Outstanding unsolved problems demand new methods for their solution, while powerful new methods beget new problems to be solved. But, as Poincare observed, it is the man, not the method, that solves a problem. –E. T. Bell ("Men of Mathematics")

Optimization Algorithms and Parallel Programming in Physical and Logic Synthesis

Prof. Hai Zhou
EECS
Northwestern University

25 Jul 2009

Outline

System-on-Chip

- Moore's Law: number of transistor doubles per generation
- Market requires more functionality on a chip
- System-on-Chip is natural both from supply and demand

Design Crisis

- Design productivity lags far behind the technology
  "What shall we use so many transistors for?"

Design Crisis

- Design productivity lags far behind the technology
  "What shall we use so many transistors for?"
- System Complexity: huge amount of functionality

Design Crisis

- Design productivity lags far behind the technology
  "What shall we use so many transistors for?"
- System Complexity: huge amount of functionality
- Silicon complexity: more physical phenomena to be modeled and considered

Communication Among Components

- Increasing frequencies and die sizes
- Shrinking gate delays
- Interconnect delay dominates circuit performance
- Interconnect optimizations such as buffering are universally applied
- Global communication requests multiple clock cycles

Noise and Crosstalk

- Increasing aspect ratios of wires
- Decreasing distances between wires
- Capacitive and inductive couplings among wires
- Noises are induced on quiet wires by switching wires
- Wire delays are changing because of crosstalk
- Analog components are sensitive to noises

Power Consumption

- Leakage has become prominent for current and future technology
- Excessive power consumption shortens battery life
- It also increase the cost and the stress of packaging

Thermal Issues

- Excessive power consumption increases temperatures on chip
- Uneven power consumption increases thermal gradients
- High temperature decreases performance and reliability
- It also increases packaging cost
- Leakage increases with high temperatures

Manufacturability

- Many new phenomena and issues in nano-lithography:

Manufacturability

- Many new phenomena and issues in nano-lithography:
- Process variations: random fabrication outcomes
- Resolution Enhancement Techniques (RET)
- Antenna effects
- ...

SoC Design Requirements

SoC Design Requirements

- Modeling and analysis techniques

SoC Design Requirements

- Modeling and analysis techniques
- Design optimization techniques

Outline

Wire pipelining

- VLSI scaling trend
  - Frequency: 2X/generation, Die size: 1.25X/generation
  - Problem: global communication requires multiple clock periods

Wire pipelining

- VLSI scaling trend
  - Frequency: 2X/generation, Die size: 1.25X/generation
  - Problem: global communication requires multiple clock periods
- Recent research
  - Insert flip-flops (FFs) on wires based on physical needs (Intel, IBM, etc.)

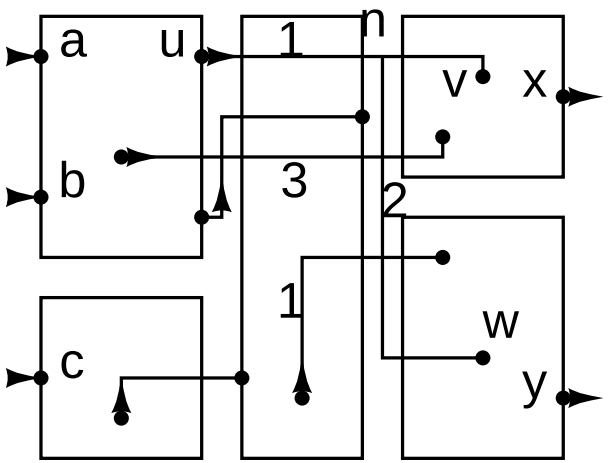
Wire pipelining

- VLSI scaling trend
  - Frequency: 2X/generation, Die size: 1.25X/generation
  - Problem: global communication requires multiple clock periods
- Recent research
  - Insert flip-flops (FFs) on wires based on physical needs (Intel, IBM, etc.)
  - How to maintain logical (functional) correctness?
    - FF insertion changes computation schedule
    - Synchronization among different computation units may be destroyed

Wire pipelining by retiming

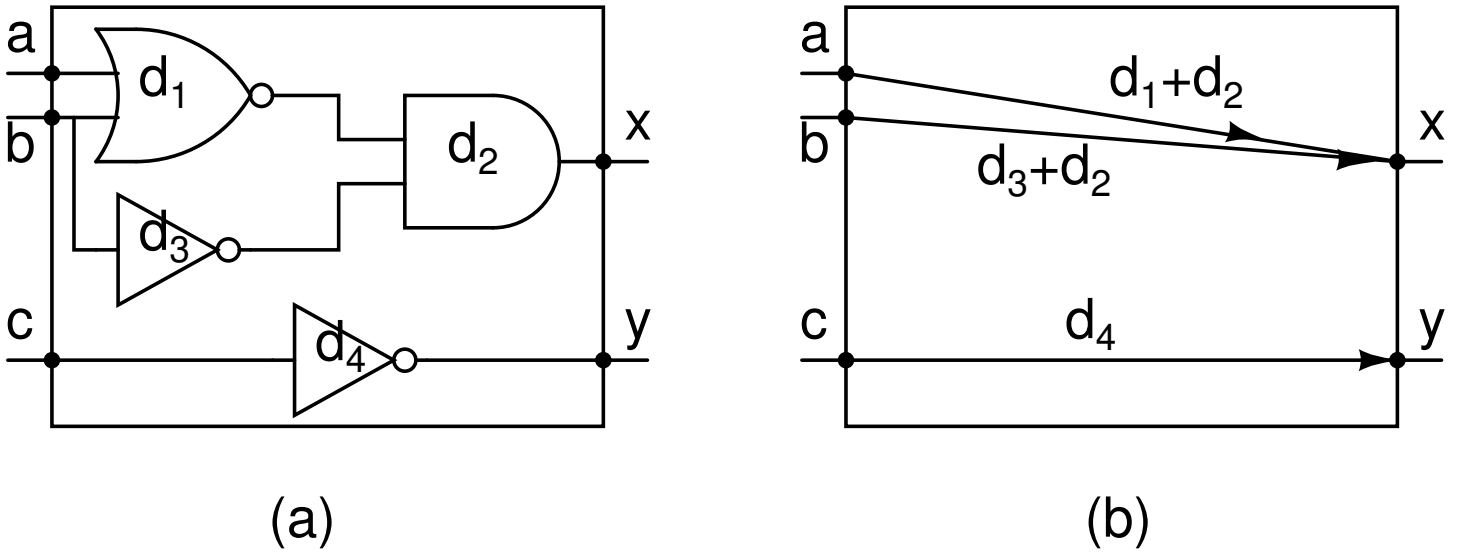
- Retiming [Leiserson and Saxe '83] relocates FFs w/o changing functionality
  - Re-scheduling computation
- We extend it for pipelining long wires
  - Re-scheduling both computation and communication
- FFs may be added at PI (or PO) and then retimed into the circuit

An SOC design example



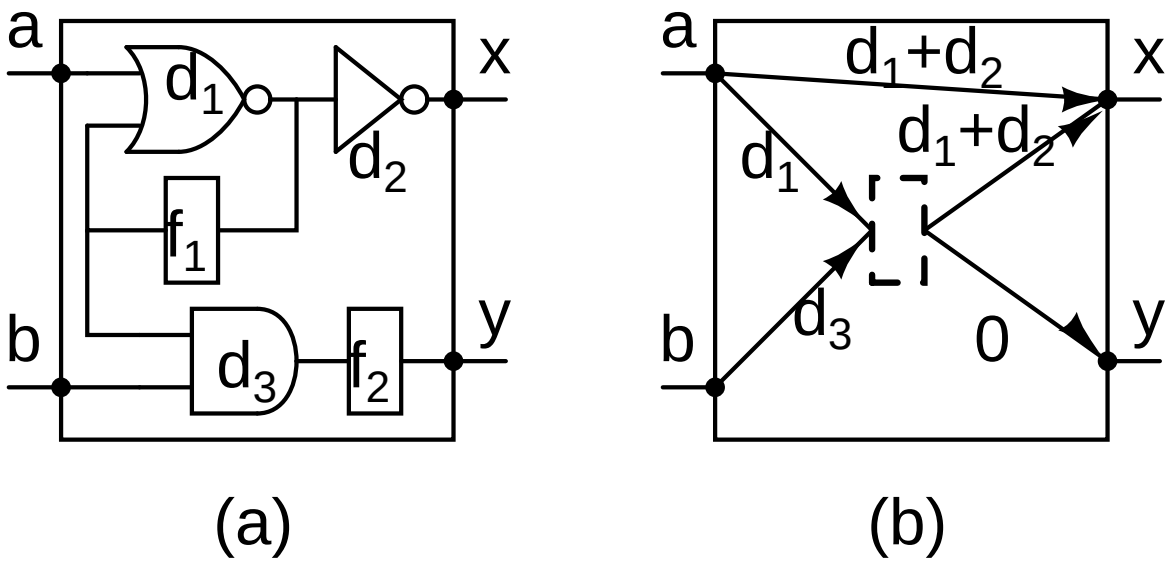
- Block placement and global routing are given
- Signal directions and register locations, too

Timing model for a combinational block



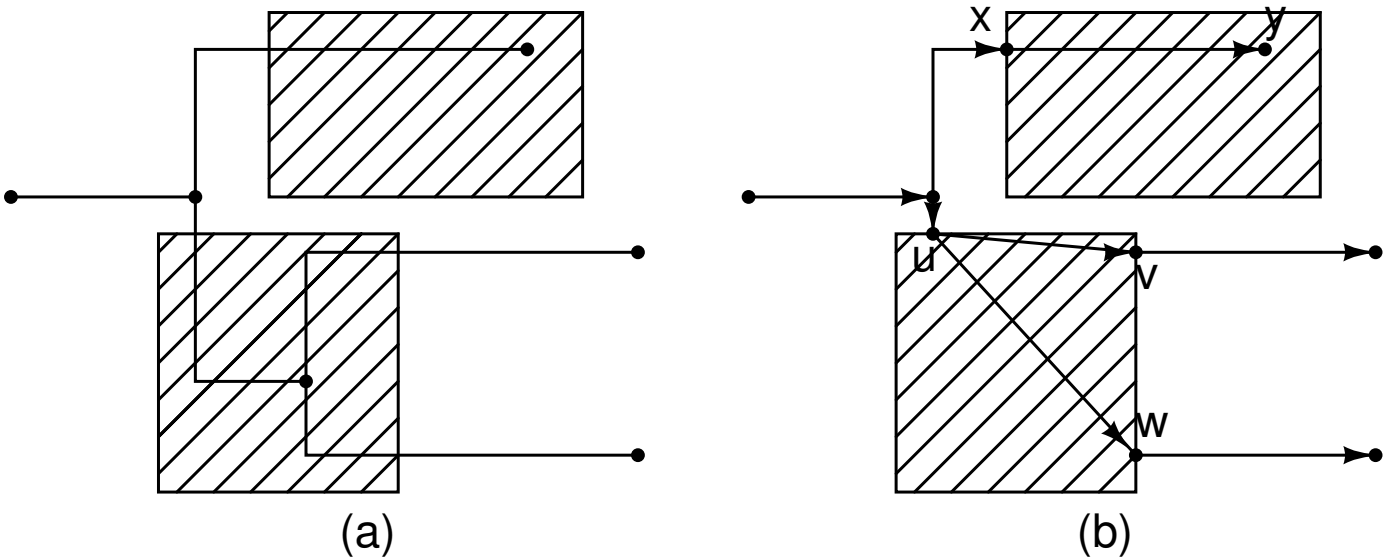
- Timing arrows represent pin-to-pin path delays

Timing model for a sequential block



- Timing arrows for pin-to-pin combinational paths
- A virtual register introduced for other paths
  - Paths starting or ending at registers

Timing model for a net



- Nodes for Steiner points
- Nodes for entrances and exits of buffer-forbidden areas

Optimal wire retiming problem

- $G = (V, E)$, $E = E_1 \cup E_2$, $E_1 \cap E_2 = \emptyset$
  delay: $d(e)$, #FF: $w(e)$, $\forall e \in E$

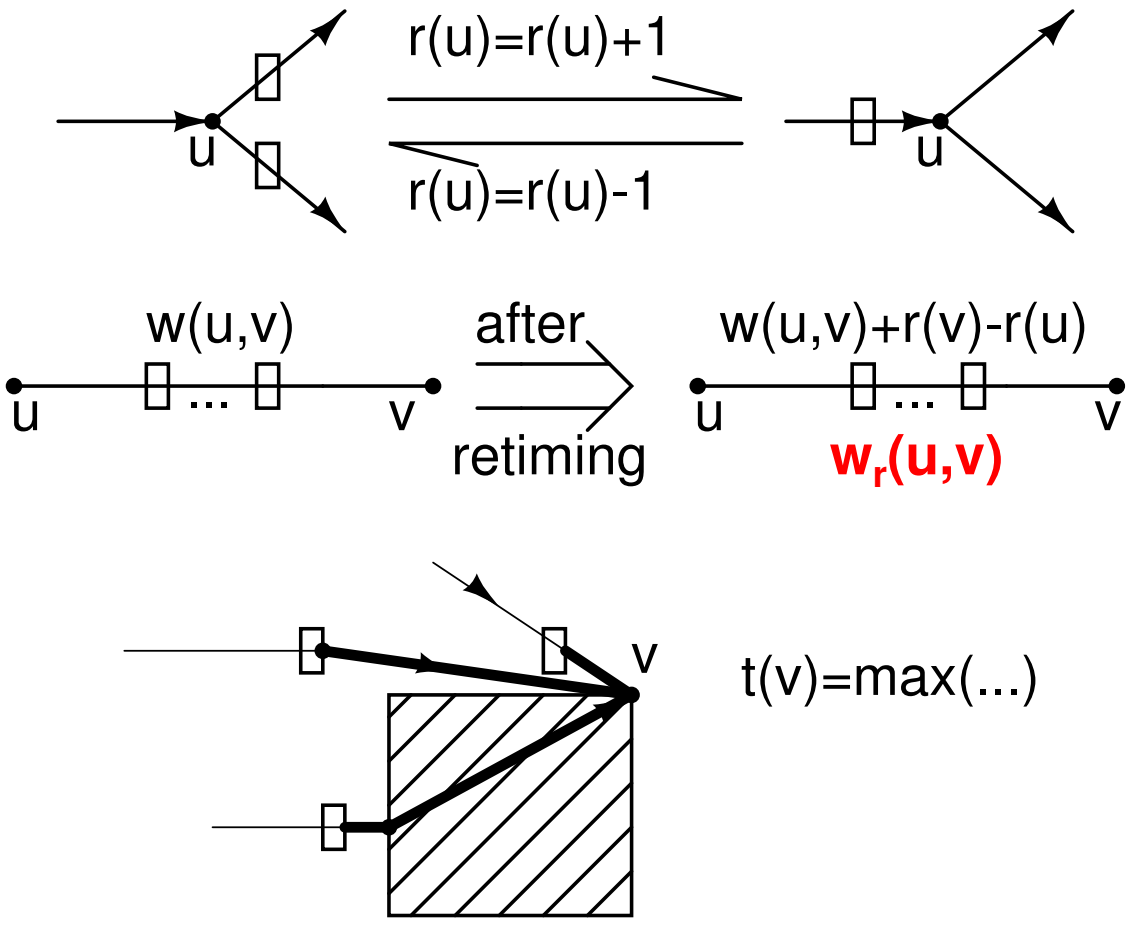
Optimal wire retiming problem

- $G = (V, E)$, $E = E_1 \cup E_2$, $E_1 \cap E_2 = \emptyset$
  delay: $d(e)$, #FF: $w(e)$, $\forall e \in E$
- $\forall e \in E_2$, $d(e)$ is proportional to its length
  - Since buffers are allowed on E_2

