An Efficient Incremental Algorithm for Min-Area Retiming

Jia Wang and Hai Zhou
Electrical Engineering and Computer Science
Northwestern University
Evanston, Illinois, United States

June, 2008

Outline

Retiming

- Relocate flip-flops (FFs) w/o changing circuit functionality. [Leiserson and Saxe 83]
- Powerful sequential transformation that re-schedules both computation and communication.
  - Wire pipelining via retiming for System-on-a-Chip. [Lin and Zhou 03]
- Become even more powerful when combined with combinational (logic) synthesis. [Malik et al. 91]
  - Essential for any sequential synthesis tool.

Sequential System Optimization by Retiming

- Min-period retiming:
  - Relocate FFs to minimize clock period.
  - Ignore cost – may significantly increase FF area.
- Min-area retiming:
  - Relocate FFs to minimize FF area under given clock period.
  - Can be used to minimize FF area under minimum clock period.
  - Of higher complexity!

Sequential System Optimization by Retiming

- Min-period retiming:
  - Relocate FFs to minimize clock period.
  - Ignore cost – may significantly increase FF area.
- Min-area retiming:
  - Relocate FFs to minimize FF area under given clock period.
  - Can be used to minimize FF area under minimum clock period.
  - Of higher complexity!

Problem Formulation

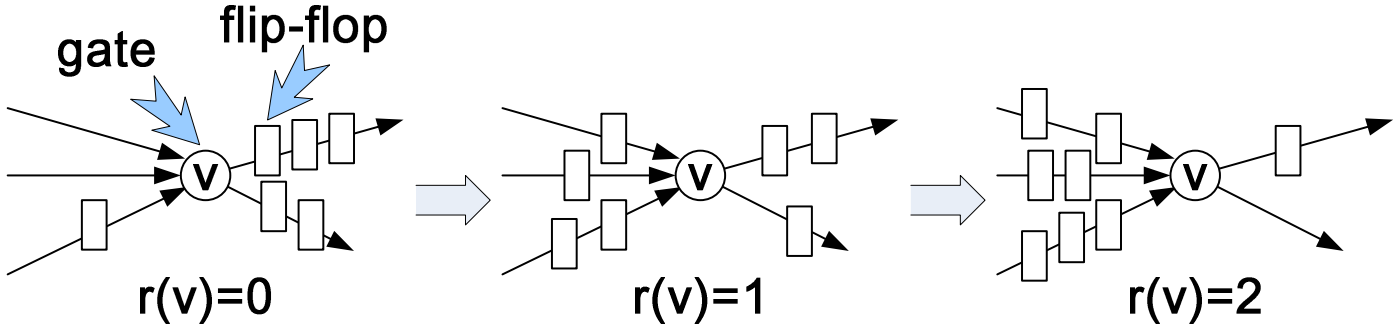
- Circuit graph $G = (V, E)$ of n vertices and m edges
  - G is usually sparse. $m << \Theta(n^2)$
  - Gate delay: $d(v)$, $v \in V$
  - # FFs on interconnects: $w(u, v)$, $(u, v) \in E$

Problem Formulation

- Circuit graph $G = (V, E)$ of n vertices and m edges
  - G is usually sparse. $m << \Theta(n^2)$
  - Gate delay: $d(v)$, $v \in V$
  - # FFs on interconnects: $w(u, v)$, $(u, v) \in E$
- Retiming is represented by an integer-valued vertex label.

$$r : V \rightarrow \mathbb{Z}$$

For gate v, $r(v)$ is the # FFs moved from all its fanouts to all its fanins.



Problem Formulation

- Validity constraints:

$$\forall (u, v) \in E : w(u, v) - r(u) + r(v) \geq 0$$

Problem Formulation

- Validity constraints:

$$\forall (u, v) \in E : w(u, v) - r(u) + r(v) \geq 0$$

- Timing constraints with clock period T:

$$\forall (u, v) \in E : w(u, v) = r(u) - r(v) \Rightarrow t(u) + d(v) \leq t(v)$$

$$\forall v \in V : d(v) \leq t(v) \leq T$$

Problem Formulation

- Validity constraints:

$$\forall (u, v) \in E : w(u, v) - r(u) + r(v) \geq 0$$

- Timing constraints with clock period T:

$$\forall (u, v) \in E : w(u, v) = r(u) - r(v) \Rightarrow t(u) + d(v) \leq t(v)$$

$$\forall v \in V : d(v) \leq t(v) \leq T$$

- Min-area objective:

$$\text{Minimize} \sum_{v \in V} r(v) * (\text{in}(v) - \text{out}(v))$$

More about Timing Constraints

- Timing constraints via arrival times t cannot be fitted into mathematical programming.
- Static longest path computation [Leiserson and Saxe 83]:
  - $W(u, v)$ is the min # of FFs on the paths from u to v.
  - $D(u, v)$ is the longest delay along the paths from u to v with $W(u, v)$ FFs.

$$\forall u, v \in V, D(u, v) > T : W(u, v) - r(u) + r(v) \geq 1$$

Mathematical Programming Formulation of Min-Area Retiming

[Leiserson and Saxe 83]

$$\begin{aligned} \text{Minimize} \quad & \sum_{v \in V} r(v) * (\text{in}(v) - \text{out}(v)) \\ \forall (u,v) \in E : \quad & w(u,v) - r(u) + r(v) \geq 0 \\ \forall u, v \in V, D(u,v) > T : \quad & W(u,v) - r(u) + r(v) \geq 1 \end{aligned}$$

Mathematical Programming Formulation of Min-Area Retiming

[Leiserson and Saxe 83]

$$\begin{aligned} \text{Minimize} \quad & \sum_{v \in V} r(v) * (\text{in}(v) - \text{out}(v)) \\ \forall (u,v) \in E: \quad & w(u,v) - r(u) + r(v) \geq 0 \\ \forall u, v \in V, D(u,v) > T: \quad & W(u,v) - r(u) + r(v) \geq 1 \end{aligned}$$

- It is a dual problem of the min-cost network flow.
- All previous optimal algorithms were based on this conventional formulation.

Previous Min-Area Retiming Algorithms

- The flow network is quite dense, especially when T is small.

Previous Min-Area Retiming Algorithms

- The flow network is quite dense, especially when T is small. No implementation was reported until [Shenoy and Rudell 94].
- Practical implementations focused on reducing # arcs in the flow network by heuristics. [Shenoy and Rudell 94], Minaret [Maheshwari and Sapatnekar 98]

Previous Min-Area Retiming Algorithms

- The flow network is quite dense, especially when T is small. No implementation was reported until [Shenoy and Rudell 94].
- Practical implementations focused on reducing # arcs in the flow network by heuristics. [Shenoy and Rudell 94], Minaret [Maheshwari and Sapatnekar 98]
- The flow network remains dense for large circuit.
  - Minaret generated more than 122,000,000 arcs for a circuit with more than 180,000 gates.
  - Require $\Theta(n^2)$ storage. Not efficient and not scalable.

