High-Speed Clock Routing

Performance-Driven Clock Routing

- Given:
  - Locations of sinks {s_1, s_2, … ,s_n} and clock source s_0
  - Skew Bound B >= 0
    - If B = 0, zero-skew routing
  - Possibly other constraints:
    - Rise/fall time at sink
    - Clock phase delay
- Construct:
  - Clock routing tree T with skew
    Max-Delay(T) - Min-Delay(T) $\leq$ B
    + meeting other specified constraints
  - Minimize cost (e.g. total wirelength, power dissipation)

High-Speed Clock Routing

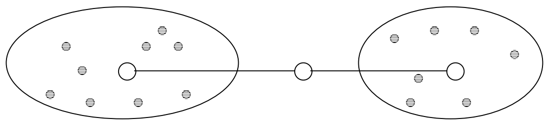
- Given:
  - Locations of sinks $\{S_1, S_2, \dots, S_n\}$ and clock source S_0
  - Skew Bound B
- Construct:
  - Clock routing tree with skew <= B
  - Minimize cost (e.g. total wirelengthe, power dissipation)

Minimum-Skew Clock Routing Techniques

- Top-down tree generation
  [Jackson-Srinivasan-Kuh, DAC'90]
- Bottom-up clock tree synthesis
  [Kahng-Cong-Robins,1991][Tsay,1991]
- Deferred merging embedding for a given topology
  [Edahino, DAC'93&ICCAD'93][Chao-Hsu-Ho, DAC'92][Boese-Kahng, ASIC'92]
- Planar clock routing
  [Zhu-Dai, ICCAD'92][Kahng-Tsao, ICCAD'94]
- Wiresizing and buffer insertion
  [Pullela et al, DAC'93][Chung-Cheng, ICCAD'94][Lin-Wong, ICCAD'94]
- Bounded-skew clock routing tree
  [Cong-Koh, ISCAS'95][Huang-Kahng-Tsao,DAC'95][Cong et al, ICCAD'95]

Top-Down Clock Tree Generation

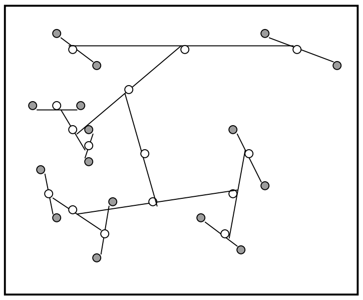
- Method of Means and Medians [Jackson-Srinivasan-Kuh, DAC'90]
  - Partition the sinks S into two sets S_L and S_R of equal size
  - Connect the center of mass of S to those of S_L and S_R
  - Recursively partition S_L and S_R in the orthogonal direction



- Given n sinks in a unit square
  - Total wirelength grows as $\frac{3}{2}\sqrt{n}$
  - Maximum path length skew grows proportional to $\frac{1}{\sqrt{n}}$

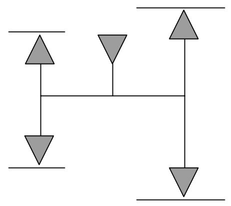
Bottom-Up Clock Tree Synthesis

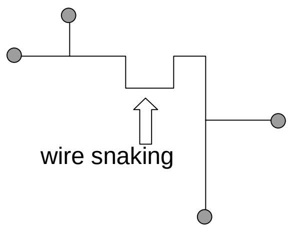
- Matching-based clock routing[Kahng-Cong-Robins, DAC'91]
  - Recursively match subtrees at each level
  - Minimize skew in each subtree
- Achieve zero-skew routing under pathlength delay model



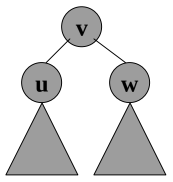
Enhancement of Bottom-Up Clock Tree Construction [Tsay, ICCAD'91]

- Use Elmore delay model instead of linear delay model
  Linear-time hierarchical computation of Elmore delay
- Consider multi-staged clock tree(buffered clock tree)
  Capacitance of subtree at buffer output will not be "carried over"
- Allow "wire snaking" for a achieving exact zero-skew


multi-staged clock tree



Deferred-Merge Embedding(DME) Algorithm [Chao-Hsu-Ho, DAC'92][Boese-Kahng, ASIC'92][Edahiro, NEC'92]



Given a Topology G
Bottom-Up:
Given sinks are all zero-skew trees
For each internal node v in topology in bottom up fashion
  let u and w be children of v
  Merging Segment ms(v)=zero-skew-merge(ms(u), ms(w))
  - Maintain zero-skew at each sub-tree
  - Find good placements of internal nodes

Deferred-Merge Embedding(DME) Algorithm

[Chao-Hsu-Ho, DAC'92][Boese-Kahng, ASIC'92][Edahiro, NEC'92]

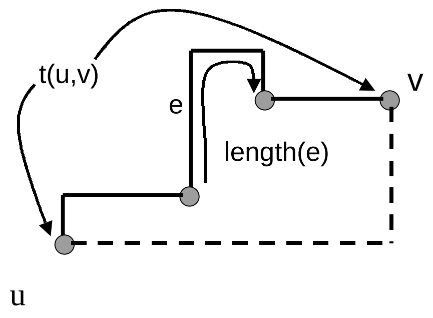
Top-Down:

S0 = any point in ms(root)

Recursively: Given location(v)

- lovation(u) = q in ms (u) s.t. d(q, location(v)) is minimized
- location(w) = q in ms (w) s.t. d(q, location(v)) is minimized

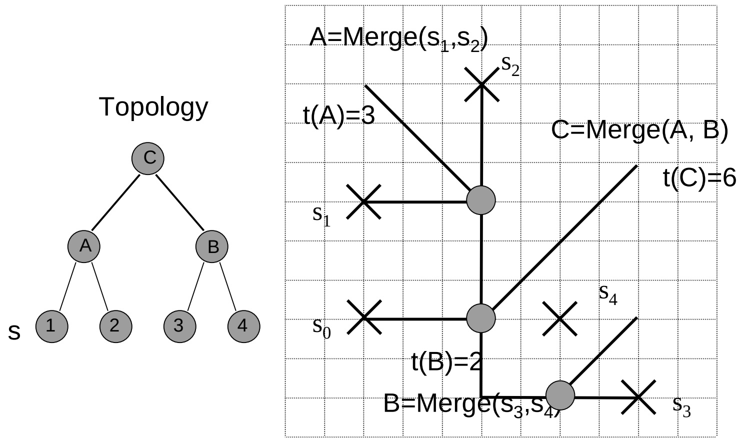
Pathlength Delay Model



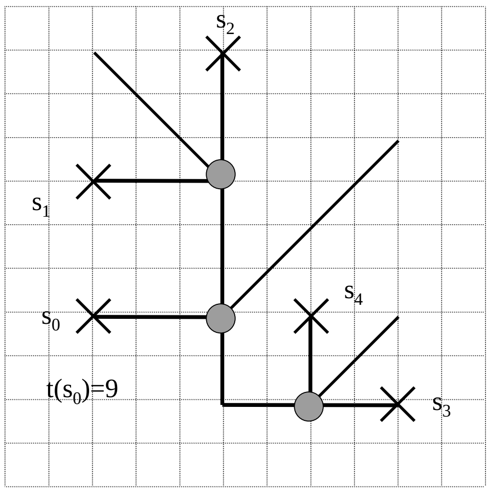
$$t(u,v) = \sum_{\text{edge } e \text{ in } u\text{-}v \text{ path}} length(e)$$

D(u,v): Manhattan Distance

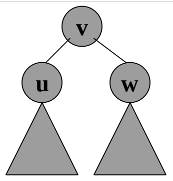
An Example of DME Algorithm: Bottom Up



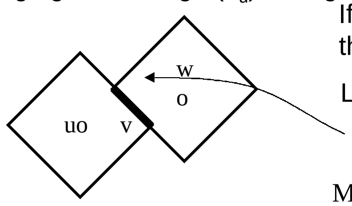
An Example of DME Algorithm: Top Down



Merging Two Zero Skew Trees



- Given two zero-skew trees T(v), T(v)
  - t(u)=delay from u to its sinks
  - t(w)=delay from w to its sinks
- Given location(u) & location(w), Find possible locations of v
  - t(v)=lengths(e_u) + length(e_w)
- Merging Cost=length(e_u) + length(e_w)

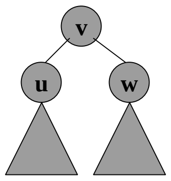


If u, w are sinks, t(u) = t(w) = 0, then for minimum merging cost

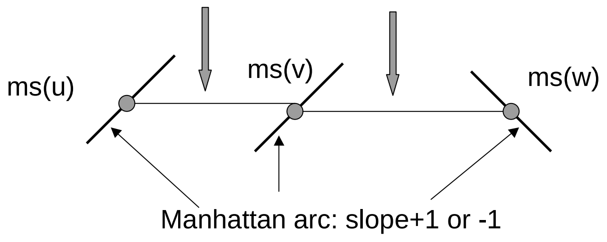
Length(e_u) = length(e_w) = $\frac{d(u, w)}{2}$