Optimal wire retiming problem

- $G = (V, E)$, $E = E_1 \cup E_2$, $E_1 \cap E_2 = \emptyset$
  delay: $d(e)$, #FF: $w(e)$, $\forall e \in E$
- $\forall e \in E_2$, $d(e)$ is proportional to its length
  - Since buffers are allowed on E_2
- Find relocation of FFs
  - No FFs changed on any $e \in E_1$
  - Minimize clock period (= the maximum delay between any two consecutive FFs)

Introducing decision variables r and t



Formal problem formulation

Minimize T subject to:

Formal problem formulation

Minimize T subject to:

- Retiming validity

$$r(u) = r(v) \qquad \forall (u,v) \in E_1 \quad (1)$$

$$w_r(u,v) = w(u,v) + r(v) - r(u) \geq 0 \qquad \forall (u,v) \in E_2 \quad (2)$$

Formal problem formulation

Minimize T subject to:

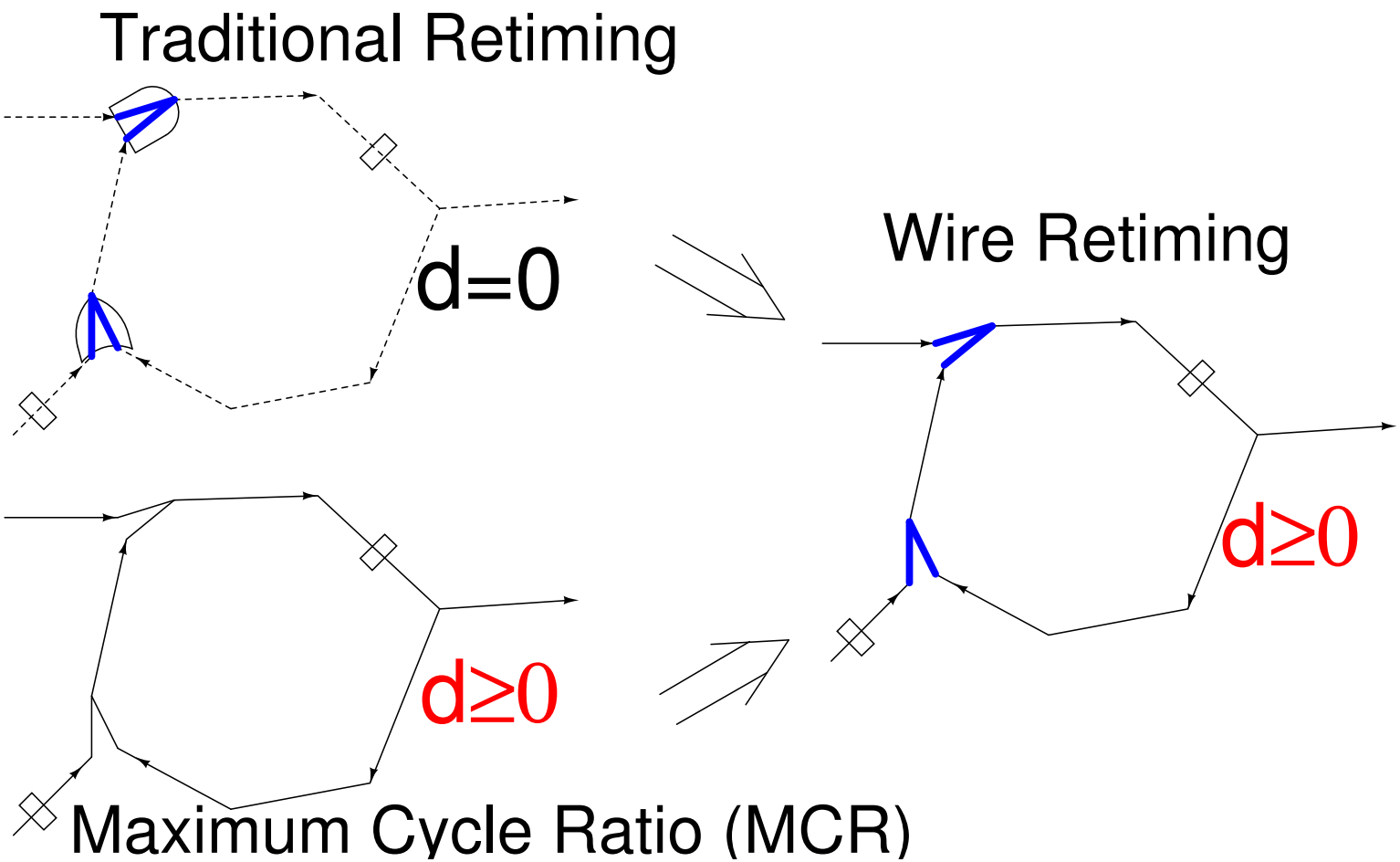
- Retiming validity

$$r(u) = r(v) \qquad \forall (u,v) \in E_1 \qquad (1)$$
$$w_r(u,v) = w(u,v) + r(v) - r(u) \geq 0 \qquad \forall (u,v) \in E_2 \qquad (2)$$

- Timing validity

$$t(v) \geq t(u) + d(u,v) - w_r(u,v)T \qquad \forall (u,v) \in E \qquad (3)$$
$$0 \leq t(v) \leq T \qquad \forall v \in V \qquad (4)$$

Algorithmic view of the problem



Traditional retiming problem

$$r(u) = r(v),\ \forall (u,v) \in E_1 \quad (1)$$

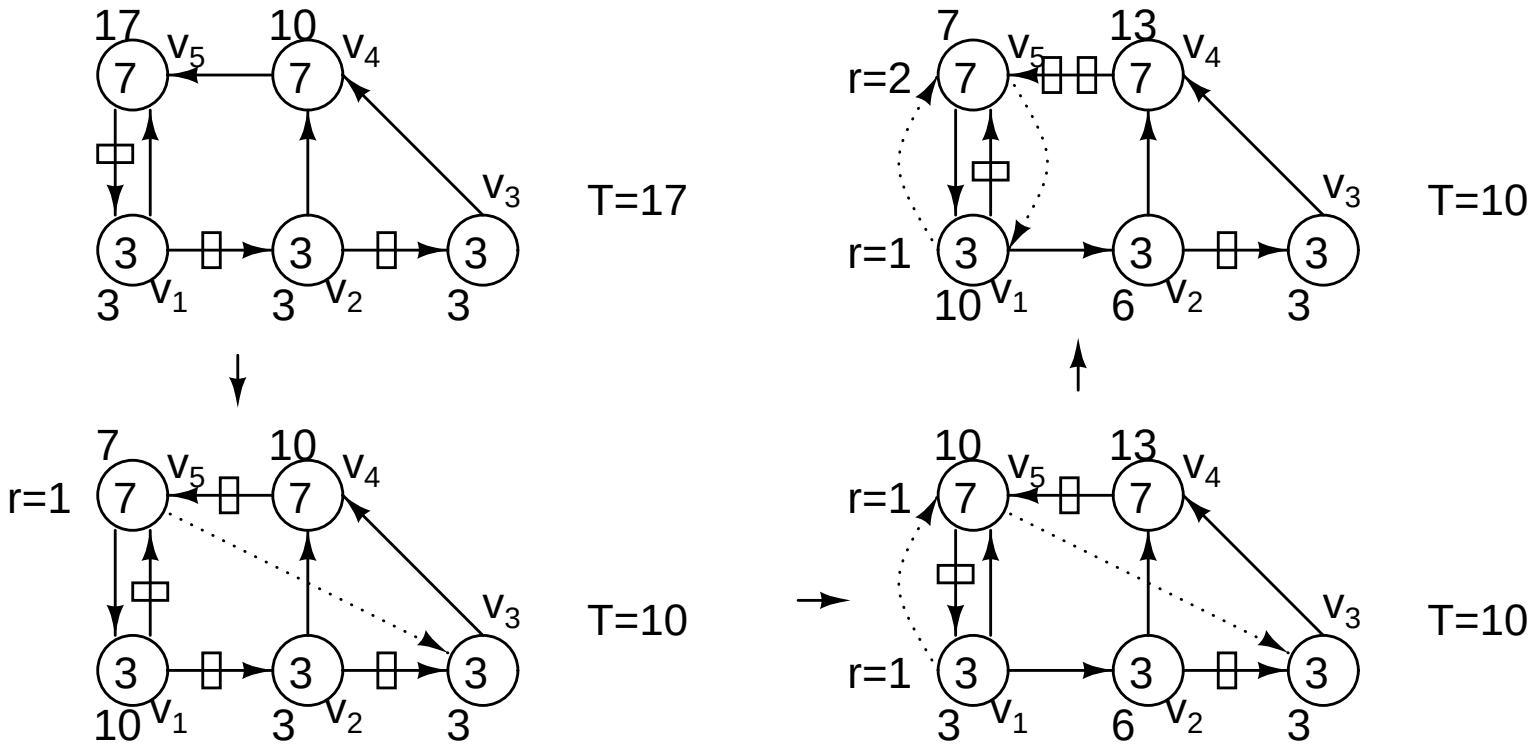
$$w_r(u,v) = w(u,v) + r(v) - r(u) \geq 0,\ \forall (u,v) \in E_2 \quad (2)$$

$$t(v) \geq t(u) + d(u,v),\ \forall (u,v) \in E : w_r(u,v) = 0$$

$$0 \leq t(v) \leq T,\ \forall v \in V \quad (4)$$

Zhou's algorithm [ASP-DAC'05]

- Solve traditional retiming incrementally w/o binary search:
  - Initialize T by $r = 0$
  - Iteratively increment $r(v)$ for $t(v) \geq T$
  - Maintain m pointers for optimality checking



Maximum cycle ratio problem

Minimize T subject to:

$$t(v) \geq t(u) + d(u,v) - w_r(u,v)T \qquad \forall (u,v) \in E(3)$$

Maximum cycle ratio problem

Minimize T subject to:

$$t(v) \geq t(u) + d(u,v) - w_r(u,v)T \qquad \forall (u,v) \in E(3)$$

- Burns's algorithm [CalTech PhD thesis '91]
- Solve MCR problem by iteratively pushing down T

Idea for solving wire retiming problem

- Initialize T with $r = 0$

Idea for solving wire retiming problem

- Initialize T with $r = 0$
- Iteratively reduce T while keeping (1)-(4)

Idea for solving wire retiming problem

- Initialize T with $r = 0$
- Iteratively reduce T while keeping (1)-(4)
  - With r unchanged
    - Extend Burns's algorithm

Idea for solving wire retiming problem

- Initialize T with $r = 0$
- Iteratively reduce T while keeping (1)-(4)
  - With r unchanged
    - Extend Burns's algorithm
  - Change r (retiming)
    - Extend Zhou's algorithm

Idea for solving wire retiming problem

- Initialize T with $r = 0$
- Iteratively reduce T while keeping (1)-(4)
  - With r unchanged
    - Extend Burns's algorithm
  - Change r (retiming)
    - Extend Zhou's algorithm
- Certify optimality

Push down T with r unchanged

- Retiming validity ((1) and (2)) is kept

Push down T with r unchanged

- Retiming validity ((1) and (2)) is kept
- Minimize T under timing validity:

$$t(v) \geq t(u) + d(u,v) - w_r(u,v)T, \ \forall (u,v) \in E \qquad (3)$$

$$0 \leq t(v) \leq T, \ \forall v \in V \qquad (4)$$

Push down T with r unchanged

- Retiming validity ((1) and (2)) is kept
- Minimize T under timing validity:

$$t(v) \geq t(u) + d(u,v) - w_r(u,v)T, \ \forall (u,v) \in E \qquad (3)$$

Push down T with r unchanged

- Retiming validity ((1) and (2)) is kept
- Minimize T under timing validity:

$$t(v) \geq t(u) + d(u,v) - w_r(u,v)T, \ \forall (u,v) \in E \qquad (3)$$

- Burns's algorithm [CalTech PhD thesis '91]
  - Returns minimal T under (3)

Push down T with r unchanged

- Retiming validity ((1) and (2)) is kept
- Minimize T under timing validity:

$$t(v) \geq t(u) + d(u,v) - w_r(u,v)T, \ \forall (u,v) \in E \qquad (3)$$

$$0 \leq t(v) \leq T, \ \forall v \in V \qquad (4)$$

- Burns's algorithm [CalTech PhD thesis '91]
  - Returns minimal T under (3)

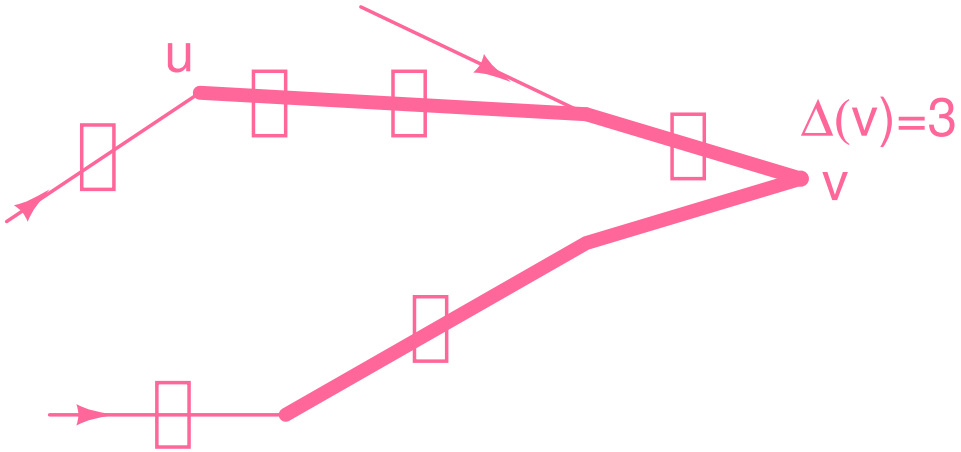
Extend Burns's to incorporate (4)

Burns's Algorithm

1. While (true)
2. $E_c \leftarrow \{(u,v) \in E \mid t(v) = t(u) + d(u,v) - w_r(u,v)T\}$;
3. Return T and r, if E_c contains a cycle;

Burns's Algorithm

1. While (true)
2. $E_c \leftarrow \{(u,v) \in E \mid t(v) = t(u) + d(u,v) - w_r(u,v)T\}$;
3. Return T and r, if E_c contains a cycle;
4. For $v \in V$ in topological sort order of $G_c = (V, E_c)$ do
5. $\Delta(v) \leftarrow 0$, if v is a root in G_c;
6. $\Delta(v) \leftarrow \max_{\forall(u,v)\in E_c}\{\Delta(v),\ \Delta(u) + w_r(u,v)\}$;