Previous Min-Area Retiming Algorithms

- The flow network is quite dense, especially when T is small. No implementation was reported until [Shenoy and Rudell 94].
- Practical implementations focused on reducing # arcs in the flow network by heuristics. [Shenoy and Rudell 94], Minaret [Maheshwari and Sapatnekar 98]
- The flow network remains dense for large circuit.
  - Minaret generated more than 122,000,000 arcs for a circuit with more than 180,000 gates.
  - Require $\Theta(n^2)$ storage. Not efficient and not scalable.
- Incremental algorithm [Singh et al. 05] is heuristic and suboptimal.

Our Min-Area Retiming Formulation

Problem (Min-Area Retiming)

$$
\begin{aligned}
\textit{Maximize} \quad & \sum_{v \in V} b(v) r(v) \\
\forall (u,v) \in E: \quad & w(u,v) - r(u) + r(v) \geq 0 \\
\forall (u,v) \in E: \quad & w(u,v) = r(u) - r(v) \Rightarrow t(u) + d(v) \leq t(v) \\
\forall v \in V: \quad & d(v) \leq t(v) \leq T
\end{aligned}
$$

- Area improvement: $b(v) = out(v) - in(v)$
  - Reduction in FF area by moving 1 FF from its fanouts to fanins.
- Not even a mathematical programming formulation.

Outline

Incremental Min-Area Retiming

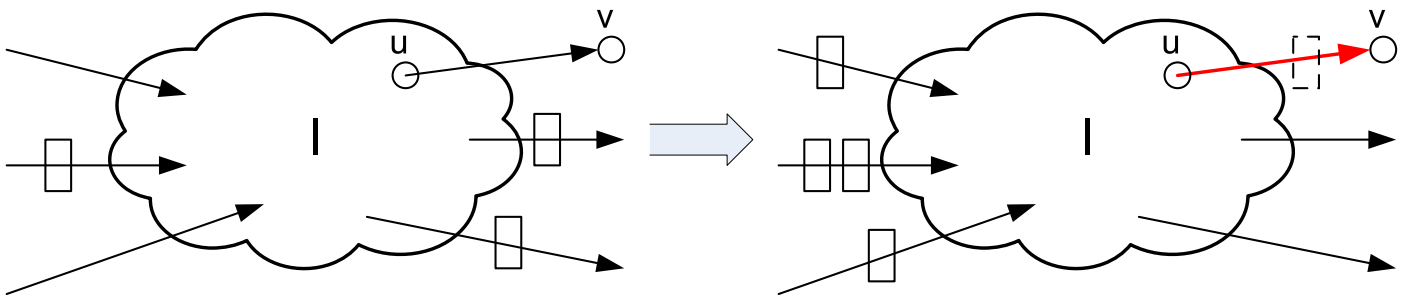
- Intuition: retime the circuit gradually.
- Start from an initial feasible retiming.
  - Feasible means validity and timing constraints are satisfied.
- Move 1 FF from the fanouts to the fanins of a cluster of gates in order to reduce FF area w/o violating validity and timing constraints.
  - Choose a cluster of gates that reduces FF area most.
  - Satisfy validity and timing constraints gradually.
- Stop when no such cluster exists.

Incremental Min-Area Retiming

- Intuition: retime the circuit gradually.
- Start from an initial feasible retiming.
  - Feasible means validity and timing constraints are satisfied.
- Move 1 FF from the fanouts to the fanins of a cluster of gates in order to reduce FF area w/o violating validity and timing constraints.
  - Choose a cluster of gates that reduces FF area most.
  - Satisfy validity and timing constraints gradually.
- Stop when no such cluster exists.

Incremental moves may violate validity and timing constraints

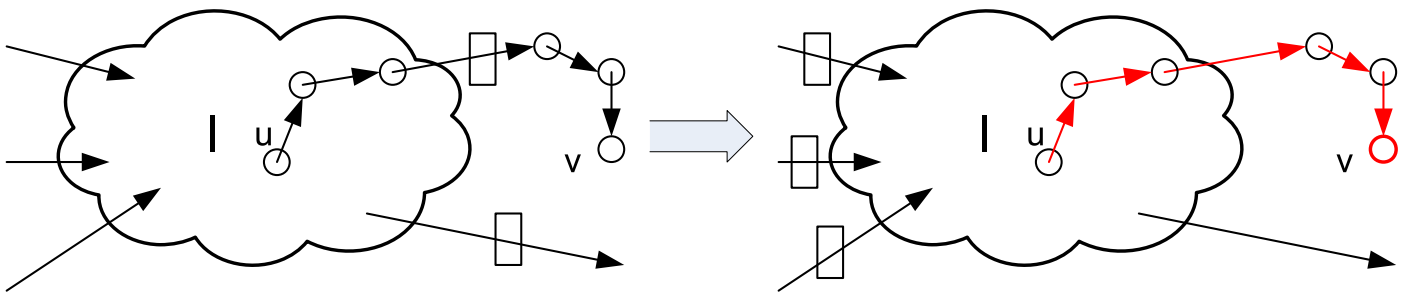
- Suppose retiming r_I is obtained by moving 1 FF from fanouts to fanins of a cluster I of gates in retiming r.
- r_I may violate the validity constraints.
  There is a fanout edge (u, v) of I with -1 FF.



- v should be included in I to satisfy the validity constraints if $u \in I$.

Incremental moves may violate validity and timing constraints

- Suppose retiming r_I is obtained by moving 1 FF from fanouts to fanins of a cluster I of gates in retiming r.
- r_I may violate the timing constraints.
  There is a combinational path from u to v longer than T.



- v should be included in I to satisfy the timing constraints if $u \in I$.

Active Constraints

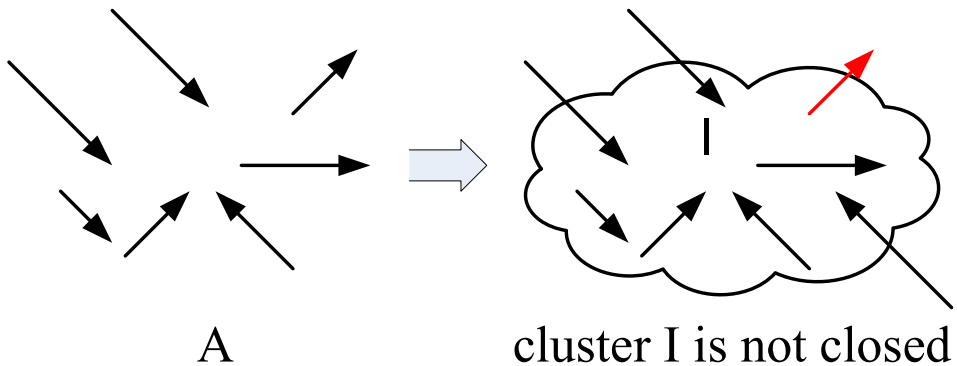
Define (u, v) to be an active constraint for the retiming r if including u but not v in the cluster would violate validity or timing constraints.