Merging Segment, ms(v)

Minimum Cost Merging Segment



- Sink s_i, ms(s_i) = location(s_i)
- Internal node v, children u and w:
  - set of possible locations of v s.t.
  - skew(v)=0
  - merging cost lengths(e_u) + length(e_w) minimized



Construction of Merging Segment

- Merging Cost is at least d(u,w)
- If |t(u) - t(w)| <= d(u,w) merging cost = d(u,v)

length(e_u) = $\frac{d(u,w)-t(u)+t(w)}{2}$

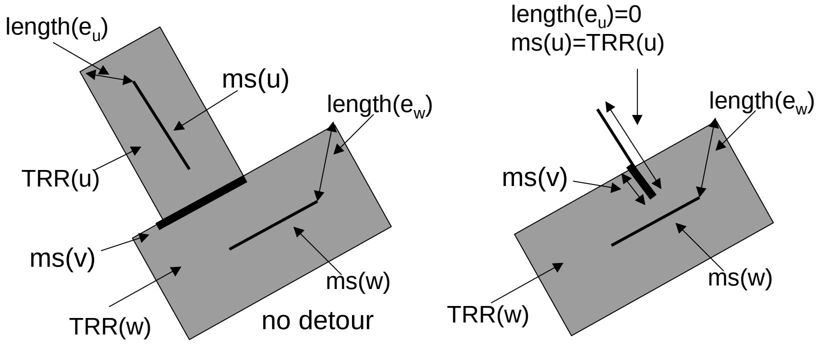
length(e_w) = $\frac{d(u,w)-t(w)+t(u)}{2}$

- Otherwise, merging cost=|t(u)-t(w)|, detour occurs
  - if t(u) <= t(w) length (e_u) = |t(u)-t(w)|
    length (e_w) = 0
  - if t(w) <= t(u) length (e_w) = |t(u)-t(w)|
    length (e_u) = 0

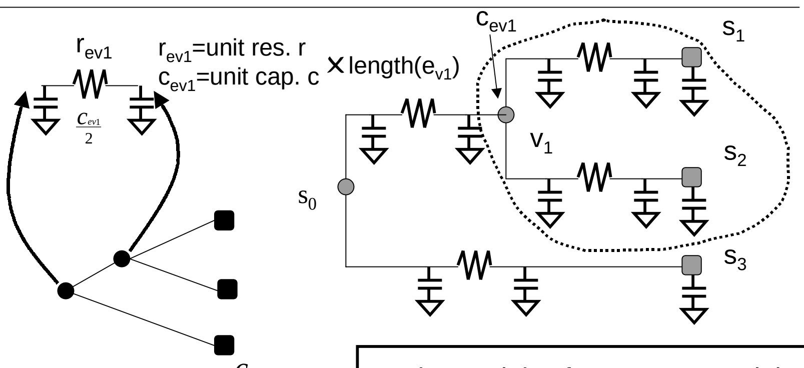
Construction of Merging Segment

(1)
TRR(u)=Expand ms(u) by length(e_u)
TRR(w)=Expand ms(u) by length(e_w)
TRR: Tilted Rectangle Region

(2)
ms(v)=TRR(u)∩TRR(w)



Circuit Model for Elmore Delay



Delay of Edge E = $r_E(\frac{c_E}{2}+C_E)$

Where r_E: wire resistance of E
c_E: wire cap. of E
C_E: total cap. rooted at E

Elmore delay from source to sink s_i

$t(N_i)=\sum$ Delay of Edge E
E in Path
to sink s_i

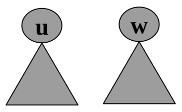
Construction of Merging Segment under Elmore Delay

- t(v) = delay of edge e_u+t(u) = delay of edge e_w+t(w)

$$r_{e_u}(\frac{c_{e_u}}{2}+c_{e_u})+t(u)=r_{e_w}(\frac{c_{e_w}}{2}+c_{e_w})+t(w)$$

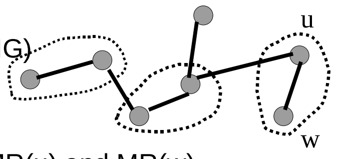
- Solve $x=\frac{t(w)-t(u)+r\times d(u,w)(c_{e_w}+\frac{1}{2}c\times d(u,w))}{r(c_{e_u}+c_{e_w}+c\times d(u,w)}$
- if $0 \leq x \leq d(u,w)$ length(eu) = x
  length(ew) = d(u,w) - w
- Otherwise detour, if $t(u) \leq t(w)$ length(ew)=0
  – solve $r_{e_u}(\frac{c_{e_u}}{2}+c_{e_u})+t(u)=t(w)$ for length(e_u)

Bounded-Skew Tree Construction with DME (BST/DME) [Cong-Koh, ISCAS'95][Huang-Kahng-Tsao, DAC'95]

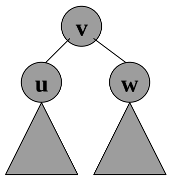


……..

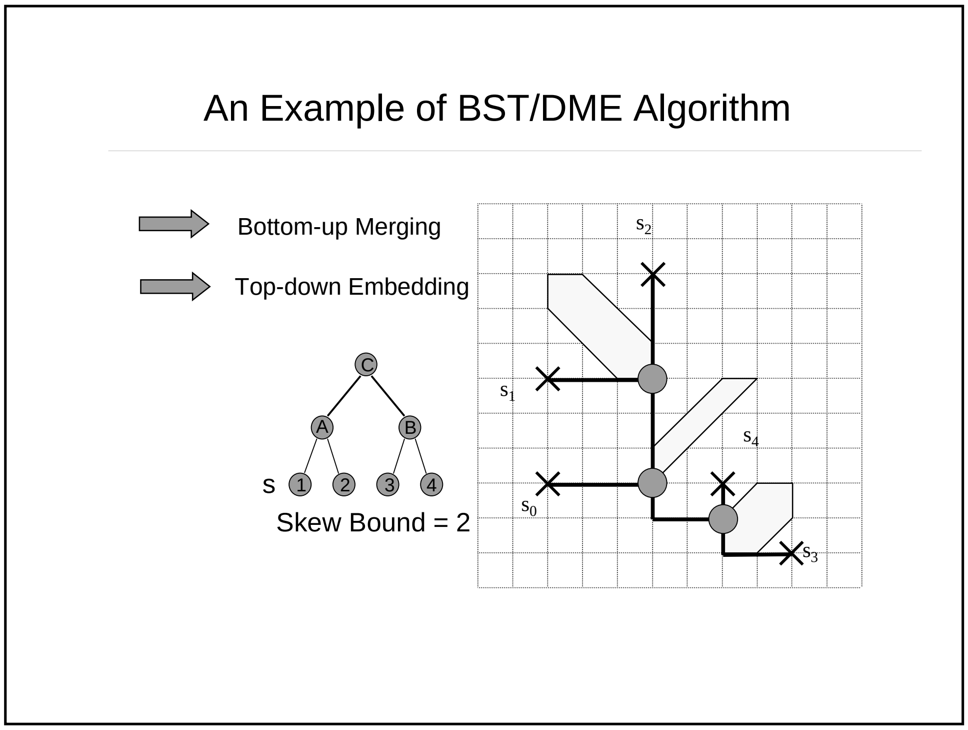
```
Bottom-Up:
while more than 1 tree do
    Compute nearest neighbor graph (NNG)
    Select a matching M in NNG
    For each edge e = uw in M do
        Merging Region MR(v)=merge MR(u) and MR(w)
```



Bounded-Skew Tree Construction with DME (BST/DME) [Cong-Koh, ISCAS'95][Huang-Kahng-Tsao, DAC'95]