Burns's Algorithm

1. While (true)
2. $E_c \leftarrow \{(u,v) \in E \mid t(v) = t(u) + d(u,v) - w_r(u,v)T\}$;
3. Return T and r, if E_c contains a cycle;
4. For $v \in V$ in topological sort order of $G_c = (V, E_c)$ do
5. $\Delta(v) \leftarrow 0$, if v is a root in G_c;
6. $\Delta(v) \leftarrow \max_{\forall (u,v) \in E_c}\{\Delta(v),\ \Delta(u) + w_r(u,v)\}$;
7. $\theta \leftarrow \infty$;
8. For each $(u,v) \in E$ do
9. If $(\Delta(u) + w_r(u,v) > \Delta(v))$ then
10. $\theta \leftarrow \min\{\theta,\ \frac{t(v)-t(u)-d(u,v)+w_r(u,v)T}{\Delta(u)+w_r(u,v)-\Delta(v)}\}$;

Burns's Algorithm

1. While (true)
2. $E_c \leftarrow \{(u,v) \in E \mid t(v) = t(u) + d(u,v) - w_r(u,v)T\}$;
3. Return T and r, if E_c contains a cycle;
4. For $v \in V$ in topological sort order of $G_c = (V, E_c)$ do
5. $\Delta(v) \leftarrow 0$, if v is a root in G_c;
6. $\Delta(v) \leftarrow \max_{\forall (u,v) \in E_c}\{\Delta(v),\ \Delta(u) + w_r(u,v)\}$;
7. $\theta \leftarrow \infty$;
8. For each $(u,v) \in E$ do
9. If $(\Delta(u) + w_r(u,v) > \Delta(v))$ then
10. $\theta \leftarrow \min\{\theta,\ \frac{t(v)-t(u)-d(u,v)+w_r(u,v)T}{\Delta(u)+w_r(u,v)-\Delta(v)}\}$;
11. $T \leftarrow T - \theta$
12. For each $v \in V$ do
13. $t(v) \leftarrow t(v) + \theta \cdot \Delta(v)$;

Modify Burns's to satisfy (4)

1. While (true)
2. $E_c \leftarrow \{(u,v) \in E \mid t(v) = t(u) + d(u,v) - w_r(u,v)T\}$;
3. Return T and r, if E_c contains a cycle;
4. For $v \in V$ in topological sort order of $G_c = (V, E_c)$ do
5. $\Delta(v) \leftarrow 0$, if v is a root in G_c;
6. $\Delta(v) \leftarrow \max_{\forall(u,v)\in E_c}\{\Delta(v),\ \Delta(u) + w_r(u,v)\}$;
7. $\theta \leftarrow \infty$;
8. For each $(u,v) \in E$ do
9. If $(\Delta(u) + w_r(u,v) > \Delta(v))$ then
10. $\theta \leftarrow \min\{\theta,\ \frac{t(v)-t(u)-d(u,v)+w_r(u,v)T}{\Delta(u)+w_r(u,v)-\Delta(v)}\}$;
11. $T \leftarrow T - \theta$
12. For each $v \in V$ do
13. $t(v) \leftarrow t(v) + \theta \cdot \Delta(v)$;

Modify Burns's to satisfy (4)

1. While (true)
2. $E_c \leftarrow \{(u,v) \in E \mid t(v) = t(u) + d(u,v) - w_r(u,v)T\}$;
3. Return T and r, if E_c contains a cycle;
4. For $v \in V$ in topological sort order of $G_c = (V, E_c)$ do
5. $\Delta(v) \leftarrow 0$, if v is a root in G_c;
6. $\Delta(v) \leftarrow \max_{\forall (u,v) \in E_c}\{\Delta(v),\ \Delta(u) + w_r(u,v)\}$;
7. $\theta \leftarrow \infty$;
8. For each $(u,v) \in E$ do
9. If $(\Delta(u) + w_r(u,v) > \Delta(v))$ then
10. $\theta \leftarrow \min\{\theta,\ \frac{t(v)-t(u)-d(u,v)+w_r(u,v)T}{\Delta(u)+w_r(u,v)-\Delta(v)}\}$;
11. For each $v \in V$ do
12. $\theta \leftarrow \min\{\theta, \frac{T-t(v)}{\Delta(v)+1}\}$;
13. $T \leftarrow T - \theta$
14. For each $v \in V$ do

Modify Burns's to satisfy (4)

Theoretical importance

Push T down to the minimum, with r unchanged

Push down T by changing r

- Condition
  - $\exists v,\ t(v) = T$

Push down T by changing r

- Condition
  - $\exists v,\ t(v) = T$
- Zhou's algorithm
  - $r(v) \leftarrow r(v) + 1$
    - Necessary to get a smaller T if it exists

Push down T by changing r

- Condition
  - $\exists v,\ t(v) = T$
- Zhou's algorithm
  - $r(v) \leftarrow r(v) + 1$
    - Necessary to get a smaller T if it exists
  - Regain retiming validity ((1) and (2))
    - Proper r adjustments

Push down T by changing r

- Condition
  - $\exists v,\ t(v) = T$
- Zhou's algorithm
  - $r(v) \leftarrow r(v) + 1$
    - Necessary to get a smaller T if it exists
  - Regain retiming validity ((1) and (2))
    - Proper r adjustments
- Run extended Burns's under new r

Criteria for certifying optimality

Optimality has been reached if:

- A critical cycle in Burns's
- An m cycle in Zhou's
- $\exists v \in V$, $r(v) > N_{ff}$, the total # of FFs in any simple path

Criteria for certifying optimality

Optimality has been reached if:

- A critical cycle in Burns's
- An m cycle in Zhou's
- $\exists v \in V$, $r(v) > N_{\text{ff}}$, the total # of FFs in any simple path

Criteria for certifying optimality

Optimality has been reached if:

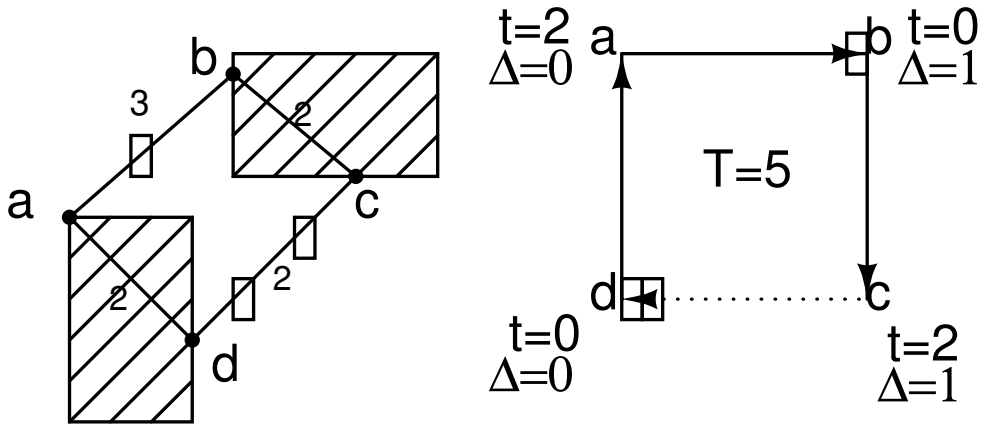
- A critical cycle in Burns's
- An m cycle in Zhou's
- $\exists v \in V$, $r(v) > N_{\text{ff}}$, the total # of FFs in any simple path

Criteria for certifying optimality

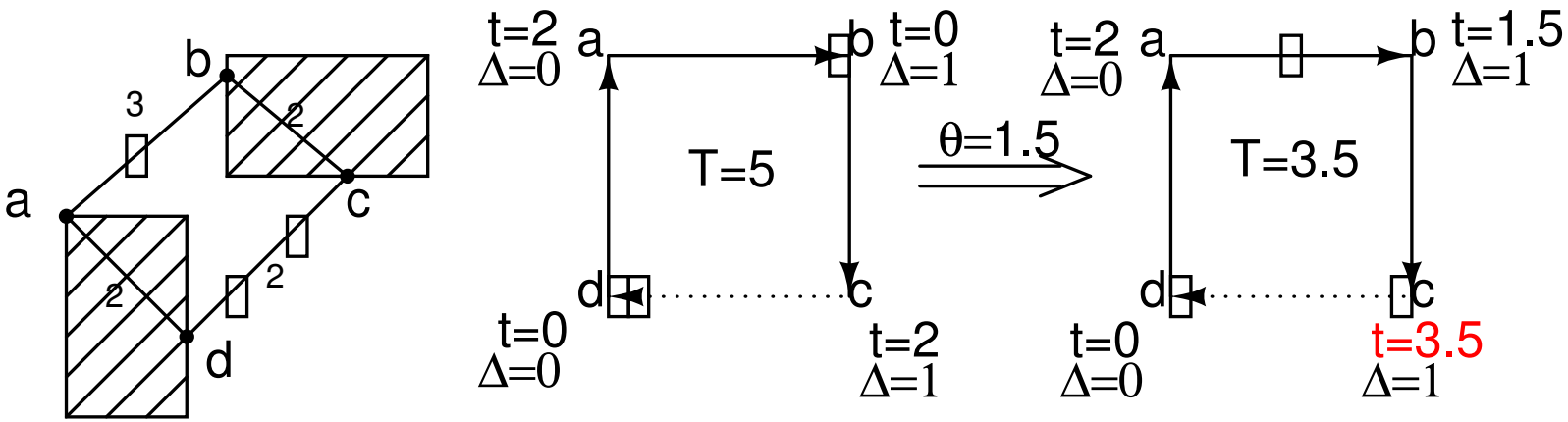
Optimality has been reached if:

- A critical cycle in Burns's
- An m cycle in Zhou's
- $\exists v \in V$, $r(v) > N_{\mathrm{ff}}$, the total # of FFs in any simple path