- Define a cluster I to be *closed* under active constraints A when no edge in A point to outside of the cluster I.

Active Constraints

Define (u, v) to be an active constraint for the retiming r if including u but not v in the cluster would violate validity or timing constraints.

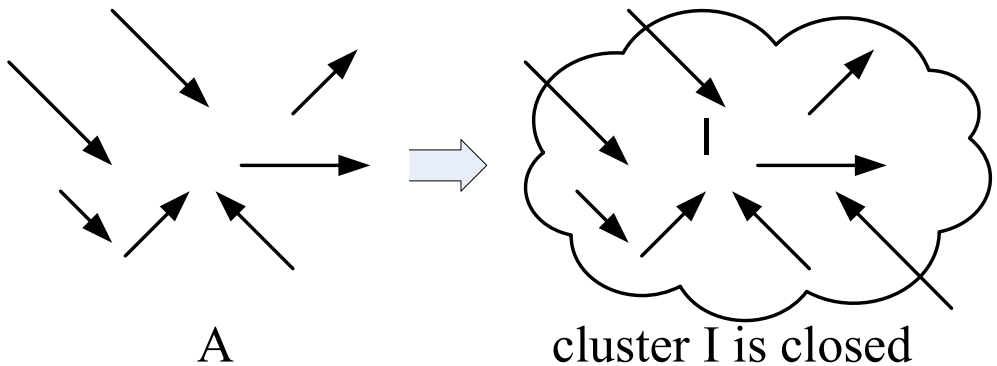
- Define a cluster I to be *closed* under active constraints A when no edge in A point to outside of the cluster I.



Active Constraints

Define (u, v) to be an active constraint for the retiming r if including u but not v in the cluster would violate validity or timing constraints.

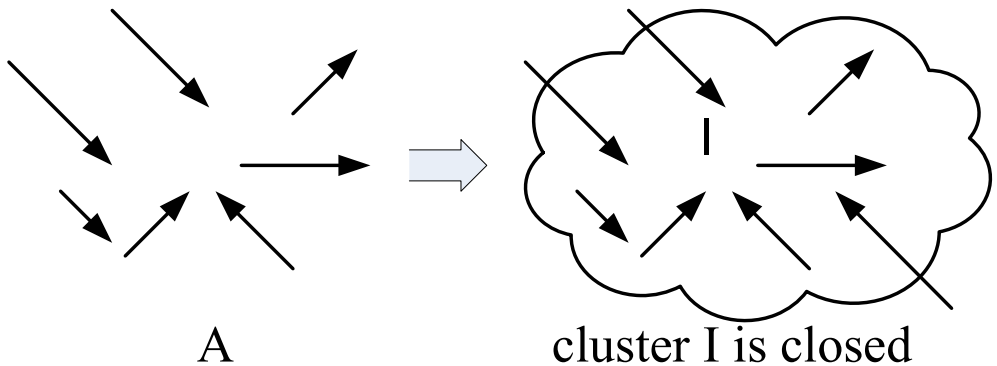
- Define a cluster I to be *closed* under active constraints A when no edge in A point to outside of the cluster I.



Active Constraints

Define (u, v) to be an active constraint for the retiming r if including u but not v in the cluster would violate validity or timing constraints.

- Define a cluster I to be *closed* under active constraints A when no edge in A point to outside of the cluster I.



- r_I is feasible $\Leftrightarrow$ I is closed under all active constraints of r.

Algorithm Sketch

- As more active constraints are introduced in A, either r_l is feasible with reduced FF area or no such l exists.

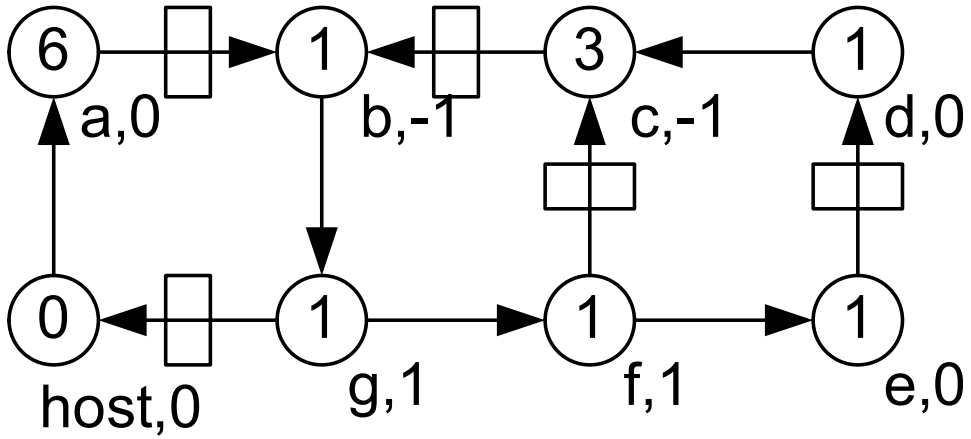
Difficulty for Incremental Algorithm

- How many active constraints are there in A?
  - Keeping every active constraint in A may increase $|A|$ to $\Theta(n^2)$.
  - Therefore, it should be able to remove active constraints from A sometime during the algorithm.
  - Removing them should not affect termination.

Difficulty for Incremental Algorithm

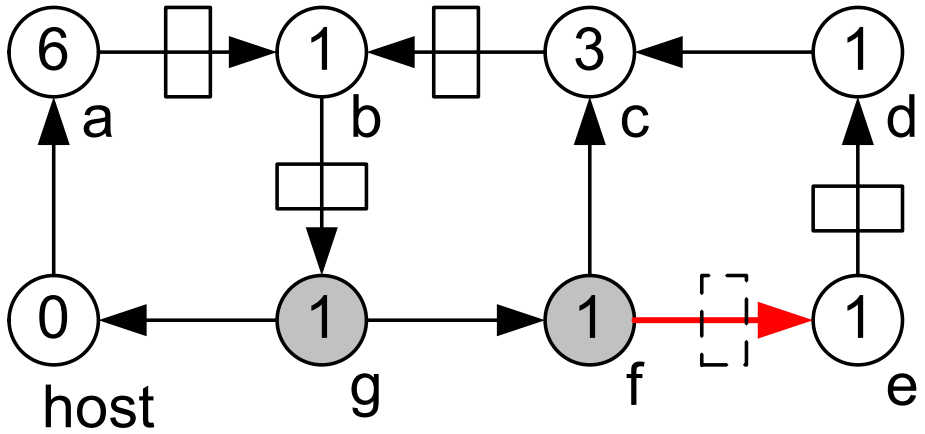
- How many active constraints are there in A?
  - Keeping every active constraint in A may increase $|A|$ to $\Theta(n^2)$.
  - Therefore, it should be able to remove active constraints from A sometime during the algorithm.
  - Removing them should not affect termination.
- Our solution: maintain A as a regular forest.
  - A directed forest satisfying special property.
  - Gates are clustered into trees.
  - Require $\Theta(n)$ storage.

