point, with area constrained to be the same as after heuristic sizing
- Step 4: report the improvement in critical delay using identical area as heuristic sizing

Results: all the details

Name	Size					Critical path delay				Arr. time infeas.		Slew infeas.		# its.	Proj. gradient		CPU time s
	# Gate	# Tx	# No	# Var	# Con	Area μm	Heur. ps	Final ps	%	W ps	Av ps	W ps	Av ps		Beg	End	
nor2	1	4	3	15	11	100.0	101	84	16.8	2e-2	3e-3	1e-3	3e-4	19	380	6e-2	10.5
inv3	3	6	4	23	15	69.7	175	127	27.4	2e-3	3e-4	1e-3	3e-4	70	120	7e-2	35.8
c17	7	28	12	63	61	345.6	241	179	25.8	2e-2	2e-3	9e-2	4e-3	58	290	0.3	152.4
a3_3	9	34	12	67	75	549.9	318	245	22.8	0.2	2e-2	0.5	3e-2	48	380	0.3	159.0
full_ adder	14	46	17	97	97	307.8	398	306	23.2	0.15	7e-3	6e-3	3e-4	83	290	0.2	339.0
iqia_ mac	22	72	30	165	147	762.2	274	243	11.5	2e-2	1e-3	4e-3	1e-4	115	330	7e-2	699.0
c8	180	584	208	1193	1203	3187	918	719	21.6	0.3	2e-3	0.5	4e-3	123	380	2.9	6816
c432	299	960	335	1939	1929	2477	1709	1369	19.9	0.4	5e-3	0.6	1e-2	214	380	5.8	18621
incre mentr	408	1554	472	2705	3237	8777	630	595	5.6	0.1	8e-4	0.5	2e-3	129	380	6.5	25466
c880	481	1582	541	3127	3213	5914	1977	1430	27.6	0.3	2e-3	1.0	3e-3	166	380	5.7	31710
c1355	665	2180	706	4155	4425	9130	1502	1193	20.5	0.4	2e-2	1.0	9e-3	248	280	9.1	67929
c499	667	2216	708	4167	4497	9240	1525	1201	21.3	0.4	8e-3	0.9	4e-3	283	370	9.4	78889
c1908	753	2416	786	4651	4883	7441											
c2670	837	2796	992	5643	5665	9358	1342	1066	20.5	0.3	1e-3	0.8	3e-3	283	380	7.0	104521

Noticeable trends

- Number of iterations relatively flat
- Are we really solving the problem?
  - Infeasibility is very small
  - Final projected gradient small (could be better)
  - Limited experiments with multiple start points
- Overnight runs for 1,000+ gate macros

Future work

- JiffyTune
  - post-optimality analysis
- EinsTuner
  - generalization of circuitry (dynamic logic, custom transistor-level designs, latches)
  - noise considerations
  - constraint pruning
- Design for manufacturability

Conclusions

- We have developed a pair of practical circuit optimization tools
- They are both applicable to custom circuitry
- They both use simulation and time-domain gradient computation in conjunction with a general-purpose nonlinear optimizer
- Several IBM designs have benefited extensively from the circuit optimizers
- Dynamic and static tuning are complementary weapons in a circuit designer's arsenal