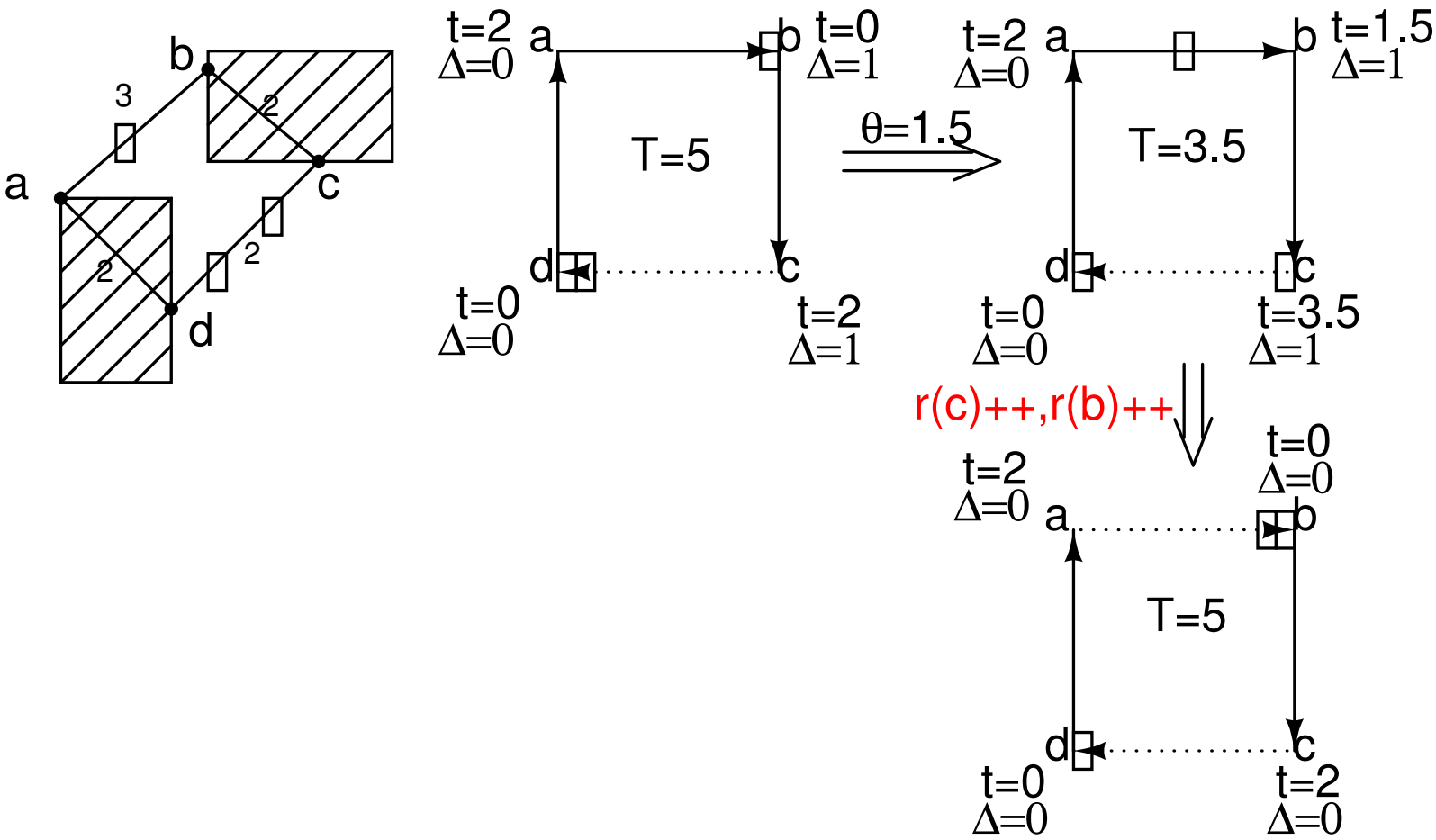
An example



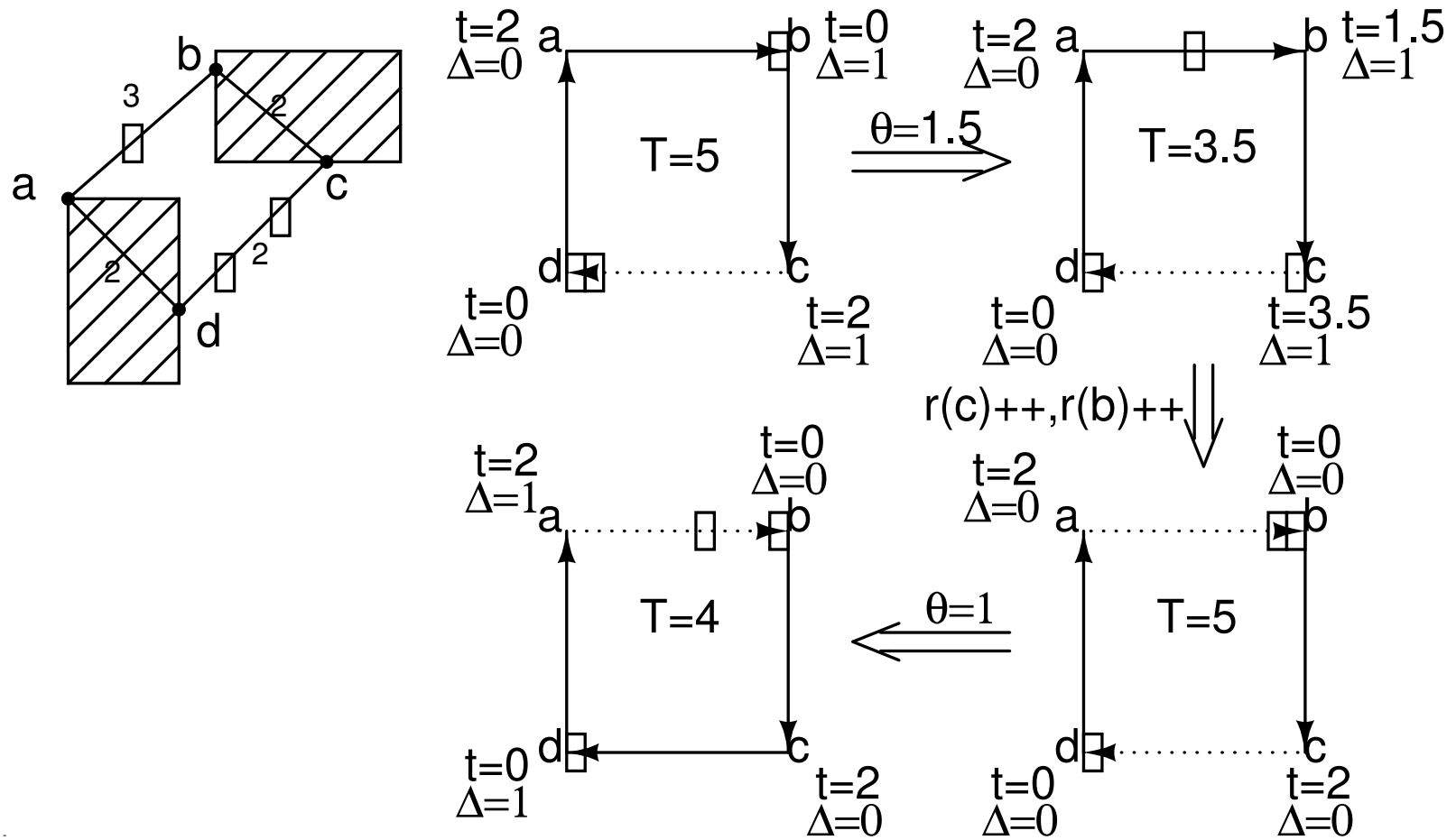
An example


a
b
c
d
3
2
2
2
t=2
Δ=0
t=0
Δ=1
t=0
Δ=0
t=2
Δ=1
T=5
θ=1.5
t=2
Δ=0
t=1.5
Δ=1
t=0
Δ=0
t=3.5
Δ=1
T=3.5

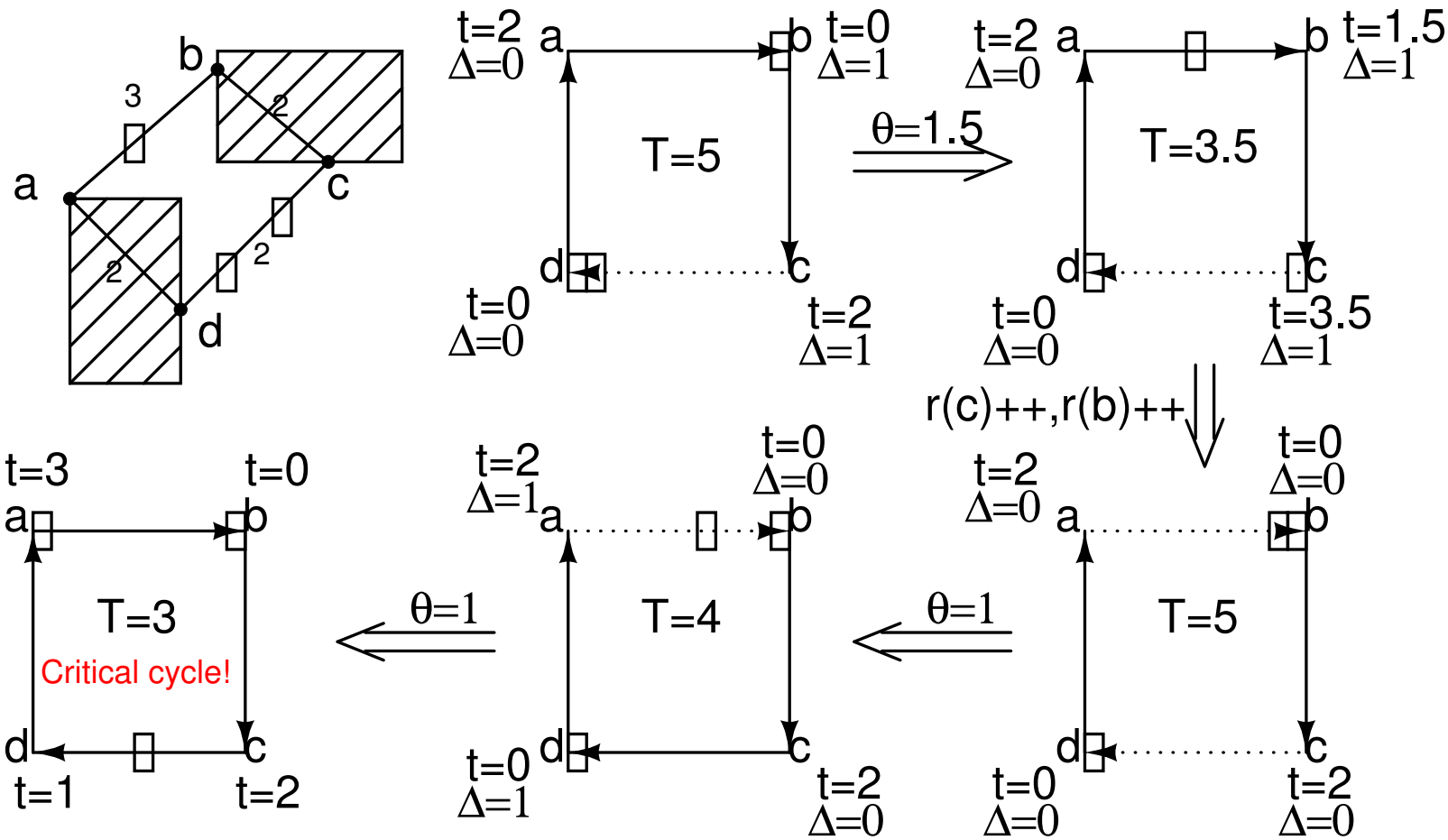
An example


a
b
c
d
3
2
2
2
t=2
Δ=0
t=0
Δ=1
T=5
t=0
Δ=0
t=2
Δ=1
θ=1.5
t=2
Δ=0
t=1.5
Δ=1
T=3.5
t=0
Δ=0
t=3.5
Δ=1
r(c)++,r(b)++
t=2
Δ=0
t=0
Δ=0
T=5
t=0
Δ=0
t=2
Δ=0

An example


a
b
c
d
3
2
2
2
t=2
Δ=0
t=0
Δ=1
t=0
Δ=0
t=2
Δ=1
T=5
θ=1.5
t=2
Δ=0
t=1.5
Δ=1
t=0
Δ=0
t=3.5
Δ=1
T=3.5
r(c)++,r(b)++
t=2
Δ=1
t=0
Δ=0
t=0
Δ=1
t=2
Δ=0
T=4
θ=1
t=2
Δ=0
t=0
Δ=0
t=0
Δ=0
t=2
Δ=0
T=5

An example


a
b
c
d
3
2
2
2
t=2
Δ=0
t=0
Δ=1
T=5
t=0
Δ=0
t=2
Δ=1
θ=1.5
t=2
Δ=0
t=1.5
Δ=1
T=3.5
t=0
Δ=0
t=3.5
Δ=1
r(c)++,r(b)++
t=3
t=0
T=3
Critical cycle!
t=1
t=2
θ=1
t=2
Δ=1
t=0
Δ=0
T=4
t=0
Δ=1
t=2
Δ=0
θ=1
t=2
Δ=0
t=0
Δ=0
T=5
t=0
Δ=0
t=2
Δ=0

Computation complexity

- Complexity per iteration
  - $O(|V|^2|E|)$

Computation complexity

- Complexity per iteration
  - $O(|V|^2|E|)$
- # of iterations
  - $O(|V| \cdot N_{\mathrm{ff}})$, where N_{ff} is the total # of FFs in any simple path

Computation complexity

- Complexity per iteration
  - $O(|V|^2|E|)$
- # of iterations
  - $O(|V| \cdot N_{\text{ff}})$, where N_{ff} is the total # of FFs in any simple path
- Entire algorithm
  - $O(|V|^3|E| \cdot N_{\text{ff}})$ in the worst case

Computation complexity

- Complexity per iteration
  - $O(|V|^2|E|)$
- # of iterations
  - $O(|V| \cdot N_{\text{ff}})$, where N_{ff} is the total # of FFs in any simple path
- Entire algorithm
  - $O(|V|^3|E| \cdot N_{\text{ff}})$ in the worst case
  - Remarkable efficiency in practice

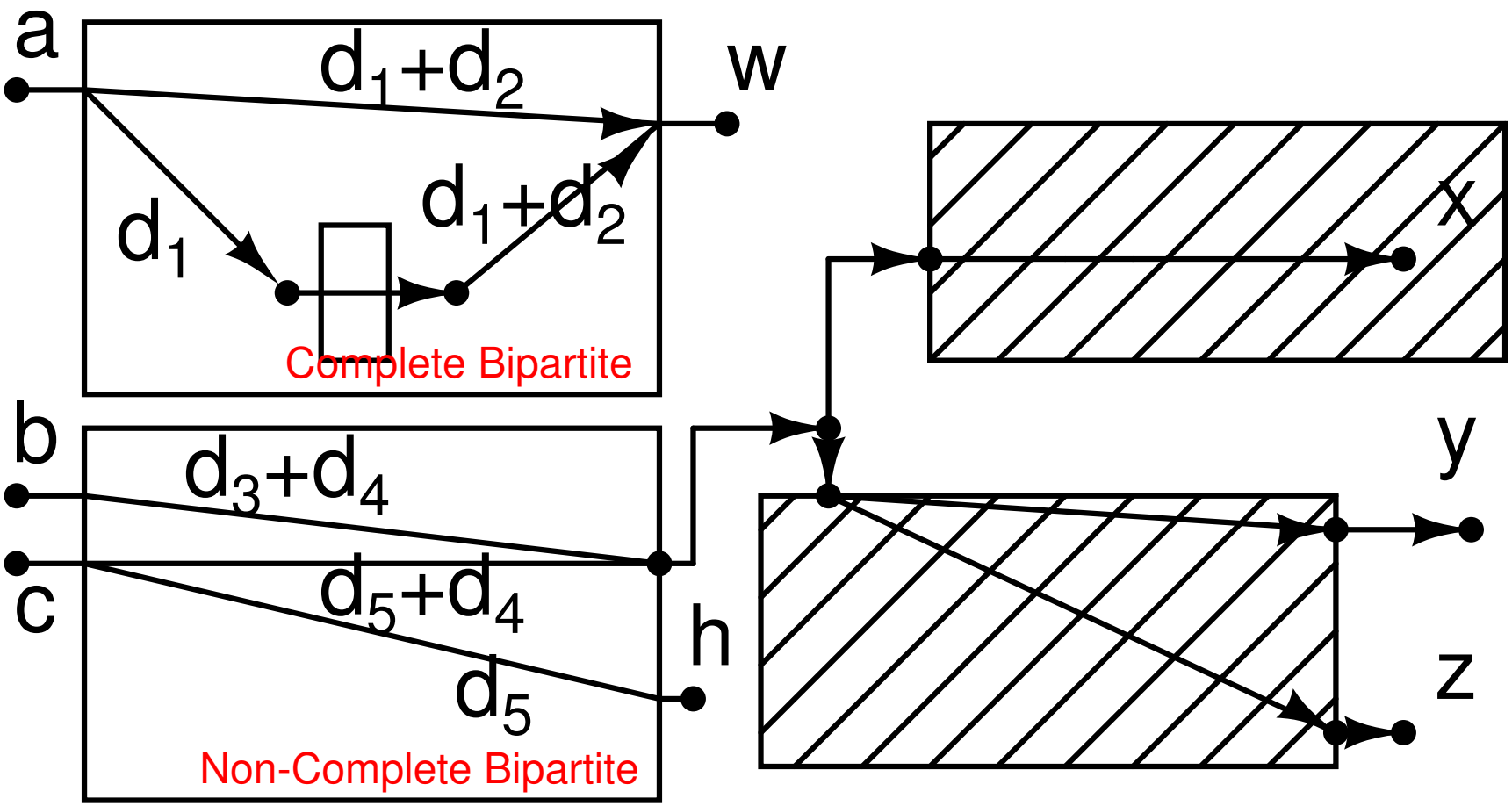
Benchmark

- ISCAS-89
  - 1st test set: treat gates as blocks
  - 2nd test set: circuits w/ non-complete bipartite blocks

Benchmark

- ISCAS-89
  - 1st test set: treat gates as blocks
  - 2nd test set: circuits w/ non-complete bipartite blocks
    - Use hMETIS to partition a circuit into groups
    - Treat each group as a block

Block models



Optimal clock period

Circuit	$\|V\|$	$\|E\|$	N_{ff}	w/o non-CB		w/ non-CB		
				#Step	T^{opt}	#Par	#Step	T^{opt}
s386	519	700	6	13	51.1	50	1	55.0
s400	511	665	21	120	32.2	50	1	50.6
s444	557	725	21	289	35.2	40	1	63.2
s838	1299	1206	32	2	76.0	130	1	84.0
s953	1183	1515	29	31	60.6	110	2	69.5
s1488	2054	2780	6	11	**70.6**	200	1	**73.3**
s1494	2054	2792	6	63	76.9	160	1	79.9
s5378	7205	8603	179	26	111.2	500	1	115.3
s13207	19816	22999	669	129	239.5	1000	1	292.8
s35932	46097	58266	1728	68	148.3	2000	1	163.2
s38584	53473	66964	1452	126	**204.0**	2000	1	**264.0**

Running time comparison (in seconds)

- t_{bs1} [ICCAD'03], t_{bs2} [DATE'04], precision=0.1

Circuit	w/o non-CB blocks			w/ non-CB blocks		
	t_{bs1}	t_{bs2}	t_{new}	t_{bs1}	t_{bs2}	t_{new}
s386	1.97	0.01	**0.00**	3.67	0.01	**0.00**
s400	1.64	0.01	**0.03**	3.38	0.01	**0.00**
s444	2.23	0.03	**0.09**	4.31	0.01	**0.00**
s838	8.79	0.03	**0.00**	33.42	0.02	**0.00**
s953	9.76	0.04	**0.02**	17.56	0.07	**0.00**
s1488	35.17	0.08	**0.08**	98.88	0.05	**0.00**
s1494	34.13	0.08	**0.06**	62.86	0.09	**0.00**
s5378	684.6	0.24	**0.31**	1344.74	0.29	**0.00**
s13207	-	1.07	**3.46**	-	206.52	**0.02**
s35932	-	18.63	**7.55**	-	6.19	**0.19**
s38584	-	7.44	**30.17**	-	21992.67	**0.19**