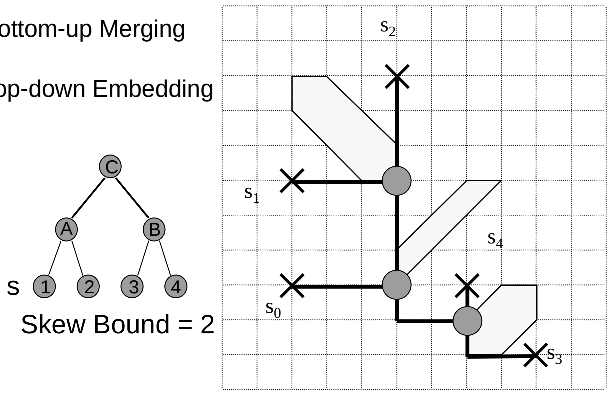


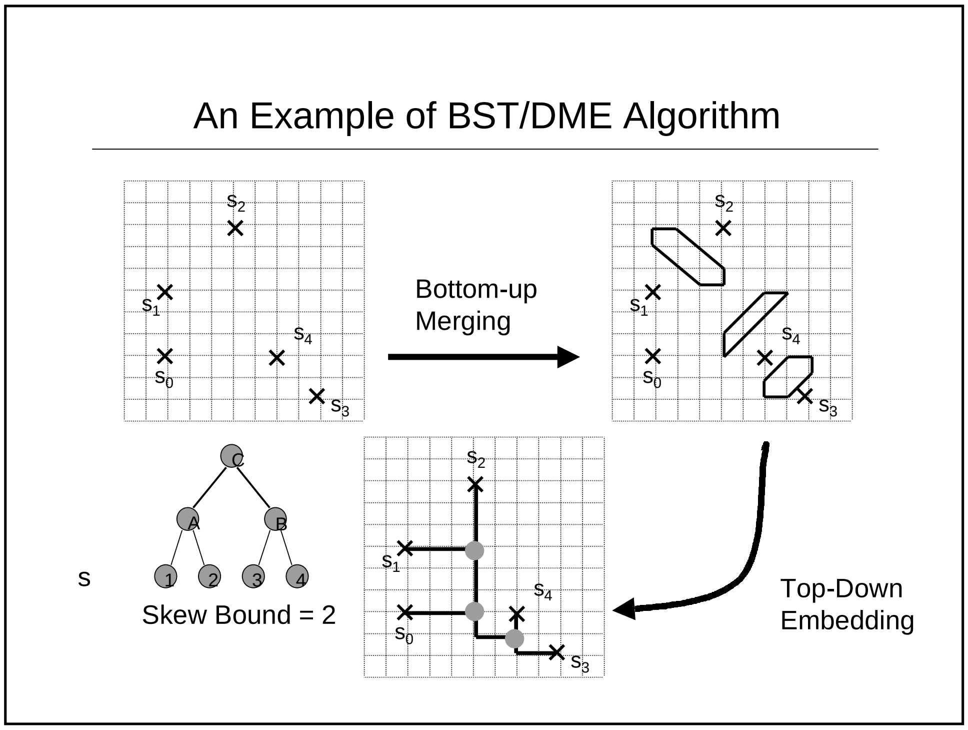
```
Top-Down:
$N_0$=any point in MR(root)
Recursively: Give location(v)
    location(u) = q in MR(u) s.t. d(q, location(v)) is minimized
    location(w) = q in MR(w) s.t. d(q, location(v)) is minimized
```



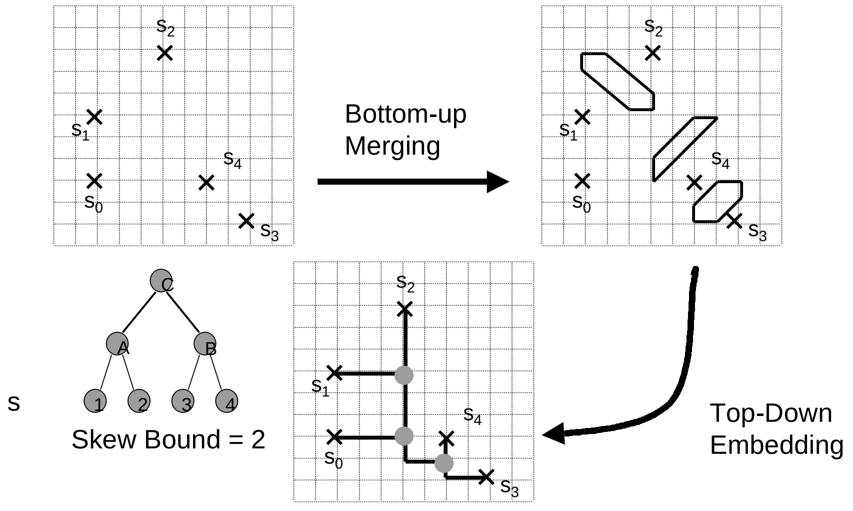
An Example of BST/DME Algorithm

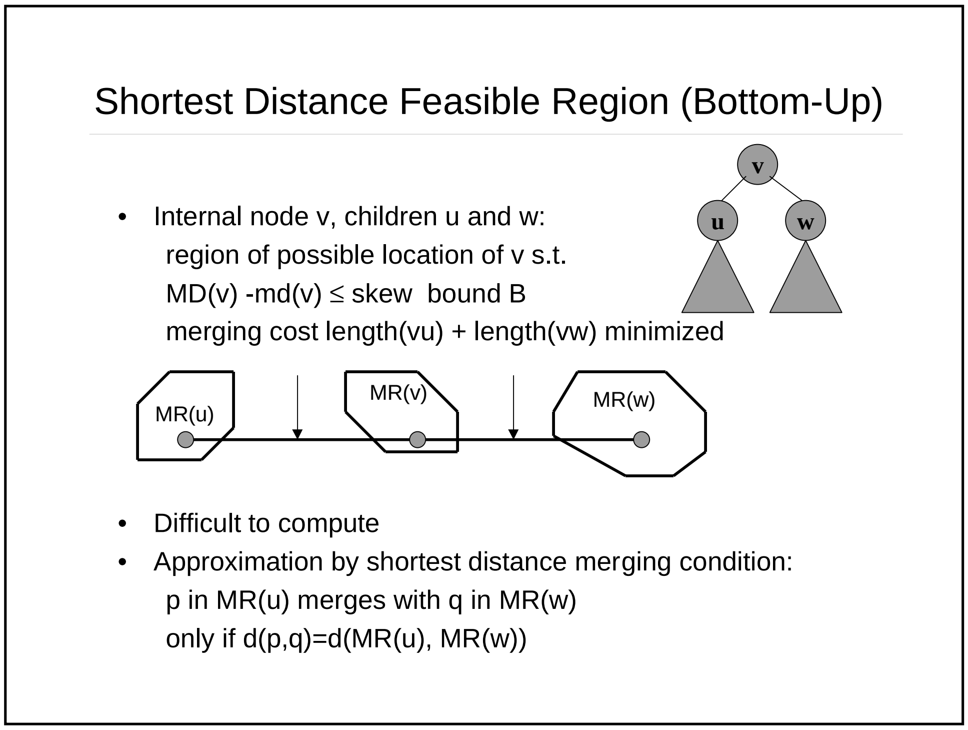
- Bottom-up Merging
- Top-down Embedding





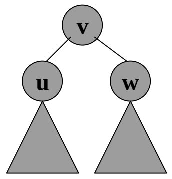
An Example of BST/DME Algorithm

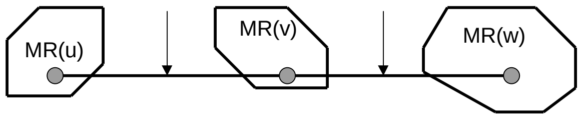




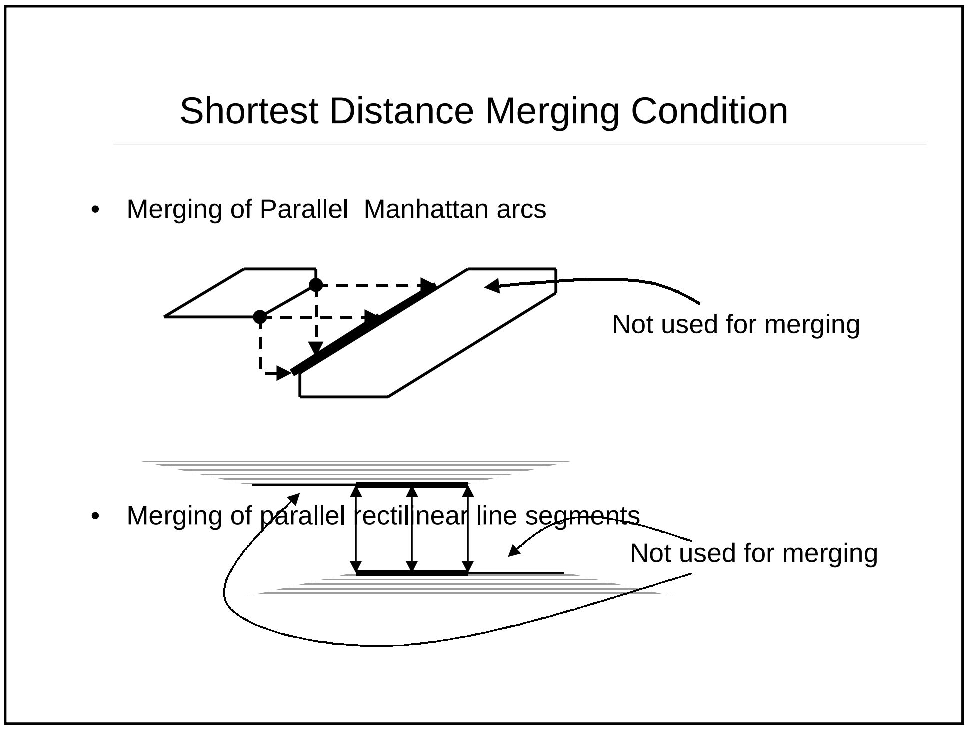
Shortest Distance Feasible Region (Bottom-Up)