Example



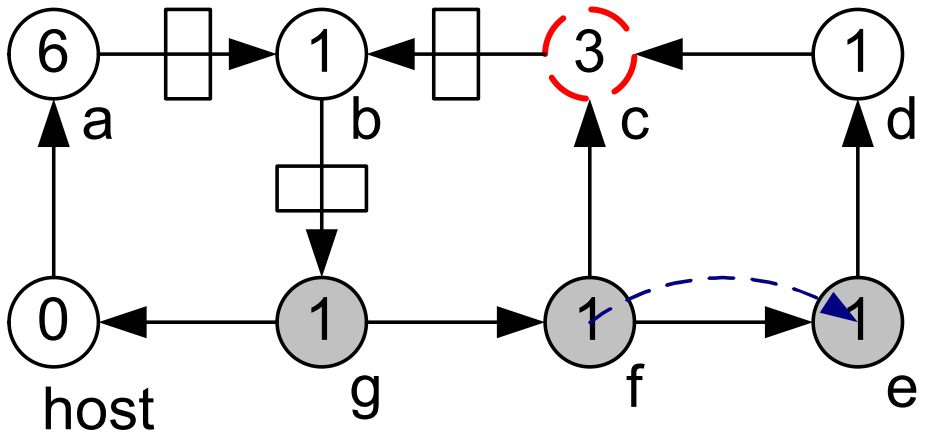
- Clock period $T = 6$.
- Inside each gate are the gate delays.
- On the right to each gate name are the area improvements.

Example



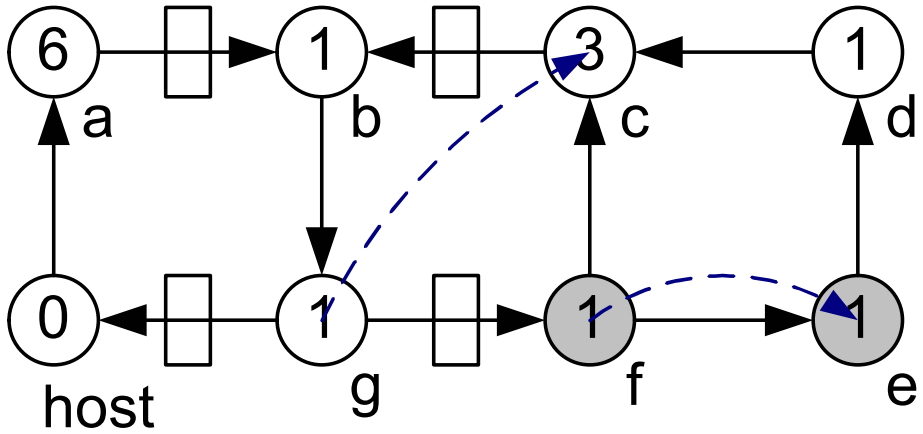
- Active constraints A: $\{\}$.
- Trees: $\{a\}$,$\{b\}$,$\{c\}$,$\{d\}$,$\{e\}$,$\{f\}$,$\{g\}$,$\{\text{host}\}$.
- Move 1 FF across $I = \{f\} \cup \{g\}$.
- # FFs on (f, e) is negative. Add (f, e) to A.

Example



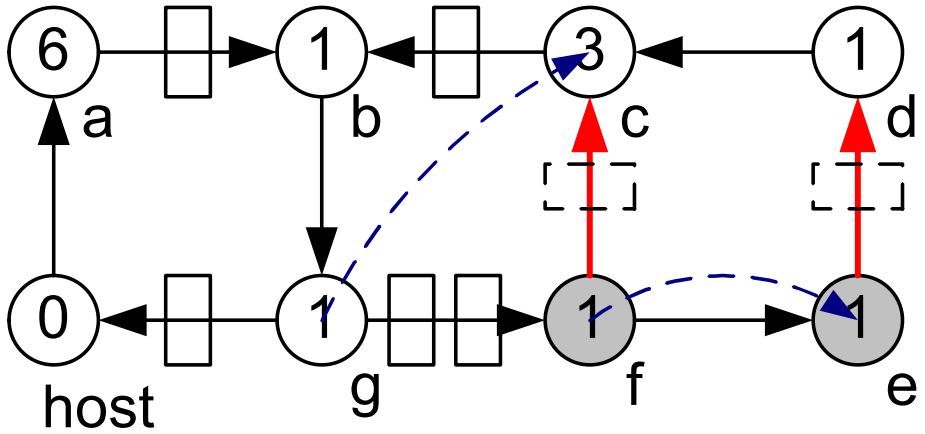
- Active constraints A: $\{(f, e)\}$.
- Trees: $\{a\}$,$\{b\}$,$\{c\}$,$\{d\}$,$\{e, f\}$,$\{g\}$,$\{\text{host}\}$.
- Move 1 FF across $I = \{e, f\} \cup \{g\}$.
- Combinational path from g to c has delay 7. Add (g, c) to A.

Example



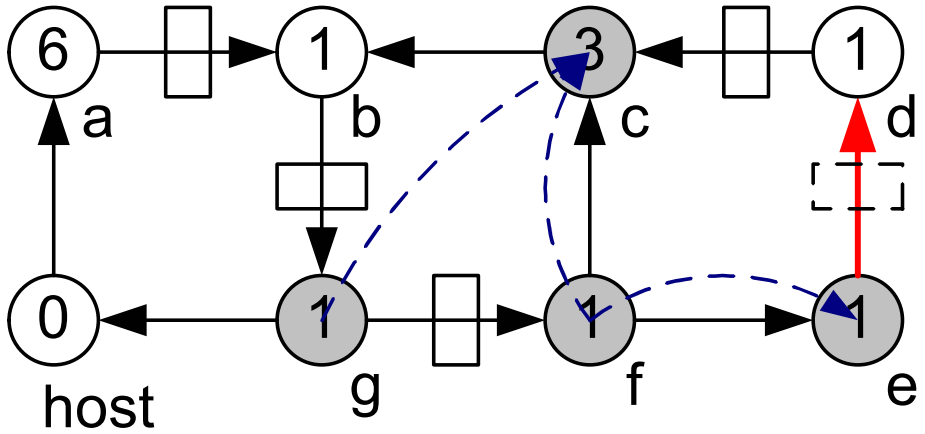
- Active constraints A: $\{(f, e), (g, c)\}$.
- Trees: $\{a\}$,$\{b\}$,$\{c, g\}$,$\{d\}$,$\{e, f\}$,$\{\text{host}\}$.
- Move 1 FF across $I = \{e, f\}$.
- Got a feasible retiming with improved area.