Summary

- Scaling trend introduces more multiple clock period interconnects
- Retiming is a critical technique for wire pipelining with correctness

Summary

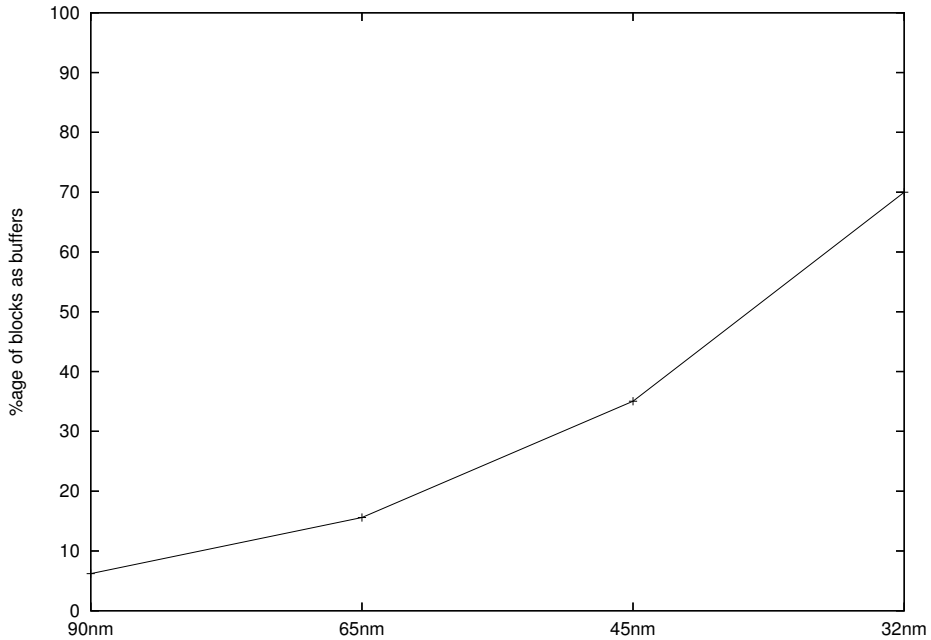
- Scaling trend introduces more multiple clock period interconnects
- Retiming is a critical technique for wire pipelining with correctness
- An efficient new algorithm is proposed and tested
  - Without binary search
  - Exact optimality
  - Polynomial time bounded
  - Simple implementation
  - Efficient in practice
  - Incremental in nature

Outline

Buffers Are Everywhere

Saxena et al. TCAD04

Projected that as many as 70% cells could just be buffers.



Efficient and Effective Techniques Are Needed

Efficient and Effective Techniques Are Needed

- Most prior researches focusing on buffering a single net:
  - van Ginneken ISCAS90
  - Shi et al. DAC03

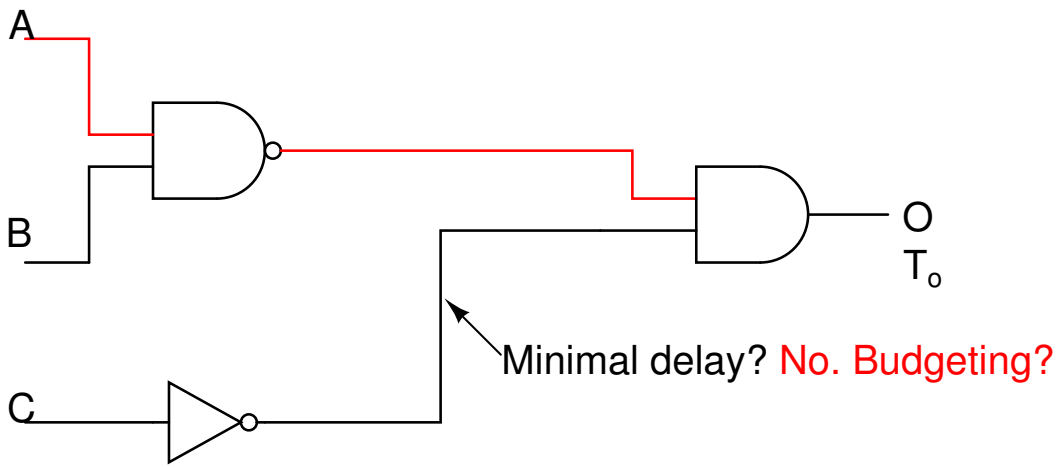
Efficient and Effective Techniques Are Needed

- Most prior researches focusing on buffering a single net:
  - van Ginneken ISCAS90
  - Shi et al. DAC03
- However, how to buffer a whole circuit is the ultimate issue

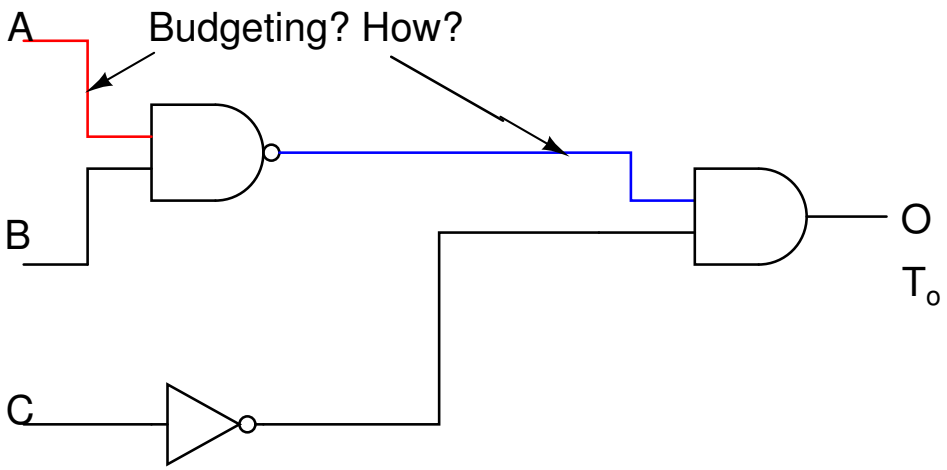
Efficient and Effective Techniques Are Needed

- Most prior researches focusing on buffering a single net:
  - van Ginneken ISCAS90
  - Shi et al. DAC03
- However, how to buffer a whole circuit is the ultimate issue
- "Budgeting + buffering each net" won't work since we do not know budgeting cost *a priori*

Simply Buffering Each Net Optimally is Overkill



Simply Buffering Each Net Optimally is Overkill



Our Goal

Problem

Given a combinational circuit, insert minimal number of buffers such that the timing constraint is satisfied.

Our Goal

Problem

Given a combinational circuit, insert minimal number of buffers such that the timing constraint is satisfied.

- It is NP-hard [Liu et al. ICCD99]

Our Goal

Problem

Given a combinational circuit, insert minimal number of buffers such that the timing constraint is satisfied.

- It is NP-hard [Liu et al. ICCD99]
- So, we just want to solve effectively and efficiently but not optimally

Limited Existing Approaches

- Lagrangian relaxation based [Liu et al. ICCD99, DATE00]
- Path-based [Sze et al. DAC05]

Lagrangian relaxation based [Liu et al. ICCD99, DATE00]

- Objective function: $\alpha \sum_{e \in E} K_e$.

Lagrangian relaxation based [Liu et al. ICCD99, DATE00]

- Objective function: $\alpha \sum_{e \in E} K_e$.
  - Objective function after LR: $\sum_{e \in E} (\alpha K_e + \beta_e d_e)$.

Lagrangian relaxation based [Liu et al. ICCD99, DATE00]

- Objective function: $\alpha \sum_{e \in E} K_e$.
  - Objective function after LR: $\sum_{e \in E} (\alpha K_e + \beta_e d_e)$.
  - Sensitive to α

Lagrangian relaxation based [Liu et al. ICCD99, DATE00]

- Objective function: $\alpha \sum_{e \in E} K_e$.
  - Objective function after LR: $\sum_{e \in E} (\alpha K_e + \beta_e d_e)$.
  - Sensitive to α
  - How to determine α?

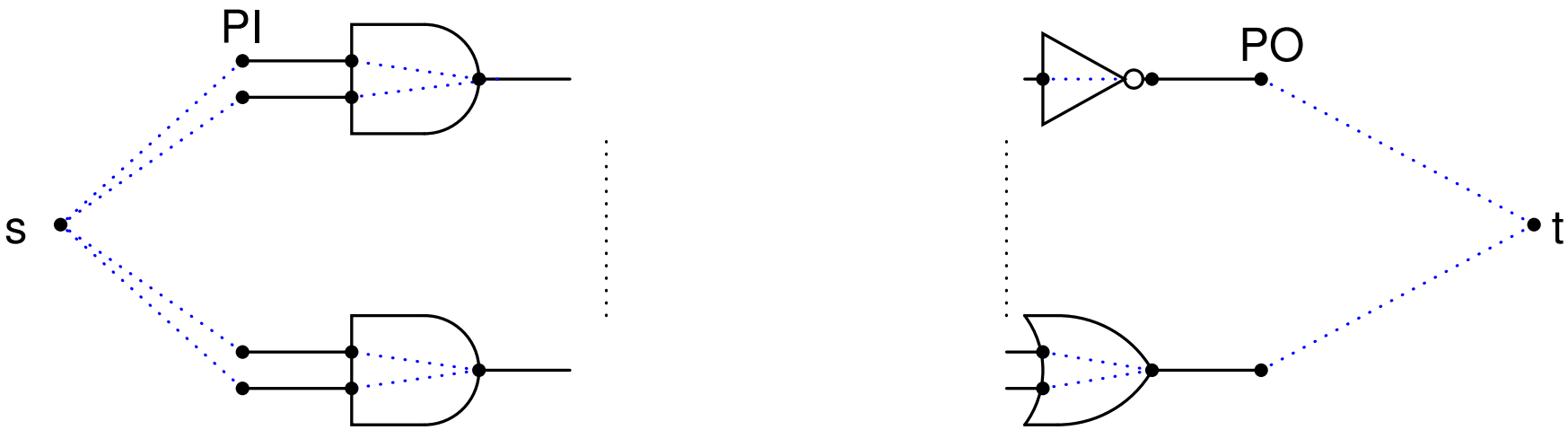
Lagrangian relaxation based [Liu et al. ICCD99, DATE00]

- Objective function: $\alpha \sum_{e \in E} K_e$.
  - Objective function after LR: $\sum_{e \in E} (\alpha K_e + \beta_e d_e)$.
  - Sensitive to α
  - How to determine α?
- Lagrangian relaxation based: expensive, over-buffering

Path based [Sze et al. DAC05]

- Select a set of critical paths
- How to determine the number of critical paths?
- Performance compared with Lagrangian relaxation based?

Problem formulation



$$E = E^P \cup E^F$$

Problem formulation

$$\begin{aligned} \textit{Minimize} \quad & \sum_{(i,j)\in E^P} K_{ij} && (5)\\ \text{s.t.} \quad & a_i + d_{ij} \le a_j \qquad \forall (i,j) \in E && (6)\\ & a_t - a_s \le REQ && (7) \end{aligned}$$

where K_{ij} is the number of buffers on edge (i,j), a_i is the arrival time at vertex i, d_{ij} is the delay of edge (i,j), and REQ is the timing constraint.

Difficulty in buffering

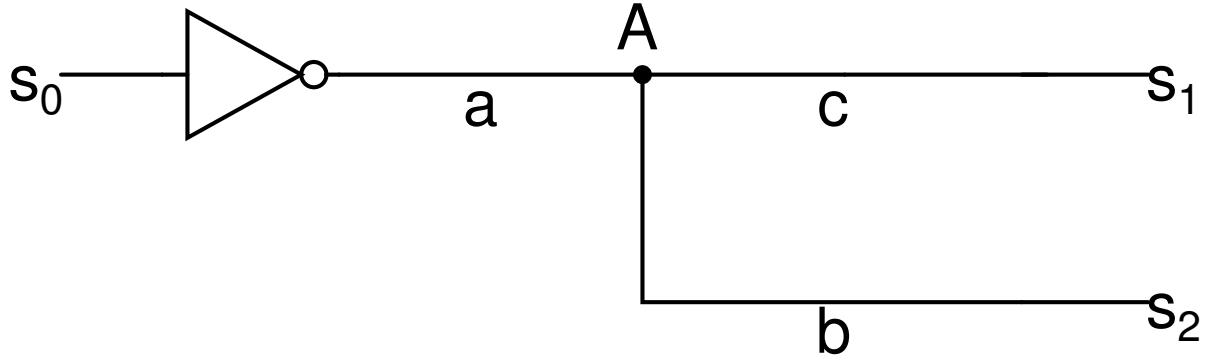
- It's a discrete optimization problem

Difficulty in buffering

- It's a discrete optimization problem
- Buffering on a branch influences delays of other branches

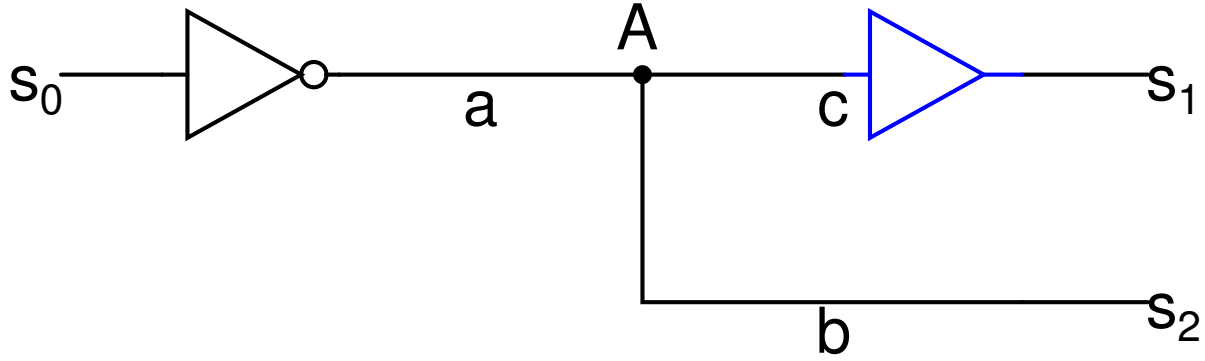
Difficulty in buffering

- It's a discrete optimization problem
- Buffering on a branch influences delays of other branches



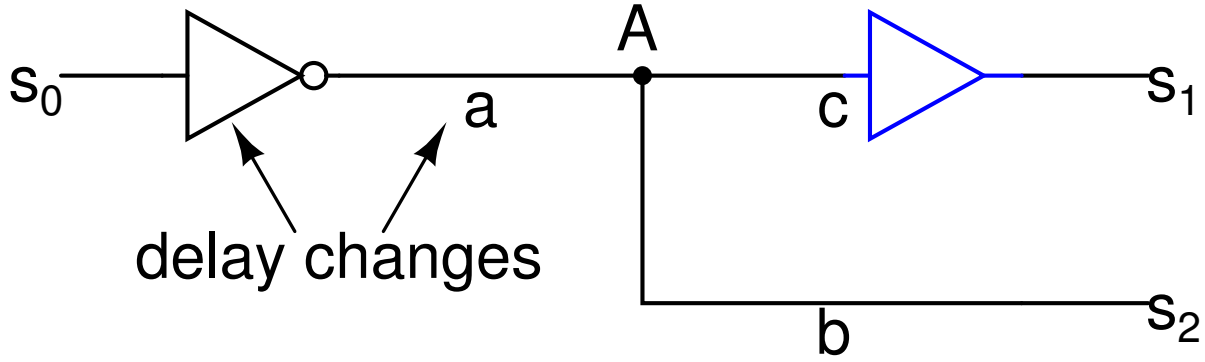
Difficulty in buffering

- It's a discrete optimization problem
- Buffering on a branch influences delays of other branches