- Internal node v, children u and w:
  region of possible location of v s.t.
  MD(v) -md(v) ≤ skew bound B
  merging cost length(vu) + length(vw) minimized



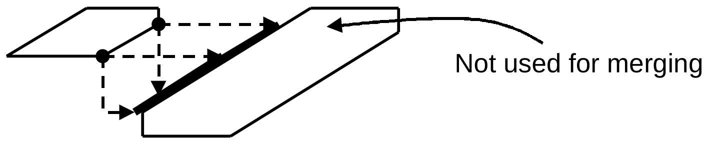


- Difficult to compute
- Approximation by shortest distance merging condition:
  p in MR(u) merges with q in MR(w)
  only if d(p,q)=d(MR(u), MR(w))

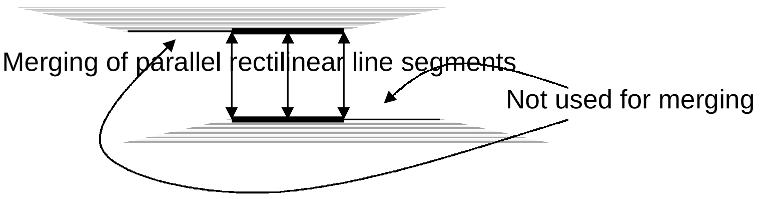


Shortest Distance Merging Condition

- Merging of Parallel Manhattan arcs

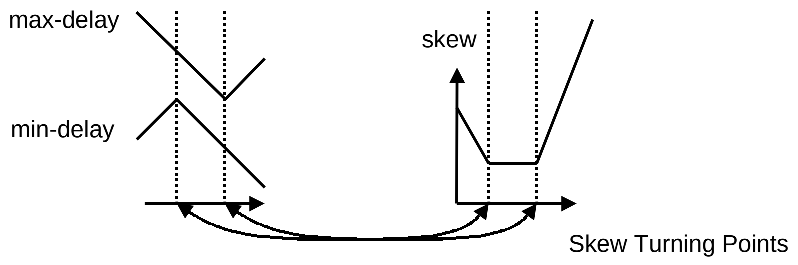


- Merging of parallel rectilinear line segments



Properties of SDFR

- Octilinear property: SDFRs are convex octilinear polygons
- Boundary property: Boundary segments of SDFR
  - Manhattan arc: constant max-delay & constant min-delay
  - Well-Behaved Rectilinear line segment



Merging of Parallel Manhattan Arcs

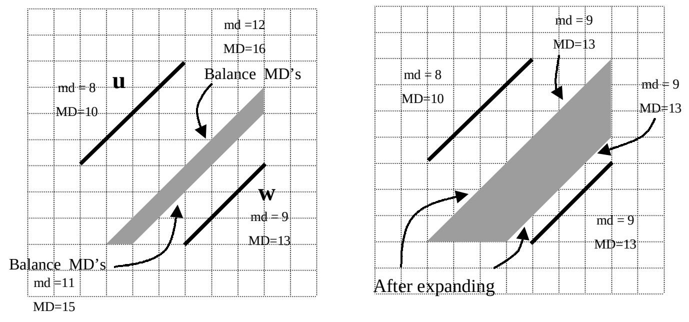
(1) Compute Core

skew(core)=max(skew(u),skew(w))

balance md, MD(as in zero-skew)

(2) slack(r) = B-skew(r)

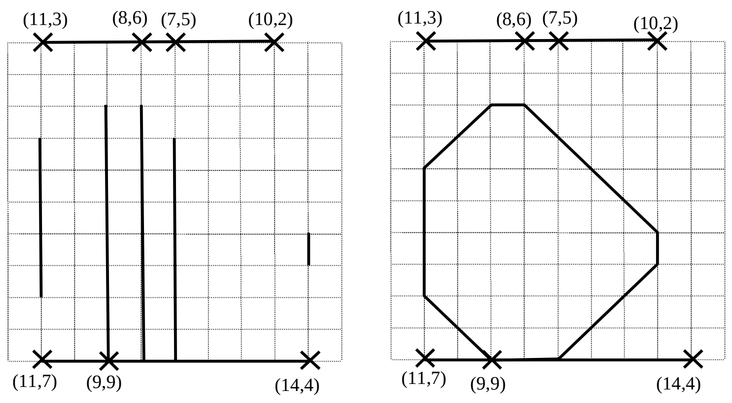
Expand by slack(r)/2



Merging of Parallel Rectilinear Segments

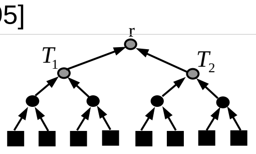
(1) Compute SDFSs at all skew turning points

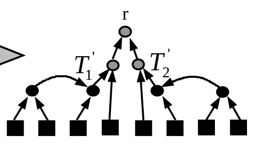
(2) Perform a walk to join all vertices



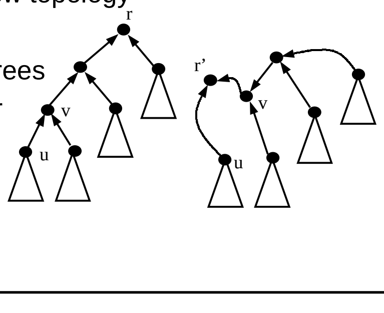
Dynamic Topology Change to Reduce Merging Cost

[Huang-Kahng-Tsao, DAC'95]

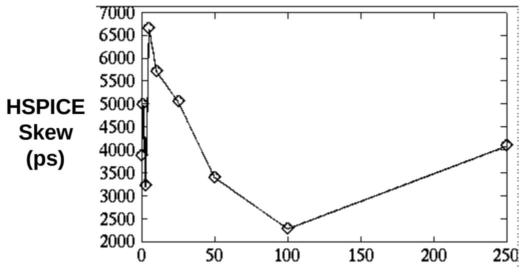




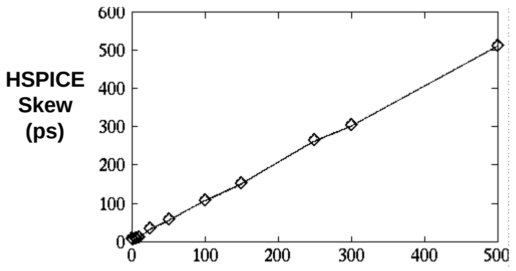
- DME always merge at root
- Lower merging cost by changing topology
  - Effective for large skew bound
- Re-rooting operation
  - Consider all edges uv in T
  - Make u and v the children of new topology
  - O(n) topologies
  - O( ) ^{2}connections between 2 trees
  - Least cost for Nearest neighbor



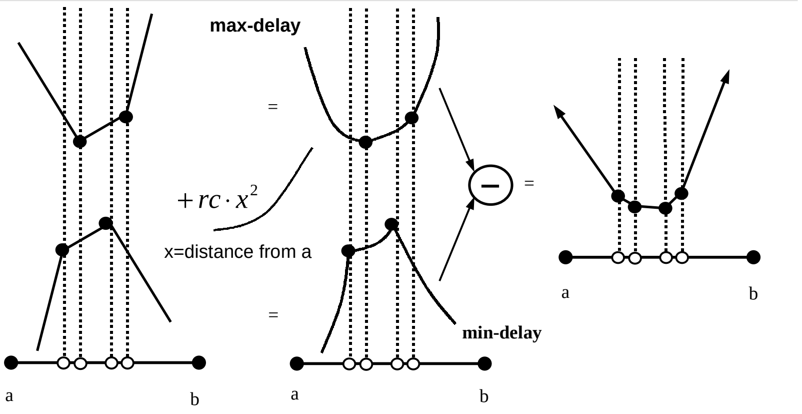
Pathlength Skew
vs.
HSPICE Skew



Elmore Skew
vs.
HSPICE Skew



Elmore Delays and Skew along Well-Bahaved Line Segment

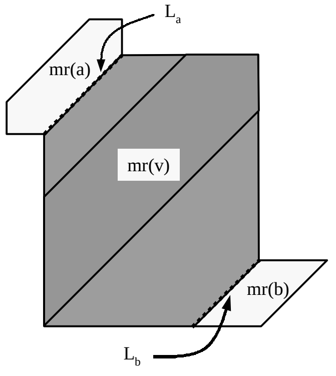


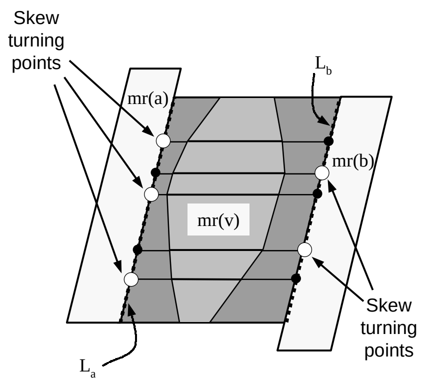
- Max-delay n_1 piecewise-linear concave + min-delay n_2 piecewise-linear convex + Quadratic Term
- skew ≤ n_1+ n_2 - 1 piecewise-linear concave