Example



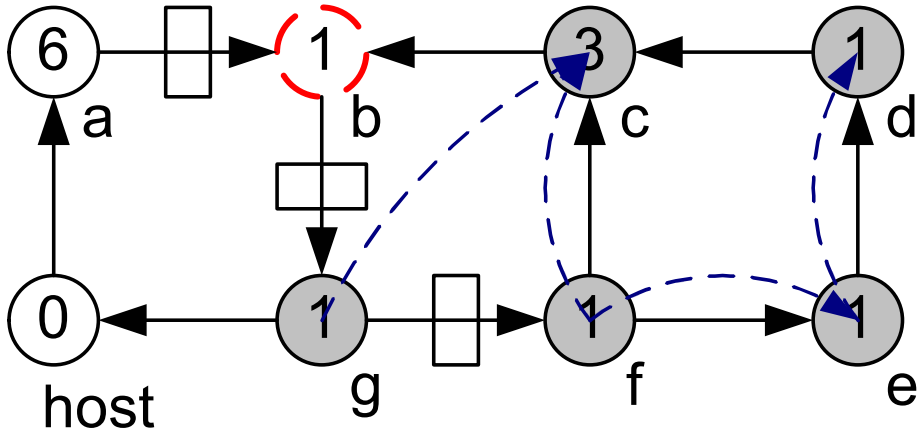
- Active constraints A: $\{(f, e), (g, c)\}$.
- Trees: $\{a\}$,$\{b\}$,$\{c, g\}$,$\{d\}$,$\{e, f\}$,$\{\text{host}\}$.
- Move 1 FF across $I = \{e, f\}$.
- # FFs on (f, c) is negative. Add (f, c) to A.

Example



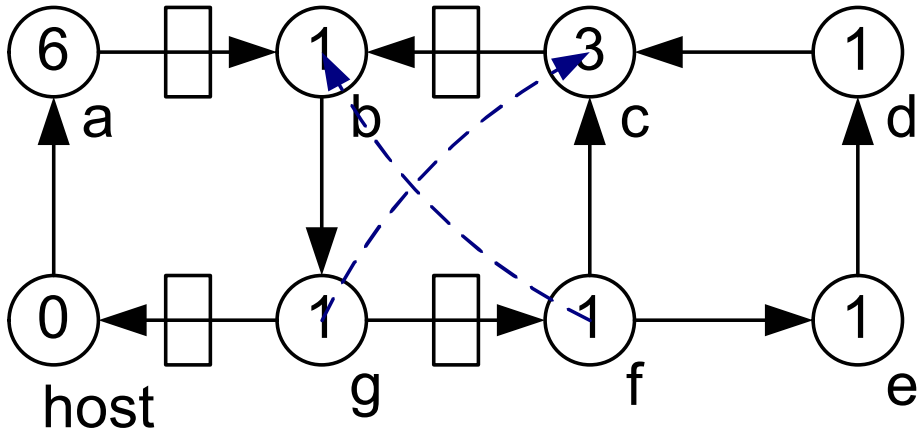
- Active constraints A: $\{(f, e), (g, c), (f, c)\}$.
- Trees: $\{a\}$,$\{b\}$,$\{c, e, f, g\}$,$\{d\}$,$\{\text{host}\}$.
- Move 1 FF across $I = \{c, e, f, g\}$.
- # FFs on (e, d) is negative. Add (e, d) to A.

Example



- Active constraints A: $\{(f, e), (g, c), (f, c), (e, d)\}$.
- Trees: $\{a\}$,$\{b\}$,$\{c, d, e, f, g\}$,$\{\text{host}\}$.
- Move 1 FF across $I = \{c, d, e, f, g\}$.
- Combinational path from f to b has delay 7. Add (f, b) to A.

Example

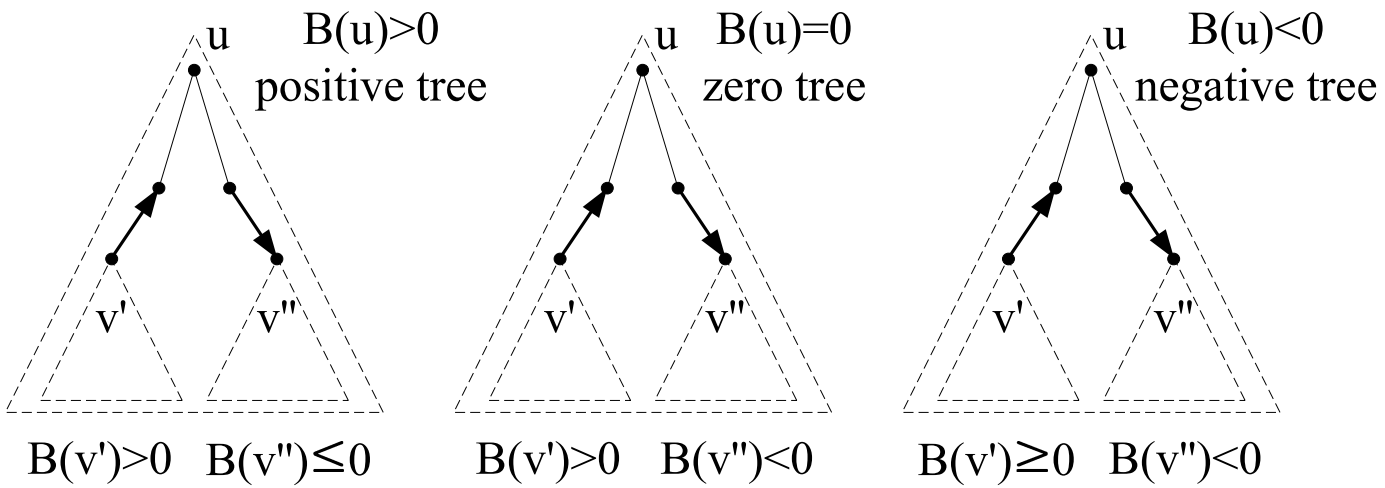


- Active constraints A: $\{(f, b), (g, c)\}$, (others are removed).
- Trees: $\{a\}$,$\{b, f\}$,$\{c, g\}$,$\{d\}$,$\{e\}$,$\{\text{host}\}$.
- Area improvements for all trees are 0.
- The current retiming is the optimal one.