Difficulty in buffering

- It's a discrete optimization problem
- Buffering on a branch influences delays of other branches

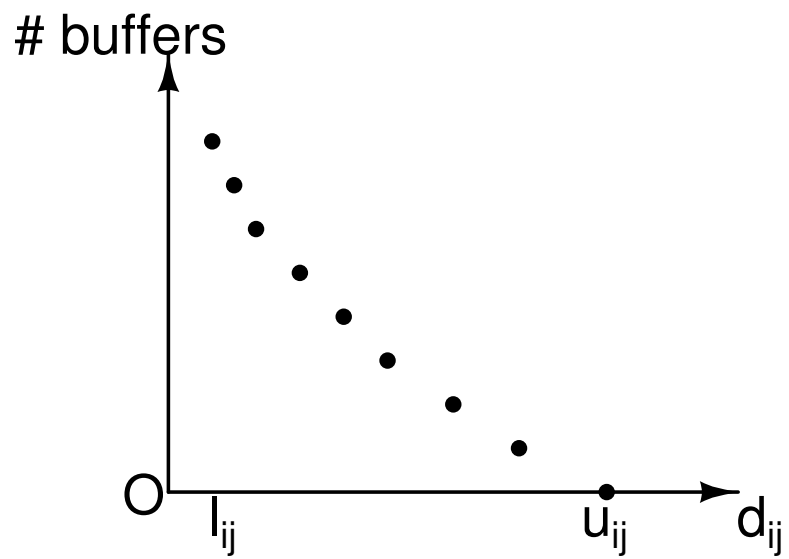


Our Approach

- Treat buffering as substitute one wire by another with different buffers
- Iterative budgeting with actual buffering cost

Our Approach

- Treat buffering as substitute one wire by another with different buffers
- Iterative budgeting with actual buffering cost

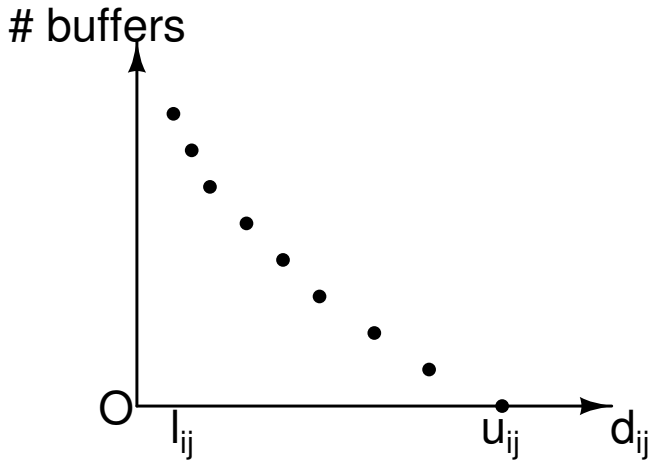


Simplified situation

- Fixed module delay
- Two-pin nets

Reformulation

- constraint graph: add one edge from t to s with weight $-REQ$
- $K_{ij} = \mathcal{C}_{ij}(d_{ij})$



Reformulation

Problem

Given a constraint graph $G(V, E)$

$$\begin{aligned} \textit{Minimize} \quad & \sum_{(i,j)\in E} C_{ij}(d_{ij}) \\ \textit{s.t.} \quad & t_j \geq t_i + d_{ij} \qquad \forall (i,j) \in E \end{aligned}$$

Reformulation

Problem

Given a constraint graph $G(V, E)$

$$\begin{aligned} \textit{Minimize} \quad & \sum_{(i,j)\in E} \mathcal{C}_{ij}(d_{ij}) \\ \textit{s.t.} \quad & t_j \geq t_i + d_{ij} \qquad \forall (i,j) \in E \end{aligned}$$

- The dual of the convex cost flow problem
  - $\mathcal{C}_{ij}$: a discrete domain with integer range
  - No existing techniques can solve this

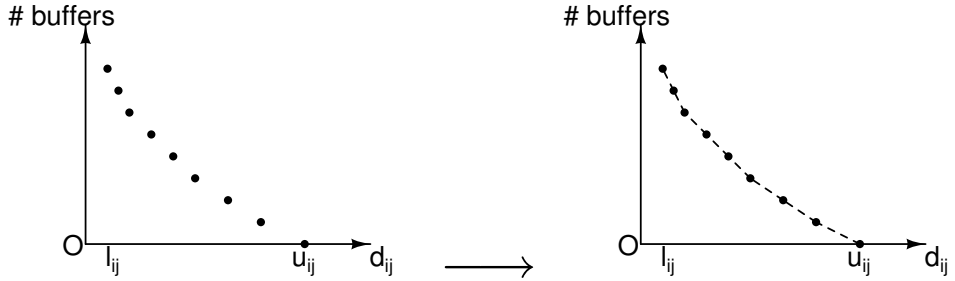
Reformulation

Problem

Given a constraint graph $G(V, E)$

$$\begin{aligned} \textit{Minimize} \quad & \sum_{(i,j)\in E} \mathcal{C}_{ij}(d_{ij}) \\ s.t. \quad & t_j \geq t_i + d_{ij} \qquad \forall (i,j) \in E \end{aligned}$$

- The dual of the convex cost flow problem
  - $\mathcal{C}_{ij}$: a discrete domain with integer range
  - No existing techniques can solve this
- Linearize $\mathcal{C}_{ij}$: convex piece-wise linear



Optimality condition

Karush-Kuhn-Tucker (KKT)

$$
\begin{aligned}
& & t_j \geq t_i + d_{ij} & \quad \forall (i,j) \in E & (8)\\
\exists \mathbf{x}: & & -\mathcal{C}_{ij}^{+}(d_{ij}) \leq x_{ij} \leq -\mathcal{C}_{ij}^{-}(d_{ij}) & \quad \forall (i,j) \in E & (9)\\
& & \textstyle\sum_{(i,j)\in E} x_{ij} - \sum_{(j,i)\in E} x_{ji} = 0 & \quad \forall i \in V & (10)\\
& & x_{ij} \geq 0 & \quad \forall (i,j) \in E & (11)\\
& & x_{ij}(t_j - t_i - d_{ij}) = 0 & \quad \forall (i,j) \in E & (12)
\end{aligned}
$$

Optimality condition

Karush-Kuhn-Tucker (KKT)

$$
\begin{aligned}
& & t_j \geq t_i + d_{ij} & \quad \forall (i,j) \in E & (8)\\
\exists \mathbf{x}: & & -\mathcal{C}_{ij}^{+}(d_{ij}) \leq x_{ij} \leq -\mathcal{C}_{ij}^{-}(d_{ij}) & \quad \forall (i,j) \in E & (9)\\
& & \textstyle\sum_{(i,j)\in E} x_{ij} - \sum_{(j,i)\in E} x_{ji} = 0 & \quad \forall i \in V & (10)\\
& & x_{ij} \geq 0 & \quad \forall (i,j) \in E & (11)\\
& & x_{ij}(t_j - t_i - d_{ij}) = 0 & \quad \forall (i,j) \in E & (12)
\end{aligned}
$$

- x_{ij}: network flow

Optimality condition

Karush-Kuhn-Tucker (KKT)

$$
\begin{aligned}
& & t_j \geq t_i + d_{ij} & \quad \forall (i,j) \in E & (8)\\
\exists \mathbf{x}: & & -\mathcal{C}_{ij}^{+}(d_{ij}) \leq x_{ij} \leq -\mathcal{C}_{ij}^{-}(d_{ij}) & \quad \forall (i,j) \in E & (9)\\
& & \textstyle\sum_{(i,j)\in E} x_{ij} - \sum_{(j,i)\in E} x_{ji} = 0 & \quad \forall i \in V & (10)\\
& & x_{ij} \geq 0 & \quad \forall (i,j) \in E & (11)\\
& & x_{ij}(t_j - t_i - d_{ij}) = 0 & \quad \forall (i,j) \in E & (12)
\end{aligned}
$$

- x_{ij}: network flow
- Condition (12): $x_{ij} \neq 0 \Rightarrow t_j = t_i + d_{ij}$.

Convex-cost flow based buffering algorithm

```
Algorithm MinCost-Buffering
d_ij ← u_ij  ∀(i,j) ∈ E
c_(i,j) ← −C_ij^−(d_ij)  ∀(i,j) ∈ E ∧ s_ij < 0
c_(i,j) ← 0  ∀(i,j) ∈ E ∧ s_ij ≥ 0
while there exist positive cycles in G
    Augment maximal flows using s as the source
        and t as the sink;
    Select a min-cut M;
    Insert one buffer into (i,j)  ∀(i,j) ∈ M;
    UpdateTimingSlack(G);
    c_(i,j) ← −C_ij^−(d_ij) − f_ij  ∀(i,j) ∈ E ∧ s_ij < 0
    c_(i,j) ← 0  ∀(i,j) ∈ E ∧ s_ij ≥ 0
```

Convex-cost flow based buffering algorithm

- No need of the explicit representation of $\mathcal{C}_{ij}$: efficiency

Convex-cost flow based buffering algorithm

- No need of the explicit representation of $\mathcal{C}_{ij}$: efficiency
- No backward flow: component delays NEVER increase, efficiency

Convex-cost flow based buffering algorithm

- No need of the explicit representation of $\mathcal{C}_{ij}$: efficiency
- No backward flow: component delays NEVER increase, efficiency
- $f_{ij} = -\mathcal{C}_{ij}^{-}(d_{ij})\ \forall (i,j) \in M$

Optimality conditions

Theorem

The solution generated by MinCost-Buffering satisfies the conditions (4)-(7).

$$
\begin{aligned}
\exists \mathbf{x}: \quad & t_j \geq t_i + d_{ij} && \forall (i,j) \in E && (4)\\
& -\mathcal{C}_{ij}^{+}(d_{ij}) \leq x_{ij} \leq -\mathcal{C}_{ij}^{-}(d_{ij}) && \forall (i,j) \in E && (5)\\
& \textstyle\sum_{(i,j)\in E} x_{ij} - \sum_{(j,i)\in E} x_{ji} = 0 && \forall i \in V && (6)\\
& x_{ij} \geq 0 && \forall (i,j) \in E && (7)\\
& x_{ij}(t_j - t_i - d_{ij}) = 0 && \forall (i,j) \in E && (8)
\end{aligned}
$$

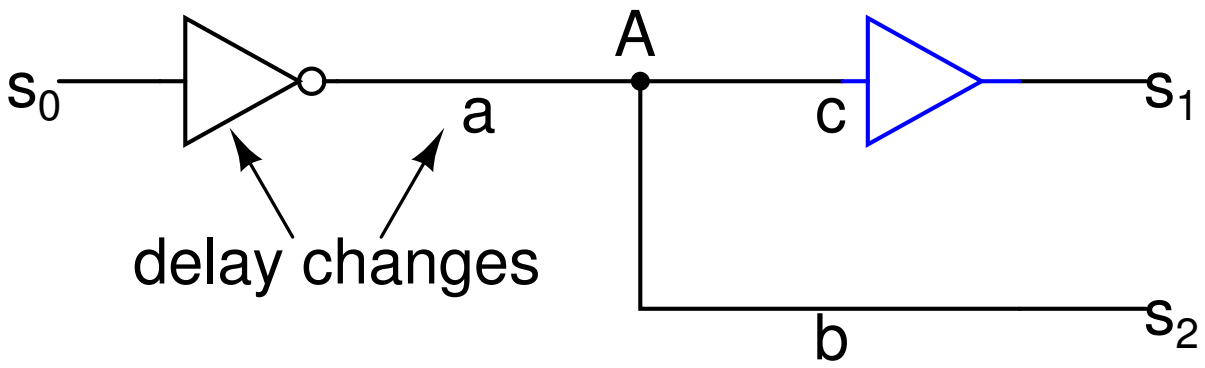
Min-cut based buffering algorithm

Algorithm MinCut-Buffering
$d_{ij} \leftarrow u_{ij} \ \forall (i,j) \in E$
$c_{(i,j)} \leftarrow -\mathcal{C}_{ij}^{-}(d_{ij}) \ \forall (i,j) \in E \wedge s_{ij} < 0$
$c_{(i,j)} \leftarrow 0 \ \forall (i,j) \in E \wedge s_{ij} \geq 0$
while there exist positive cycles in G
    Augment maximal flows using s as the source
        and t as the sink;
    Select a min-cut M;
    Insert one buffer into $(i,j) \ \forall (i,j) \in M$;
    UpdateTimingSlack(G);
    $[c_{(i,j)} \leftarrow -\mathcal{C}_{ij}^{-}(d_{ij}) - f_{ij} \ \forall (i,j) \in E \wedge s_{ij} < 0]$
    $c_{(i,j)} \leftarrow -\mathcal{C}_{ij}^{-}(d_{ij}) \ \forall (i,j) \in E \wedge s_{ij} < 0$
    $c_{(i,j)} \leftarrow 0 \ \forall (i,j) \in E \wedge s_{ij} \geq 0$

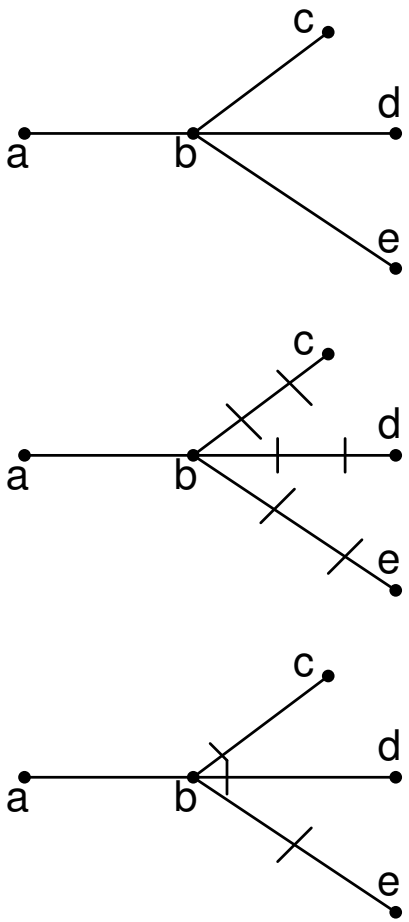
Difficulty in buffering

- Objective function is non-separable:

$$K_{ij} = f(d_1, d_2, \ldots, d_k)$$

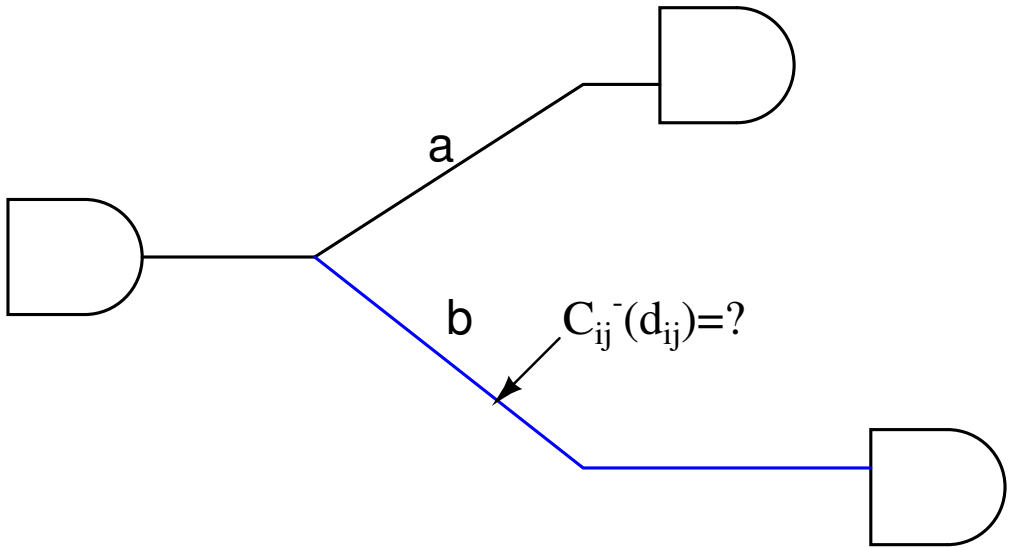


Delay sensitivity computation ($-\mathcal{C}_{ij}^{-}(d_{ij})$)