Construction of Merging Region for Boundary Merging and Embedding(BME)

Manhattan Arcs with constant max-delay and min delay

Line segments with well-behaved max-delay and min-delay

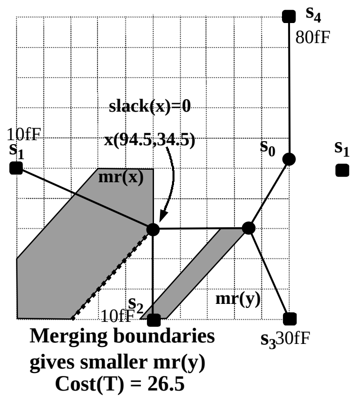


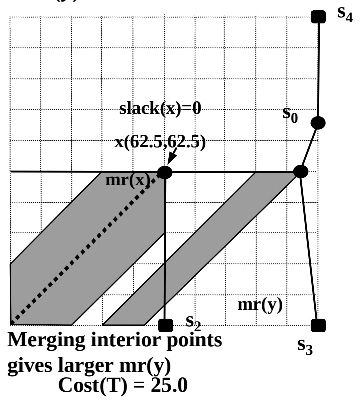


Each point in the region has the same total capacitance

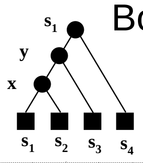
Boundary Segments vs. Interior points Merging

- Interior merging uses less skew resources
- Preserve skew resource for merging cost reduction at an upper level
  - e.g., merge mr(s) with s_3 followed by merging mr(y) with s_4
  - Merging boundaries gives smaller mr(y), final cost = 26.5
  - Merging interior points gives larger mr(y), final cost = 25.0

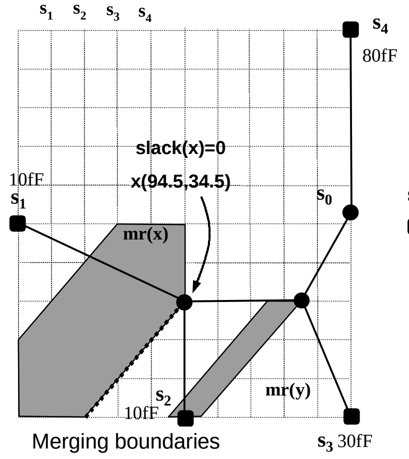




Boundary Segments vs. Interior points Merging

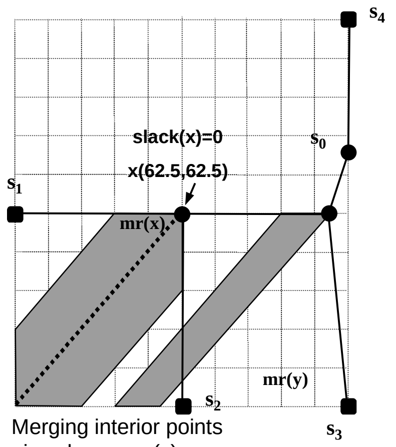


Slack(p) = Skew bound - skew(p)



Merging boundaries gives smaller mr(y)

Cost(T) = 26.5

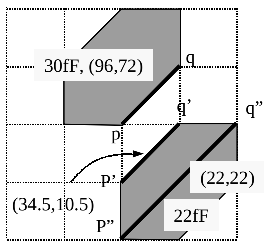


Merging interior points gives larger mr(y)

Cost(T) = 25.0

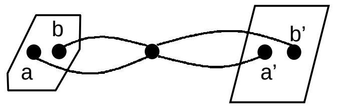
Advantages of Interior Merging and Embedding (IME)

- Conserve slack for upper level use
  - larger merging region and less merging cost
- Eliminate the needs of detour
  - Merge pq with p’q’, merging cost 2.5
  - Merge pq with p’’q’’, merging cost 2



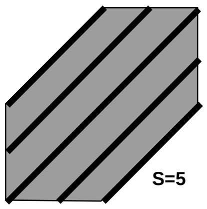
Difficulty of Merging Interior Points

- Ambiguity of max-delay and min-delay of a point



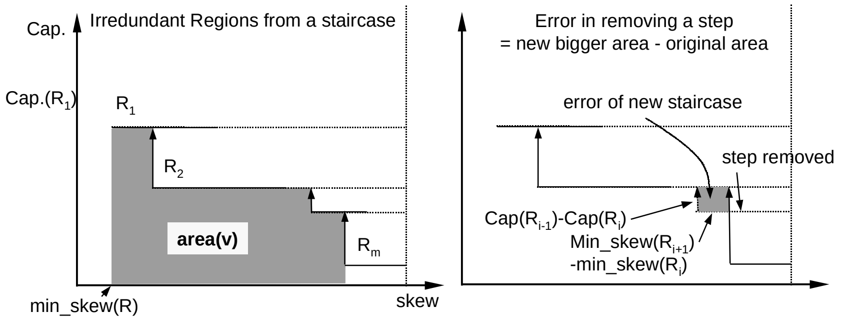
V has different delays due to merging of different interior point

IME using Sampling and Dynamic Programming



- An Internal node has a set of merging regions
- Each region sampled by s Manhattan arcs
- Merging two regions gives s^2 merging regions
- Problem: exponential growth
  O(s^n) merging regions at root of n merging regions
- Solution: keep at most k regions per node
  1. Merge children to get $(ks)^2$ merging regions
  2. Remove "redundant" regions
     R redundant if Cap(R) > Cap(R') &
     min_skew(R) > min_skew(R')
  3. Choose "best" k out of m irredundant regions
     Optimal (m,k)-Sampling by dynamic programming

Optimal (m.k)-Sampling Problem

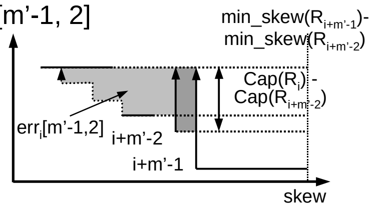


- Given m irredundant merging regions
  IMR = {R_1, R_2, …, R_m}
- Find $2 \leq k \leq m$ merging regions
  IMR' = {$R_1 = R_{\pi(1)}$, $R_{\pi(2)}$, …, $R_{\pi(k-1)}$, $R_m = R_{\pi(k)}$}
- Error is minimum
  error = area(IMR') - area(IME)

Optimal (m,k)-Sampling Algorithm

S_i[m’, k’] = Optimal Solution for {R_i, R_{i+1}, …, $R_{i+m'-1}$}
err_i[m’, k’] = Error for S_i[m’, k’]
$next_i$[m’, k’] = next region after R_i in S_i[m’, k’]

(1) If m’ = k’, select all, zero error
err_i[m’, m’] = 0 & $next_i$[m’, m’] = i+1
(2) If m’ > k’, k’ = 2, retain R_i & $R_{i+m'-1}$
erri[m’, m’] = (min_skew($R_{i+m'-1}$) - min_skew($R_{i+m'-2}$)) × (Cap(R_i) - Cap($R_{i+m'-2}$)) + err_i[m’-1, 2]
$next_i$[m’, 2] = i+m’-1



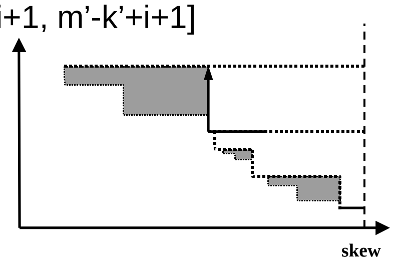
(3) if m’>k’, k’>2,

$$err_i[m',k'] = \min_{i' \in [i+1, m'-k'+i+1]} \begin{pmatrix} err_i[i'-i+1, 2] \\ + \\ err_{i'}[m'-i'+i, k'-1] \end{pmatrix}$$

$next_i$[m’,k’] = best i’ in [i+1, m’-k’+i+1]