Outline

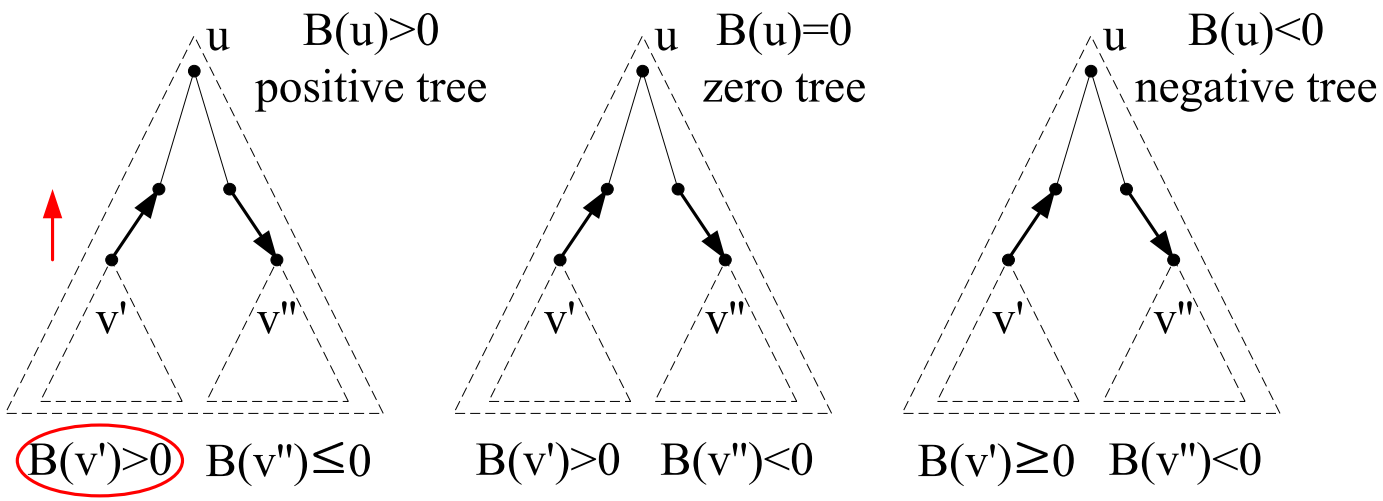
Regular Forest

- Organize active constraints as rooted directed trees.
- Area improvements for each tree and subtree: $B(v)$.
- Move 1 FF across the positive trees.
- Definition of regular forest:
  - Positive trees with $> 0 \uparrow$ and $\leq 0 \downarrow$ subtrees.
  - Zero trees with $> 0 \uparrow$ and $< 0 \downarrow$ subtrees.
  - Negative trees with $\geq 0 \uparrow$ and $< 0 \downarrow$ subtrees.



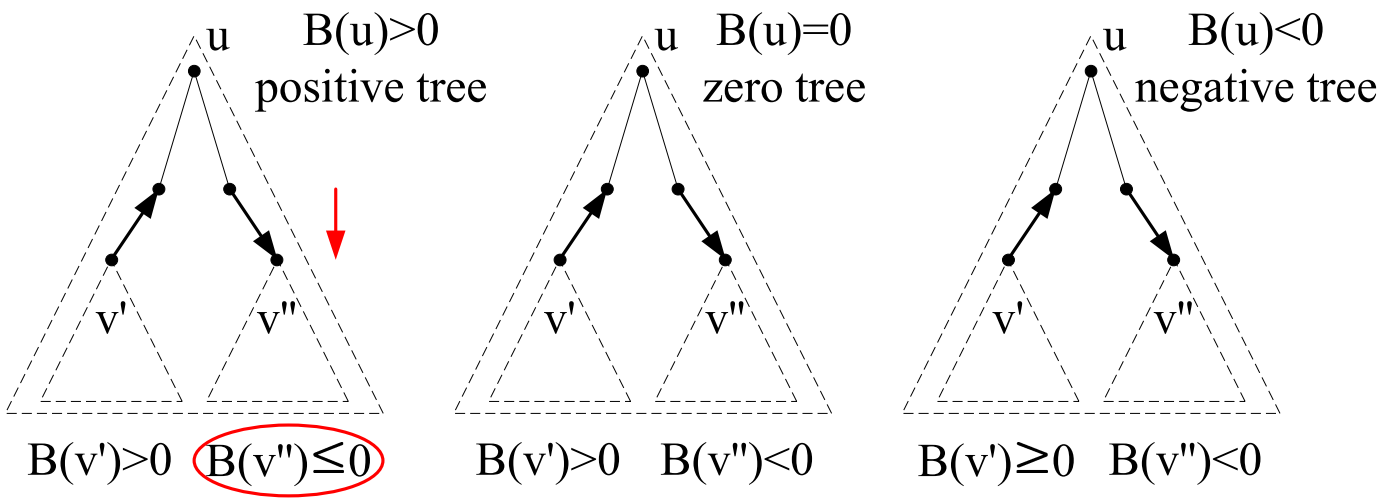
Regular Forest

- Organize active constraints as rooted directed trees.
- Area improvements for each tree and subtree: $B(v)$.
- Move 1 FF across the positive trees.
- Definition of regular forest:
  - Positive trees with $> 0 \uparrow$ and $\leq 0 \downarrow$ subtrees.
  - Zero trees with $> 0 \uparrow$ and $< 0 \downarrow$ subtrees.
  - Negative trees with $\geq 0 \uparrow$ and $< 0 \downarrow$ subtrees.



Regular Forest

- Organize active constraints as rooted directed trees.
- Area improvements for each tree and subtree: $B(v)$.
- Move 1 FF across the positive trees.
- Definition of regular forest:
  - Positive trees with $> 0 \uparrow$ and $\leq 0 \downarrow$ subtrees.
  - Zero trees with $> 0 \uparrow$ and $< 0 \downarrow$ subtrees.
  - Negative trees with $\geq 0 \uparrow$ and $< 0 \downarrow$ subtrees.



Optimality Condition

Theorem

If every tree is a zero tree, then the current retiming is the optimal.

- As there are many active constraints for a retiming, there are many regular forests.
- The retiming is optimal if a regular forest of all zero trees can be constructed.

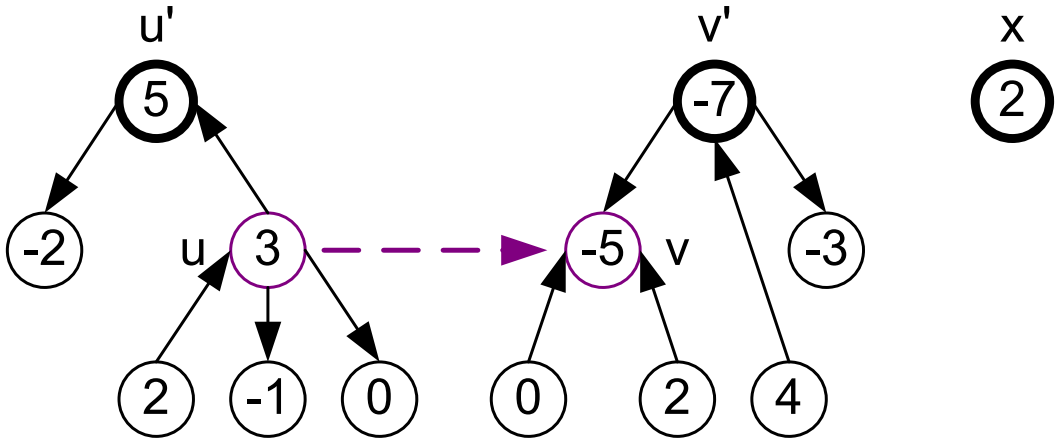
Operation Toward Optimality

- Optimality condition cannot be achieved by a single operation.
- Potential tuple in lexicographic ordering to capture the progress:

$$\Phi \triangleq (\text{area improvement of all the positive trees}, \\ \#\text{ vertices in non-positive trees})$$