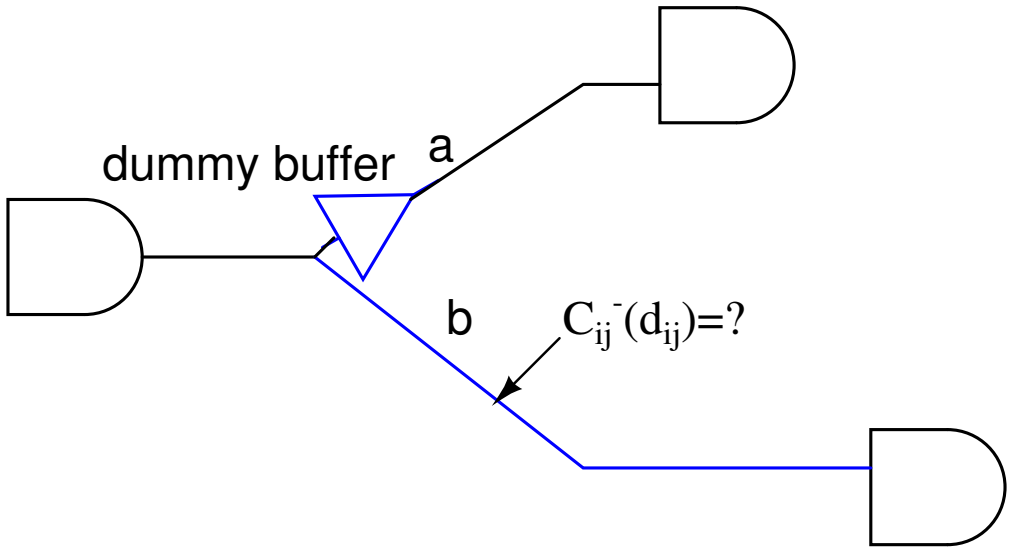


$$(a \rightarrow b \rightarrow e) >> (a \rightarrow b \rightarrow c) = (a \rightarrow b \rightarrow d)$$

Delay sensitivity computation ($-\mathcal{C}_{ij}^{-}(d_{ij})$)



Delay sensitivity computation ($-\mathcal{C}_{ij}^{-}(d_{ij})$)



Delay sensitivity computation ($-\mathcal{C}_{ij}^{-}(d_{ij})$)

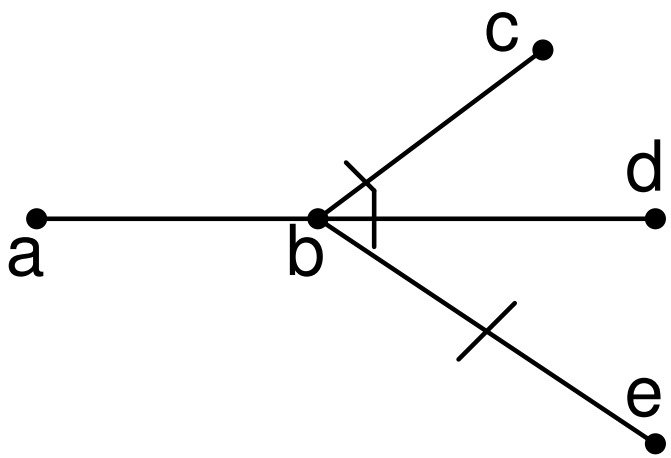
- Enforce that at most one wire of a net is in the min-cut

Delay sensitivity computation ($-\mathcal{C}_{ij}^{-}(d_{ij})$)

- Enforce that at most one wire of a net is in the min-cut
- Decouple other branches when inserting buffer on a branch

Delay sensitivity computation ($-\mathcal{C}_{ij}^{-}(d_{ij})$)

- Enforce that at most one wire of a net is in the min-cut
- Decouple other branches when inserting buffer on a branch



Delay sensitivity computation ($-\mathcal{C}_{ij}^{-}(d_{ij})$)

- δ_e: the delay change of edge e when a new buffer is inserted into e

Delay sensitivity computation ($-\mathcal{C}_{ij}^{-}(d_{ij})$)

- δ_e: the delay change of edge e when a new buffer is inserted into e

$$\delta_e = 0 \ \forall e \in E^F$$

Delay sensitivity computation ($-\mathcal{C}_{ij}^{-}(d_{ij})$)

- δ_e: the delay change of edge e when a new buffer is inserted into e

$$\delta_e = 0 \ \forall e \in E^F$$

- T_e: The maximal delay change of e and its fanin edge

$$T_e = \delta_e \ \forall e \in E^F$$

Delay sensitivity computation ($-\mathcal{C}_{ij}^{-}(d_{ij})$)

- δ_e: the delay change of edge e when a new buffer is inserted into e

  $$\delta_e = 0 \ \forall e \in E^F$$

- T_e: The maximal delay change of e and its fanin edge

  $$T_e = \delta_e \ \forall e \in E^F$$

- Delay sensitivity computation: $1/T_e$

Delay sensitivity computation ($-\mathcal{C}_{ij}^{-}(d_{ij})$)

- δ_e: the delay change of edge e when a new buffer is inserted into e

  $$\delta_e = 0 \ \forall e \in E^F$$

- T_e: The maximal delay change of e and its fanin edge

  $$T_e = \delta_e \ \forall e \in E^F$$

- Delay sensitivity computation: $1/T_e$

  $$= \infty \ \forall e \in E^F$$

Delay sensitivity computation ($-\mathcal{C}_{ij}^{-}(d_{ij})$)

- δ_e: the delay change of edge e when a new buffer is inserted into e

  $$\delta_e = 0 \ \forall e \in E^F$$

- T_e: The maximal delay change of e and its fanin edge

  $$T_e = \delta_e \ \forall e \in E^F$$

- Delay sensitivity computation: $1/T_e$

  $$= \infty \ \forall e \in E^F$$

- Buffer-forbidden area is handled transparently

Network flow based buffering algorithm framework

```
Algorithm NetworkBIN
maxdelay ← ComputeTimingAndSlack(G);
SetCapacity(G);
while maxdelay> REQ
    Find a min-cut of G;
    for each wire (u, v) in the found min-cut
        Insert one buffer into (u, v);
        Decouple the other wires that connect from u;
    maxdelay ← UpdateTimingSlack(G);
    UpdateCapacity(G);
```

CostBIN and CutBIN

- CostBIN: consider the historical flow
  - Use the timing constraint as the required time at t
  - The flow only flows through the edges with slacks less than 0
- CutBIN: no historical flow
  - Use the current maximal delay from s to t as the required time at t ($s_{ij} \geq 0$)
  - The flow only flows through the edges with slacks less than S_{th}
  - S_{th} is used to speed up the algorithm

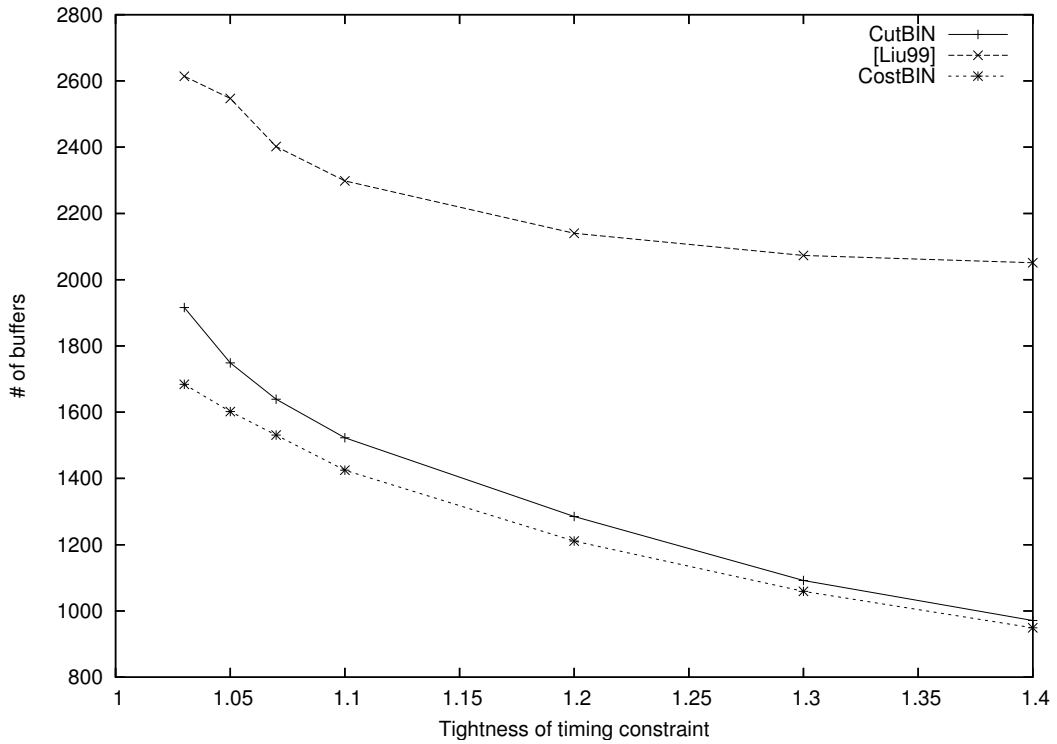
Experiment setup

- 100-nano technology
- Random graph generator used by [Liu99]
- Timing constraint: 1.2 times the minimal delay that can be achieved by buffering

Comparison results of CutBIN, CostBIN and [Liu99]

circuit		[Liu99]		CutBIN				CostBIN			
#M	#W	#B	T(s)	#B	T(s)	Reduce	Spd up	#B	T (s)	Reduce	Spd up
22	98	172	0	146	0.01	15%	1×	127	0.01	26%	1×
44	197	305	5	176	0.02	43%	250×	145	0.01	52%	500×
81	398	606	16	306	0.06	50%	267×	276	0.04	54%	400×
159	799	887	10	464	0.14	48%	71×	449	0.11	49%	91×
258	1037	1096	28	767	0.25	30%	112×	709	0.34	35%	82×
505	2039	2140	20	1251	1.04	42%	19×	1137	1.80	47%	11×
2514	10039	10297	170	5612	20	45%	9×	5206	40	49%	4×
5034	20038	21201	344	10059	58	53%	6×	9403	142	56%	2×
Avg						41%				46%	

The influence of the tightness of timing constraint



Extensions to handle multiple buffer types and fixed buffering candidate locations

circuit		[Liu00]		CostBIN			NetBIN	CutBIN	
	# Cand	Area	Time	Area	Time	Reduction	Area	Area	Time
c1	695	598.5	151	333.5	0.17	44%	629	340.5	0.17
c2	1278	659.0	90	368.0	0.37	44%	773.5	381.0	0.40
c3	2564	1955.0	256	643.5	2.58	67%	1473.0	660.0	2.98
c4	5168	2636.0	979	1089.0	5.54	59%	3096.0	1092.0	8.49
c5	5579	2933.5	859	1594.5	16.43	46%	-	1668.0	11.05
c6	11163	4842.5	1855	2604.0	32.83	46%	-	2762.0	25.00
c7	53612	24272.0	4662	11234.0	682.73	54%	-	11823.0	555.51
c8	107931	66592.0	21000	21191.5	2635.43	68%	-	22418.0	1716.93
Avg						54%			

Summary

- Effective and efficient techniques for buffering a whole circuit is needed for SoC designs
- Minimal buffering under timing constraint has a similar structure as the dual of a convex cost flow problem
- Iterative network flows are proposed for buffering a whole circuit
- Two efficient buffering algorithms based on network flow are implemented:
  - CostBIN (convex cost flow based) and CutBIN (min-cut based);
  - 46% and 41% reductions on the number of buffers.
  - 54% reductions on buffers in realistic cases.

Outline

Multicore Revolution

- Since 2004, μP frequency scaling has been flattened
- Only more cores in new generations

Multicore Revolution

- Since 2004, μP frequency scaling has been flattened
- Only more cores in new generations
- Applications will not speed up automatically

Multicore Revolution

- Since 2004, μP frequency scaling has been flattened
- Only more cores in new generations
- Applications will not speed up automatically
- Who wants to upgrade?

Multicore Revolution

- Since 2004, μP frequency scaling has been flattened
- Only more cores in new generations
- Applications will not speed up automatically
- Who wants to upgrade?
- We are all doomed if computers are like washing machines
  - No industry growth
  - No exciting projects
  - No funding

CAD Challenges

- CAD problems are huge
- CAD problems are computationally intensive
- CAD software traditionally depends heavily on frequency scaling

Salvation

Parallel programming is the only rescue!

Salvation

Parallel programming is the only rescue!

Parallel programming is very very difficult!

Salvation

Parallel programming is the only rescue!

Parallel programming is very very difficult!

- Automated parallelization is in general a failure

Salvation

Parallel programming is the only rescue!

Parallel programming is very very difficult!

- Automated parallelization is in general a failure
- Message passing based programming is too low level

Salvation

Parallel programming is the only rescue!

Parallel programming is very very difficult!