Complexity

- $O(km^3)$ Algorithm
  For 2 ≤ m’ ≤ m
  2 ≤ k’ ≤ k ≤ m
  1 ≤ i ≤ m-m’+1
  Find err_i[m’, k’] & $next_i$[m’, k’]
- Restrict m’ = m-i+1 for case (3); $O(km^2)$

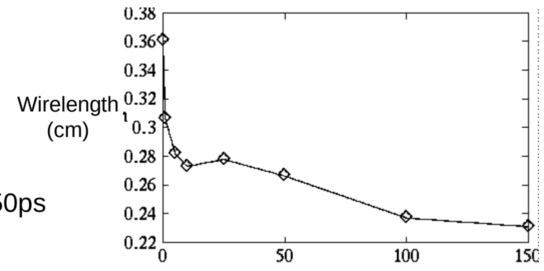


Performance of BME/IME

skew(ps)		*r1*	*r2*	*r3*	*r4*	*r5*
0	CL+6	0.1253347	0.2483754	0.3193801	0.6499660	0.9723726
0	BME	0.1307637	0.2647476	0.3344828	0.6934030	1.1066897
	IME	0.1445555	0.2907593	0.3605778	0.7255455	1.0706940
1	BME	0.1223125	0.2397494	0.3427426	0.6415233	0.9470000
	IME	0.1274819	0.2526961	0.3060284	0.6571435	0.9896655
10	BME	0.1087703	0.2155481	0.2789321	0.5411937	0.8140187
	IME	0.1112512	0.2353202	0.2727299	0.5350241	0.8065110
100	BME	0.0926205	0.2201023	0.2515178	0.4860958	0.7375204
	IME	0.0930426	0.1978378	0.2366874	0.4953715	0.7003996
1000	BME	0.0793498	0.1839666	0.2506399	0.4971769	0.7134697
	IME	0.0861561	0.1995665	0.2097784	0.4740962	0.6280007
10000	BME	0.0780100	0.1668872	0.2102182	0.4017261	0.6136574
	IME	0.0790285	0.1574153	0.1998007	0.4326234	0.5912472

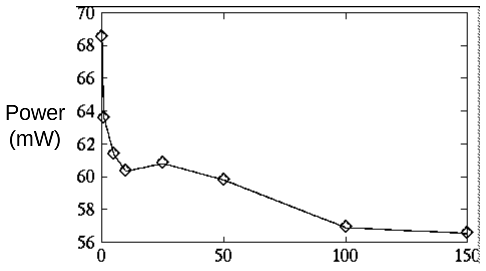
Wirelength and Skew Tradeoff

- r3 benchmark
- skew from 0 to 150ps



HSPICE Power and Skew Tradeoff

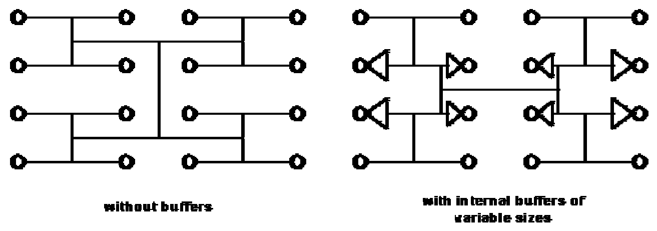
- Single buffer at root
- Clock rate 50Mhz



Topology and Buffer insertion

[Vittal-Marek-Sadowska, DAC'95]

- Buffer can be used to balance sink delay (Instead of using "detour" wirelength)



Insertion of buffer to improve rise/fall time

- However, may increase delay/skew sensitivity due to process variations

Greedy Internal Buffering Algorithm (Grin)

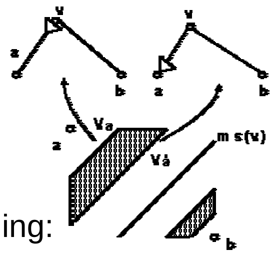
- DME-style bottom-up merging
- Min-cost merging"

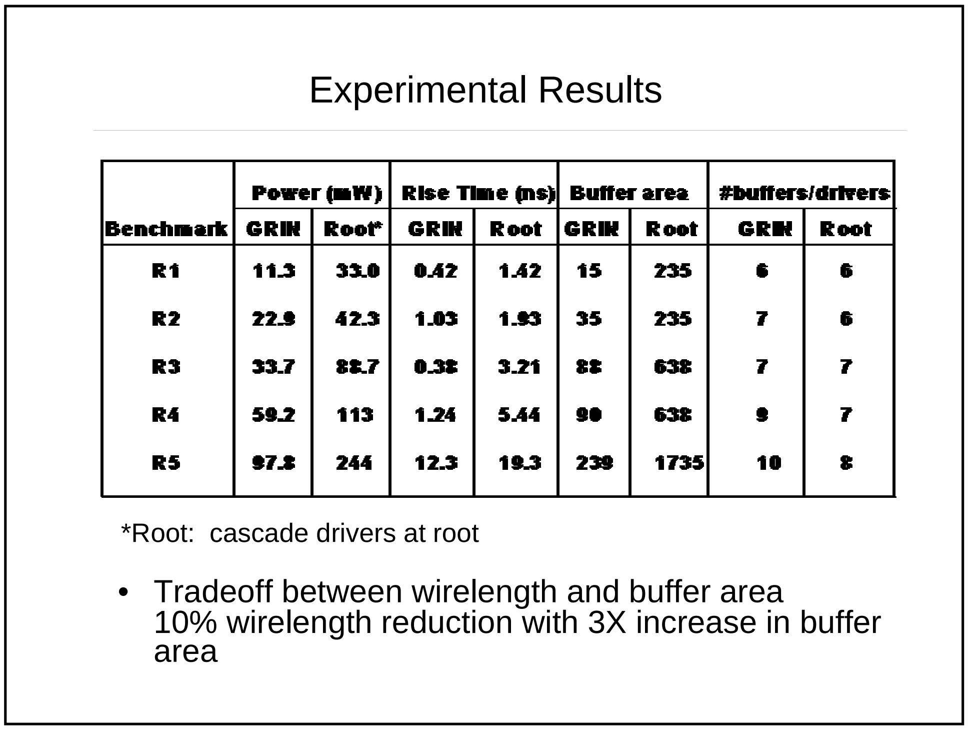
  cost = linear combination of
  tree length + buffer area
- Consider buffer insertion at each merging:

  ms(v): merging segment when no buffer is inserted

  V_a: buffer inserted right after v, the parent node

  V'_a: buffer inserted right before a
- Insert buffer to meet rise/fall time constraint after merging if necessary



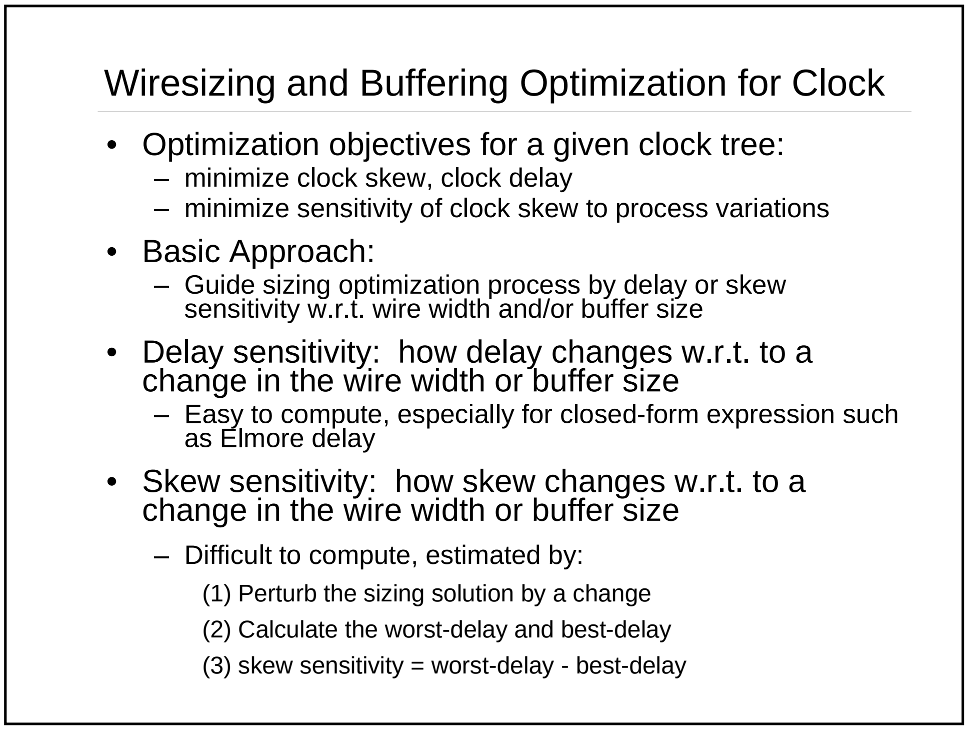


Experimental Results

Benchmark	Power (mW)		Rise Time (ns)		Buffer area		#buffers/drivers	
	GRIN	Root*	GRIN	Root	GRIN	Root	GRIN	Root
R1	11.3	33.0	0.42	1.42	15	235	6	6
R2	22.9	42.3	1.03	1.93	35	235	7	6
R3	33.7	88.7	0.38	3.21	88	638	7	7
R4	59.2	113	1.24	5.44	90	638	9	7
R5	97.8	244	12.3	19.3	239	1735	10	8