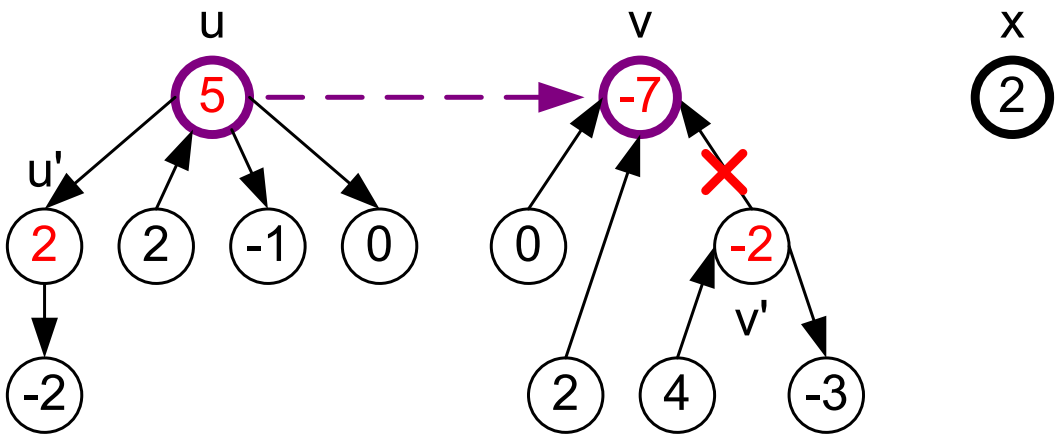
- Optimality $\Leftrightarrow \Phi = (0, n)$.
- Finite number of forests $\Rightarrow$ Finite number of Φ.
- Design the UpdateForest operation to decrease Φ strictly after identifying new active constraint.

The UpdateForest Operation



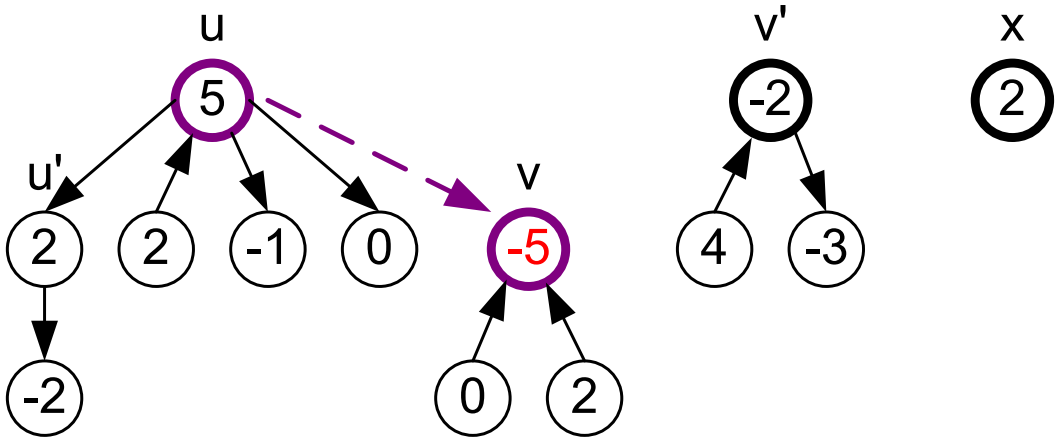
- Inside each gate are the area improvements of subtrees (B).
- A new active constraint (u, v) should be added (later).
- u' and v' are the roots of the two trees in the regular forest.
- Potential tuple $\Phi = (7, 6)$.

The UpdateForest Operation



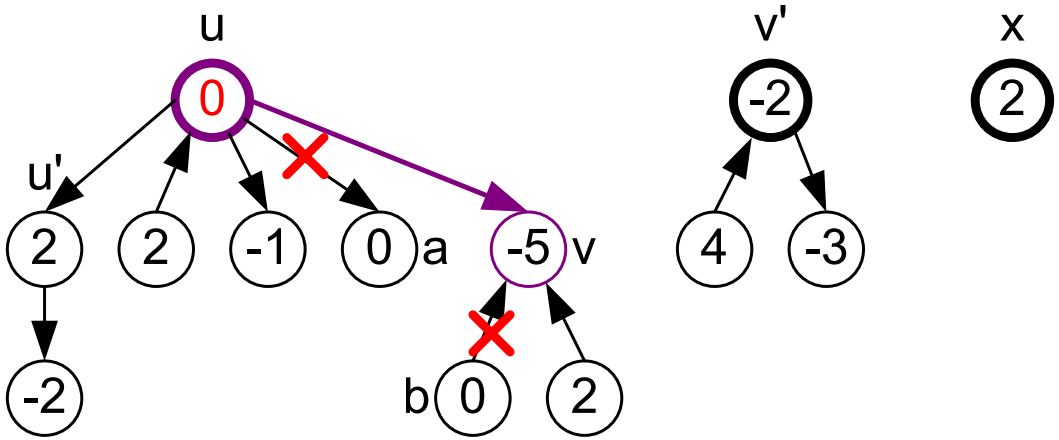
- Change the roots of the trees to u and v.
- $B(u'), B(u), B(v'), B(u)$ should be updated.
- The forest is no longer regular. The subtree rooted at v' is < 0 but $\uparrow$.

The UpdateForest Operation



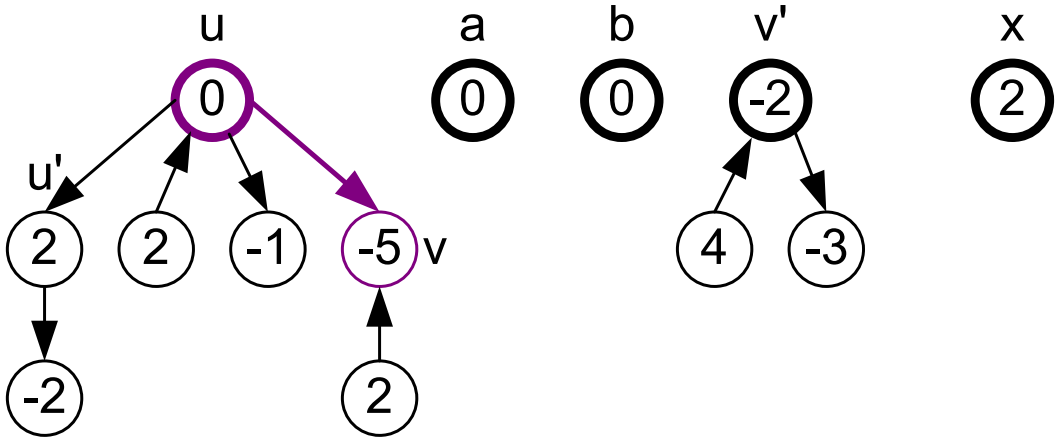
- Removing (v', v) creates a tree with the root v'.
- $B(v)$ should be updated.
- The forest becomes regular.
- Potential tuple $\Phi = (7, 6)$.