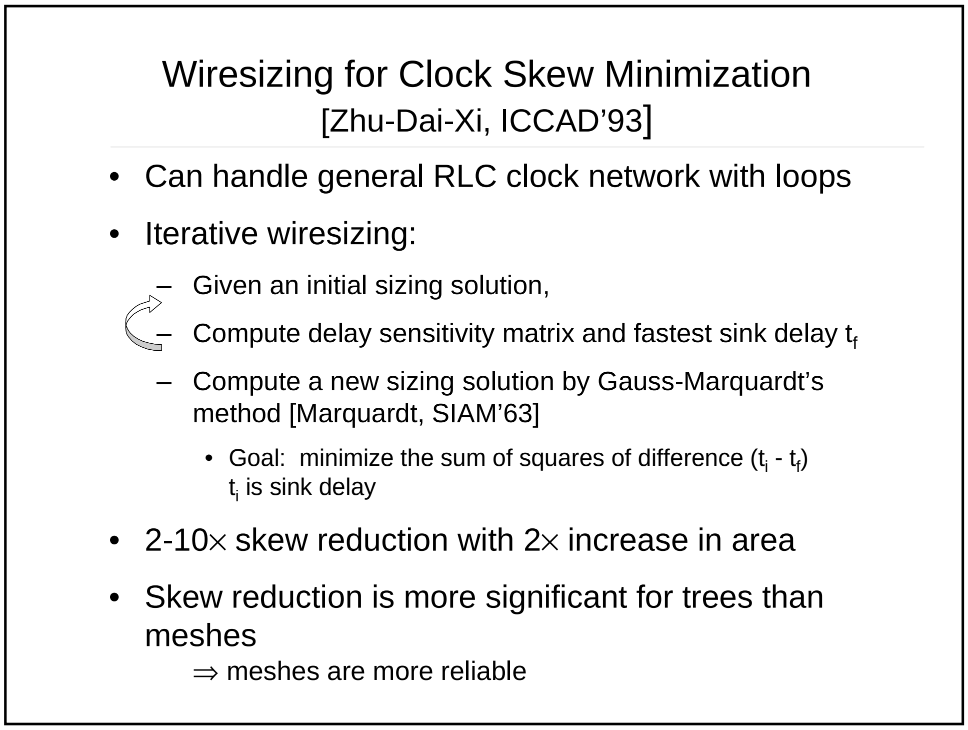
*Root: cascade drivers at root

- Tradeoff between wirelength and buffer area
  10% wirelength reduction with 3X increase in buffer area



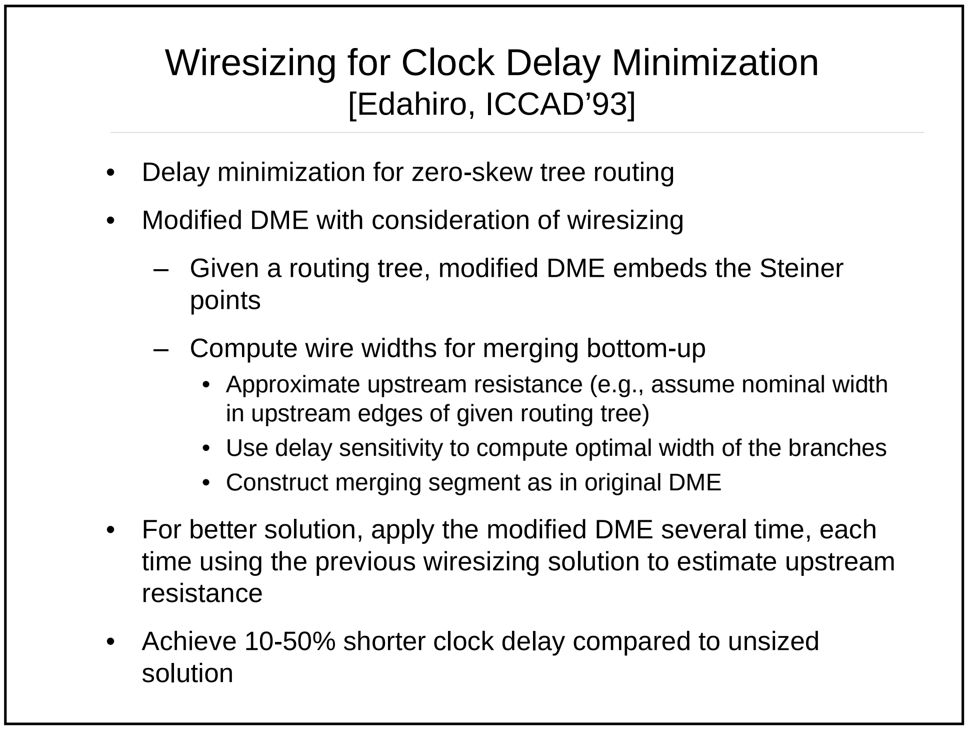
Wiresizing and Buffering Optimization for Clock

- Optimization objectives for a given clock tree:
  - minimize clock skew, clock delay
  - minimize sensitivity of clock skew to process variations
- Basic Approach:
  - Guide sizing optimization process by delay or skew sensitivity w.r.t. wire width and/or buffer size
- Delay sensitivity: how delay changes w.r.t. to a change in the wire width or buffer size
  - Easy to compute, especially for closed-form expression such as Elmore delay
- Skew sensitivity: how skew changes w.r.t. to a change in the wire width or buffer size
  - Difficult to compute, estimated by:
    - (1) Perturb the sizing solution by a change
    - (2) Calculate the worst-delay and best-delay
    - (3) skew sensitivity = worst-delay - best-delay



Wiresizing for Clock Skew Minimization
[Zhu-Dai-Xi, ICCAD'93]

- Can handle general RLC clock network with loops
- Iterative wiresizing:
  - Given an initial sizing solution,
  - Compute delay sensitivity matrix and fastest sink delay t_f
  - Compute a new sizing solution by Gauss-Marquardt's method [Marquardt, SIAM'63]
    - Goal: minimize the sum of squares of difference (t_i - t_f)
      t_i is sink delay
- 2-10× skew reduction with 2× increase in area
- Skew reduction is more significant for trees than meshes
  $\Rightarrow$ meshes are more reliable



Wiresizing for Clock Delay Minimization
[Edahiro, ICCAD'93]

- Delay minimization for zero-skew tree routing
- Modified DME with consideration of wiresizing
  - Given a routing tree, modified DME embeds the Steiner points
  - Compute wire widths for merging bottom-up
    - Approximate upstream resistance (e.g., assume nominal width in upstream edges of given routing tree)
    - Use delay sensitivity to compute optimal width of the branches
    - Construct merging segment as in original DME
- For better solution, apply the modified DME several time, each time using the previous wiresizing solution to estimate upstream resistance
- Achieve 10-50% shorter clock delay compared to unsized solution

Buffer Insertion and Wiresizing for Clock
[Chung-Cheng, ICCAD'94]

- Skew sensitivity and delay minimization using dynamic programming
- Construct a lookup table B[b,l,s] bottom-up
  - B[b, l, s]: min.skew sensitivity with b buffer levels,
    first level buffers at l, buffer size s
  - SS[l, s, l', s']: skew sensitivity for buffers at levels l and l'
    with sizes s and s', respectively
  - At level l, compute B[b, l, s] = min {SS[l, s, l's, s'] + B[b-1, l', s']}
  - At root, l = 0, choose the smallest B[b, 0, s], and trace back to get optimal buffering levels, and buffer types
- Buffers at same level have identical size
  $\Rightarrow$ Reduce impact of process variations in devices on skew

Buffer Insertion and Wiresizing for Clock (cont'd)

- Consider wiresizing in computation of SS[l, s, l', s']
  - Compute wire widths for branches from level l to level l' based on delay sensitivity
  - Perturb the wire widths according to possible process variations, and compute worst-case skew as skew sensitivity
- Post-processing relocates buffers to reduce total wirelength
- 87-144$\times$ reduction in worst-case skew
- 2-11$\times$ reduction in clock delay

Buffer Insertion/Sizing and Wiresizig for Clock [Pullela-Menezes-Pileggi, TCAD'97]

- Clock delay and skew sensitivity minimization while satisfying skew bound constraint B
- Assumed n levels of buffers to be placed in a l-level clock tree; determine the buffer levels in the tree exhaustively
- Divide skew resource B evenly s.t. each buffer level and each level of clock tree has same skew resource = B/(l+n)
- For each DC-connected subtree (defined by buffers/driver)
  – Compute the min. required width of a branch s.t. the maximum change in delay induced by a process variation <= B/2(l+n)

    $\Rightarrow$ the worst case skew under process variations <= B
  – Achieve zero-skew routing within subtree by wiresizing with possible detour wirelength (similar to [Edahiro, ICCAD'93])

Buffer Insertion/Sizing and Wiresizig for Clock, (cont'd)

- Buffer sizing optimization for DC-connected subtrees at the same level:
  – To minimize impact of process variation on devices on skew, buffers at the same level are of identical size
  – Problem: Loads do not match

    $\Rightarrow$ difficult to use buffers of identical size
  - Solution: Add a properly sized stub between a buffer and its subtree to achieve

    (i) Identical loading (under effective cap. model) for all buffers

    (ii) Identical buffer-to-sink delays, i.e., zero-skew
  - Use the smallest buffer size such that the maximum change in skew under device process variations <= B/(l+n)
- 25$\times$ delay reduction for large circuits compared to wiresizing only
- Buffer insertion reduces max. wire width used

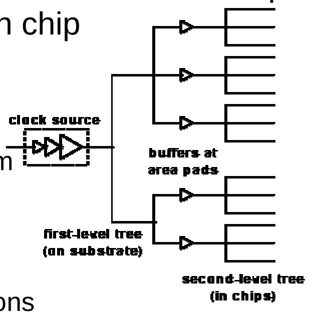
Other Studies on Clock Routing for Power Minimization

- Hierarchical Routing for Low Power
  [Zhu, et al., IWLPD'94]
- Gated Clock Tree for Low Power
  [Tellez-Farrahi-Sarrafzadeh, ICCAD'95]
- Device or Interconnect Sizing for Low Power
  [Xi-Dai, DAC'95]
  [Desai-Cvijetic-Jensen, DAC'96]
- Clock Scheduling with Gate Sizing for Low Power
  [Xi-Dai, DAC'96]

⋮

Hierarchical Routing
[Zhu et al., IWLPD'94]

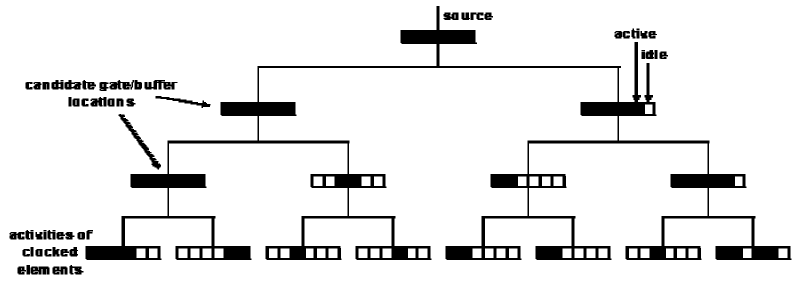
- Applicable to multi-chip module (MCM) technology
- Flip chip technology: area pads distributed over chip
  Several clock area pads for each chip
- Two-level clock routing
  – Routing on MCM substrate:
    Planar clock routing to connect from clock source to clock area pads
    [Zhu-Dai, ICCAD'92]
  – Routing in chip:
    Partition a chip into small size regions
    Clock pins in each region connected to a clock area pad



- Up to 76% clock power reduction

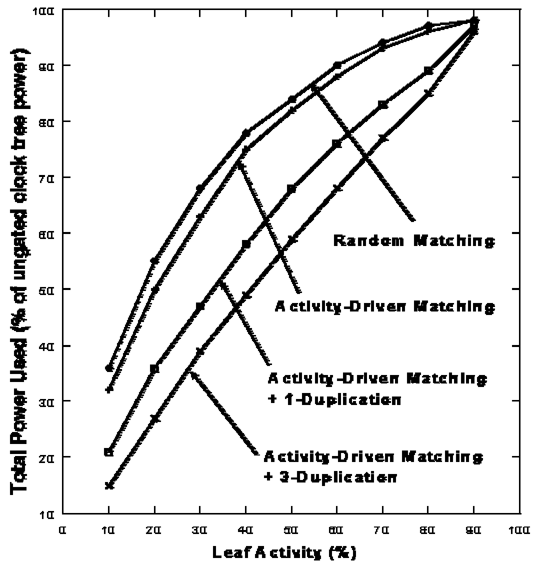
Activity-Driven Gated Clock Tree
[Tellez-Farrahi-Sarrafzadeh, ICCAD'95]

- Applicable when module activity patterns are known in advance
  - DSP circuits -- data activity is known
  - Microprocessors -- module activity sampled from simulations
- Difficult for circuits where high level behavior is data dependent
- Recursive matching to merge subtrees with similar activities


source
active
idle
candidate gate/buffer locations
activities of clocked elements

- Bottom-up dynamic programming to insert gates
- Insert buffers/gates to balance skew

Power Reduction by Using Matching and Gating



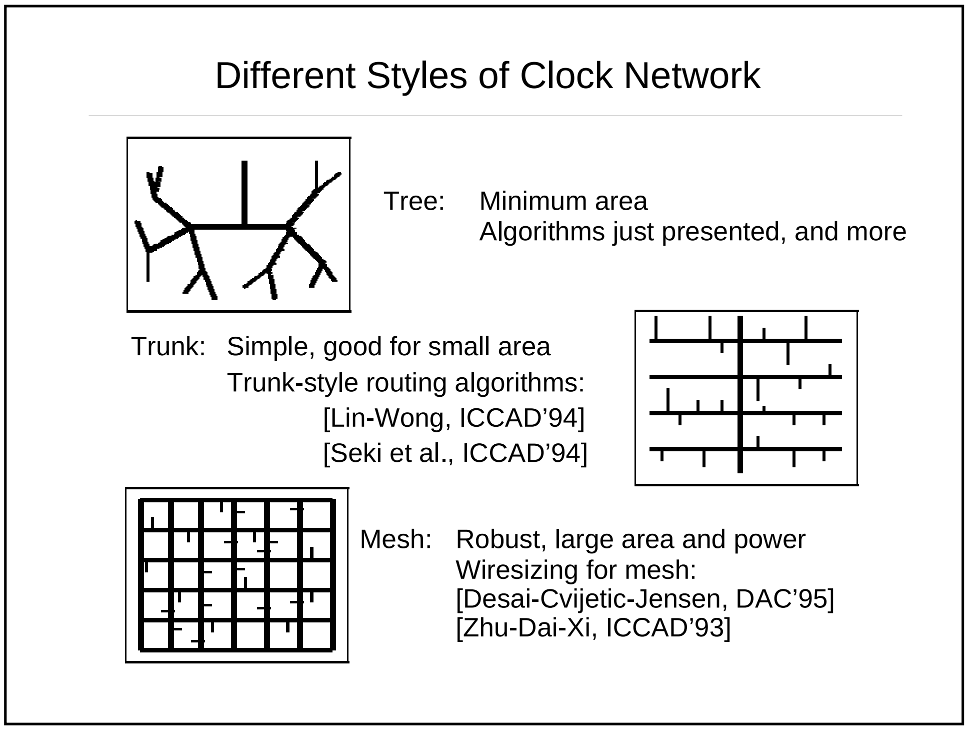
Activity pattern generated randomly:

(i) Randomly set k out of u time periods to be active

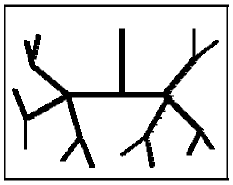
(ii) Apply (i), and duplicate the pattern d times

Gated clock can reduce power, even with random matching

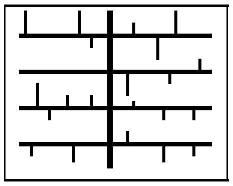
More power reduction when activity-driven matching is used



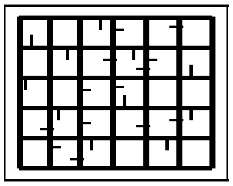
Different Styles of Clock Network



Tree: Minimum area
Algorithms just presented, and more



Trunk: Simple, good for small area
Trunk-style routing algorithms:
[Lin-Wong, ICCAD'94]
[Seki et al., ICCAD'94]



Mesh: Robust, large area and power
Wiresizing for mesh:
[Desai-Cvijetic-Jensen, DAC'95]
[Zhu-Dai-Xi, ICCAD'93]