The UpdateForest Operation



- Connecting (u, v) makes v a child of u.
- $B(u)$ should be updated.
- The forest is no longer regular. The subtrees rooted at a and b are $= 0$.

The UpdateForest Operation



- Removing (u, a) and (b, v) creates two trees with the root a and b.
- The forest becomes regular.
- Potential tuple $\Phi = (2, 12)$.
- Other cases are handled similarly.

The iMinArea Algorithm

1. Compute an initial feasible retiming r using any fixed period retiming algorithm.
2. Initialize F to be a forest with no edge.
3. $I \leftarrow$ all the nodes of the positive trees in F.
4. Claim r is optimal if $I = \emptyset$.
5. If (u, v) violates the validity or timing constraints in r_I, then update F with (u, v) using the UpdateForest operation. Continue to Step 3.
6. Update r to r_I. Continue to Step 3.

Complexity Analysis

- Need $O(n)$ extra space for the regular forest and other auxiliary data structures on top of the circuit graph G.
- Each iteration consumes $O(m)$ time.
- For practical VLSI circuits, assume area improvements b are integer-valued, summation of positive b is bounded by $O(n)$, and FF area in the initial feasible retiming is bounded by $O(m)$. The time complexity is $O(n^2 m)$.

Complexity Analysis

- Need $O(n)$ extra space for the regular forest and other auxiliary data structures on top of the circuit graph G.
- Each iteration consumes $O(m)$ time.
- For practical VLSI circuits, assume area improvements b are integer-valued, summation of positive b is bounded by $O(n)$, and FF area in the initial feasible retiming is bounded by $O(m)$. The time complexity is $O(n^2 m)$.
- For general graphs, the algorithm terminates in finite time for bounded problems.
  - Note that termination is guaranteed for real valued b.

Outline

Experimental Setup

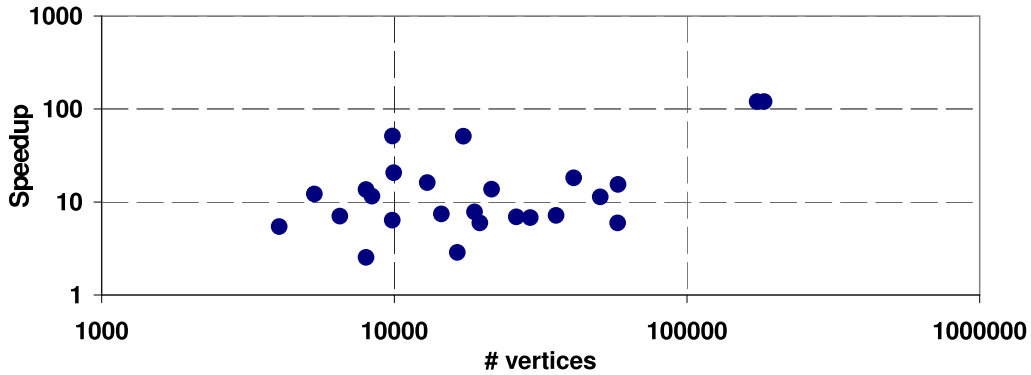
- Benchmarks.
  - ISCAS89 sequential circuits.
  - Large circuits (myex1 through myex5) created in Minaret.
  - ITC'99 sequential circuits.
- The largest circuit has $>$ 180K gates and $>$ 320K edges.
- For comparison to Minaret, assume unit FF area, unit gate delay, and sharing of FFs at the fanouts of gates.
- Use Zhou's algorithm [Zhou 05] to determine the minimum clock period.
- Perform min-area retiming under the minimum clock period.

Experimental Results

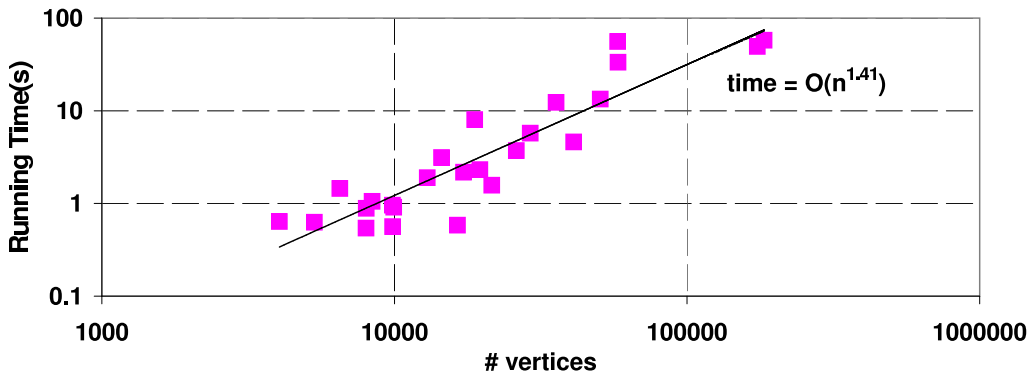
- FF area (# FFs) obtained by Minaret and that obtained by iMinArea are the same.
- Compared to Minaret, 60× faster on average.
- For the circuit with more than 180K gates, iMinArea uses less than 1 minute and 65MB memory.
  - Compared to Minaret, $\geq$ 30× less memory consumption and $\geq$ 100× faster.

Experimental Results

Speedups for circuits with > 4000 gates compared to Minaret:



Running times:



Outline

Conclusions

- Efficient incremental min-area retiming algorithm with provable optimality.

Conclusions

- Efficient incremental min-area retiming algorithm with provable optimality.
- Combine incremental retiming with a special forest data structure.
  - Will not formulate the dual problem of min-cost network flow.
  - Generate critical constraints dynamically only when they are needed.
  - Require only linear storage ($O(n)$) on top of circuit graph.

Conclusions

- Efficient incremental min-area retiming algorithm with provable optimality.
- Combine incremental retiming with a special forest data structure.
  - Will not formulate the dual problem of min-cost network flow.
  - Generate critical constraints dynamically only when they are needed.
  - Require only linear storage ($O(n)$) on top of circuit graph.
- Extensions
  - Perform forward retiming to enforce initial state.
  - Min-area retiming under hold conditions.

Q & A

Thank you!

Optimality Condition

Lemma

If (u, v) is an active constraint of r, then for any feasible r',

$$r'(v) - r'(u) \geq r(v) - r(u).$$

Lemma

If every tree is a zero tree, then $\exists \gamma(u, v) \geq 0$ such that,

$$b(v) = \sum_{(v,j) \in A} \gamma(v, j) - \sum_{(i,v) \in A} \gamma(i, v), \forall v \in V.$$

Optimality Condition

Theorem

If every tree is a zero tree, then the current retiming is the optimal.

Proof sketch

For any feasible r',

$$\begin{aligned}
\sum_{v \in V} b(v) r(v) &= \sum_{(u,v) \in A} \gamma(u,v) * \big(r(u) - r(v)\big) \\
&\geq \sum_{(u,v) \in A} \gamma(u,v) * \big(r'(u) - r'(v)\big) \\
&= \sum_{v \in V} b(v) r'(v)
\end{aligned}$$