- Automated parallelization is in general a failure
- Message passing based programming is too low level
- Multithreading is hard to get right due to data racing

Thinking Parallel

To get parallelism, we have to think parallel

Thinking Parallel

To get parallelism, we have to think parallel

- With a small skull, we cannot think about true parallel

Thinking Parallel

To get parallelism, we have to think parallel

- With a small skull, we cannot think about true parallel
  - Number of possible scenario are exponential

Thinking Parallel

To get parallelism, we have to think parallel

- With a small skull, we cannot think about true parallel
  - Number of possible scenario are exponential
- Nondeterministic Transactional Model is the best possible

Nondeterministic Transactional Model in UNITY

- An algorithm is an initialization followed by a loop
- The loop is an iterative execution of any command with a true guard
- Execution is atomic (i.e. a transaction)
- Order of execution is arbitrary (nondeterministic)

Ancient Wisdom

- Euclid's GCD Algorithm ($a, b \in \mathbf{N}$)

```
Euclid's alg.  {
    x, y := a, b
    do /* GCD(x, y) = GCD(a, b) */
        x > y → x := x − y
        x < y → y := y − x
    od /* GCD(x, y) = GCD(a, b) ∧ x = y */
    output x
}
```

Min-Cost Flow Problem

- Timing-constrained optimization problems in CAD:

$$\begin{aligned} Min \quad & \sum_{(i,j)\in E} cost_{ij}(d(i,j)) \\ s.t. \quad & \forall (i,j) \in E : p(i) + d(i,j) \leq p(j) \end{aligned}$$

Min-Cost Flow Problem

- Timing-constrained optimization problems in CAD:

$$\begin{aligned} Min \quad & \sum_{(i,j)\in E} cost_{ij}(d(i,j)) \\ s.t. \quad & \forall (i,j) \in E : p(i) + d(i,j) \leq p(j) \end{aligned}$$

- Dual: min-cost flow problem:

$$\begin{aligned} Min \quad & \sum_{(i,j)\in E} w(i,j) f(i,j) \\ s.t. \quad & \forall (i,j) \in E : 0 \leq f(i,j) \leq c(i,j) \\ & \forall j \in V : \sum_{(i,j)\in E} f(i,j) = \sum_{(j,k)\in E} f(j,k) \end{aligned}$$

Optimality Condition

- Karush-Kuhn-Tucker condition for min-cost flow

$$P0 \triangleq \forall (i,j) \in E : 0 \le f(i,j) \le c(i,j)$$

$$P1 \triangleq \forall j \in V : \sum_{(i,j)\in E} f(i,j) = \sum_{(j,k)\in E} f(j,k)$$

$$P2 \triangleq \forall (i,j) \in E(f) : w^p(i,j)(\triangleq w(i,j) - p(i) + p(j)) \ge 0$$

Optimality Condition

- Karush-Kuhn-Tucker condition for min-cost flow

$$P0 \triangleq \forall (i,j) \in E : 0 \le f(i,j) \le c(i,j)$$

$$P1 \triangleq \forall j \in V : \sum_{(i,j)\in E} f(i,j) = \sum_{(j,k)\in E} f(j,k)$$

$$P2 \triangleq \forall (i,j) \in E(f) : w^p(i,j) (\triangleq w(i,j) - p(i) + p(j)) \ge 0$$

- ϵ-optimality (optimal if $\epsilon < 1/|V|$)

$$P2(\epsilon) \triangleq \forall (i,j) \in E(f) : w^p(i,j) \ge -\epsilon$$

Parallelism by Nondeterministic Transactions

- Valid guarded commands can be execute in parallel if there is no conflict.

Parallelism by Nondeterministic Transactions

- Valid guarded commands can be execute in parallel if there is no conflict.
- Non-deterministic transactional programming for multicore algorithm design
  - Easy to reason (focus on isolated atomic commands)
  - Guaranteed correctness
  - Rich parallelism

Designing Min-Cost Flow Algorithm

- Post-condition revisited:

$$
\begin{aligned}
P0 &\triangleq \forall (i,j) \in E : 0 \le f(i,j) \le c(i,j) \\
P1 &\triangleq \forall j \in V : \sum_{(i,j)\in E} f(i,j) = \sum_{(j,k)\in E} f(j,k) \\
P2(\epsilon) &\triangleq \forall (i,j) \in E(f) : w^p(i,j) \ge -\epsilon \\
Post &\triangleq P0 \wedge P1 \wedge P2(\epsilon) \wedge \epsilon < 1/|V|
\end{aligned}
$$

Designing Min-Cost Flow Algorithm

- Post-condition revisited:

$$
\begin{aligned}
P0 &\triangleq \forall (i,j) \in E : 0 \le f(i,j) \le c(i,j) \\
P1 &\triangleq \forall j \in V : \sum_{(i,j)\in E} f(i,j) = \sum_{(j,k)\in E} f(j,k) \\
P2(\epsilon) &\triangleq \forall (i,j) \in E(f) : w^p(i,j) \ge -\epsilon \\
Post &\triangleq P0 \wedge P1 \wedge P2(\epsilon) \wedge \epsilon < 1/|V|
\end{aligned}
$$

- Design strategy: use $P0$ as invariant, and all the other conditions as loop goals.

To Satisfy Loop Goals

- For $P1$
  - Push out excess $X(j) \triangleq \sum_{(i,j)\in E} f(i,j) - \sum_{(j,k)\in E} f(j,k)$
  - Keeping $P2(\epsilon)$: push only on admissible edge with $w^p(i,j) < 0$
  - If nowhere to push, increase self price: $p(i) = p(i) + \epsilon/2$

To Satisfy Loop Goals

- For $P1$
  - Push out excess $X(j) \triangleq \sum_{(i,j)\in E} f(i,j) - \sum_{(j,k)\in E} f(j,k)$
  - Keeping $P2(\epsilon)$: push only on admissible edge with $w^p(i,j) < 0$
  - If nowhere to push, increase self price: $p(i) = p(i) + \epsilon/2$
- For $P2(\epsilon)$
  - Remove residue edge by filling its capacity: $f(i,j) = c(i,j)$

To Satisfy Loop Goals

- For $P1$
  - Push out excess $X(j) \triangleq \sum_{(i,j)\in E} f(i,j) - \sum_{(j,k)\in E} f(j,k)$
  - Keeping $P2(\epsilon)$: push only on admissible edge with $w^p(i,j) < 0$
  - If nowhere to push, increase self price: $p(i) = p(i) + \epsilon/2$
- For $P2(\epsilon)$
  - Remove residue edge by filling its capacity: $f(i,j) = c(i,j)$
- For $\epsilon < 1/|V|$
  - Half ϵ when $P1$ and $P2(\epsilon)$

Nondeterministic Transactional Algorithm

- Goldberg's algorithm

$f, p, \epsilon := 0, 0, \max_{(i,j)\in E} |w(i,j)|$
do /* $P0$ */
$\exists (i,j) \in E(f) : X(i) > 0 \wedge -\epsilon \leq w^p(i,j) < 0$
$\rightarrow push(i,j)$
$\exists i \in V : X(i) > 0 \wedge \forall (i,j) \in E(f) : w^p(i,j) \geq 0$
$\rightarrow p(i) := p(i) + \epsilon/2$
$\exists (i,j) \in E(f) : w^p(i,j) < -\epsilon$
$\rightarrow f(i,j) := f(i,j) + c_f(i,j)$
$P1 \wedge P2(\epsilon) \wedge \epsilon \geq 1/|V| \rightarrow \epsilon := \epsilon/2$
od /* $P0 \wedge P1 \wedge P2(\epsilon) \wedge P3$ */

Good Features

- Correctness by construction
  - Post-condition is true when algorithm ends.

Good Features

- Correctness by construction
  - Post-condition is true when algorithm ends.
- Termination
  - No node distance decreases more than $3|V|$ times for one ϵ.

Good Features

- Correctness by construction
  - Post-condition is true when algorithm ends.
- Termination
  - No node distance decreases more than $3|V|$ times for one ϵ.
- Parallelism exposed
  - $2|E| + |V| + 1$ guarded commands, many of which are independent.

General Principle

- General ideas
  - Bind one thread to each core.
  - Thread has same life span as program.

General Principle

- General ideas
  - Bind one thread to each core.
  - Thread has same life span as program.
  - Each thread can execute every guarded command.
  - Executed command depends on available data on a core.

General Principle

- General ideas
  - Bind one thread to each core.
  - Thread has same life span as program.
  - Each thread can execute every guarded command.
  - Executed command depends on available data on a core.
  - Data (or their tokens) move among cores.

General Principle

- General ideas
  - Bind one thread to each core.
  - Thread has same life span as program.
  - Each thread can execute every guarded command.
  - Executed command depends on available data on a core.
  - Data (or their tokens) move among cores.
- Advantages
  - Long live threads to avoid overhead on creating/destroying threads
  - Thread bound to core to avoid preemption
  - Cores are keeping busy

Multicore Min-Cost Flow Program

- Same program for each core

```
while ε > 1/|V|
    if get some active nodes V_a
        for i ∈ V_a
            for (i,j) ∈ E(f)
                {if (w^p(i,j) < −ε) f(i,j) := f(i,j) + c_f(i,j)
                elseif (w^p(i,j) < 0) push(i,j)}
            end for
            if (X(i) > 0) {relabel(i)}
        end for
    elseif Sync on idle
        ε := ε/2
        activate V
end while
```

Scheduling for Each Thread

- Iteratively fetch active nodes from a global queue Q

Scheduling for Each Thread

- Iteratively fetch active nodes from a global queue Q
- Check active nodes for enabled commands

Scheduling for Each Thread

- Iteratively fetch active nodes from a global queue Q
- Check active nodes for enabled commands
- Execute each enabled command atomically

Scheduling for Each Thread

- Iteratively fetch active nodes from a global queue Q
- Check active nodes for enabled commands
- Execute each enabled command atomically
- Put new active node into the global queue

Atomicity Enforcement

- Atomic semantics of commands
  - Transactional memory: natural but immature
  - Mutual exclusion by atomic *Compare-And-Swap*

```
if(node->token.compare_and_swap(BUSY, IDLE) == IDLE) ->
the command;
```

How to Detect Termination

- No thread can terminate if there is one busy
- Take a global snapshot: Termination Detection Barrier
  - TDBarrier holds a counter implemented by atomic integer
  - Counter initialized to zero
  - Once a thread idle/active, it decrements/increments counter
  - Counter being zero means global condition achieved

Load Balancing

- Dynamically adjust length of local $b_k = q_{in}/q_{out}$

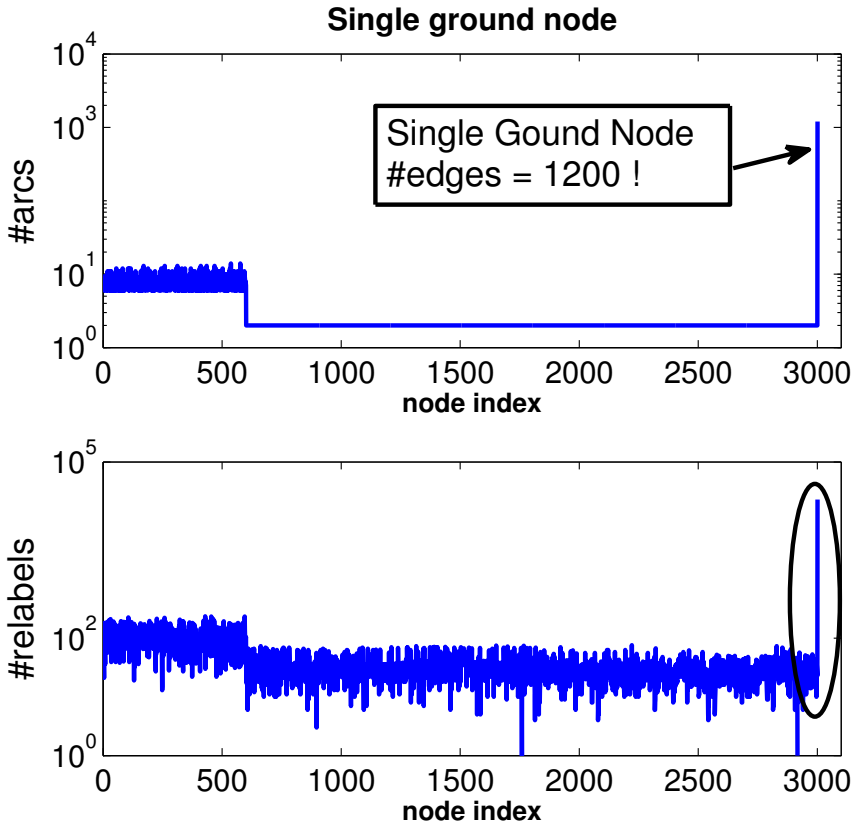
$$\begin{aligned}
&\text{if } n_{active} \leq n_{total} \times 0.75 \\
&\quad b_k = b_k/2 \\
&\text{else if } n_{active} + L/b_k \geq n_{total} \\
&\quad b_k = b_k \times 2
\end{aligned}$$

Performance Improvement

- Speed-up excellent on random networks
- Speed-up not as expected on voltage island assignment

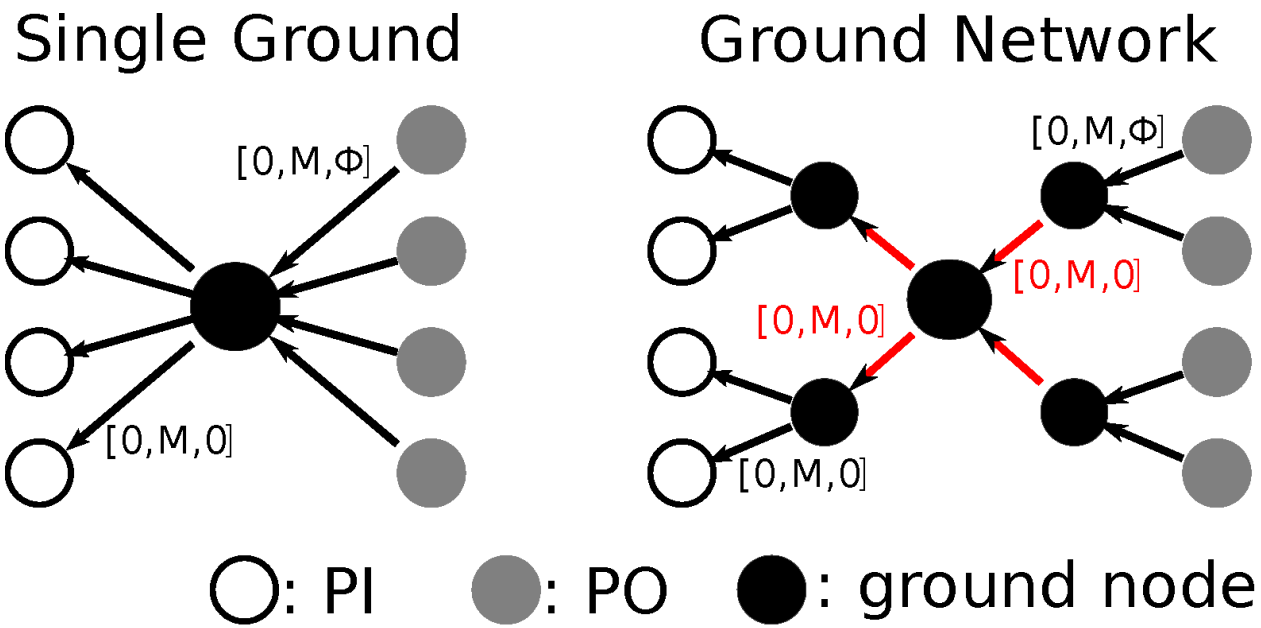
Performance Improvement

- Speed-up excellent on random networks
- Speed-up not as expected on voltage island assignment
- Caught by huge connectivity of ground node.

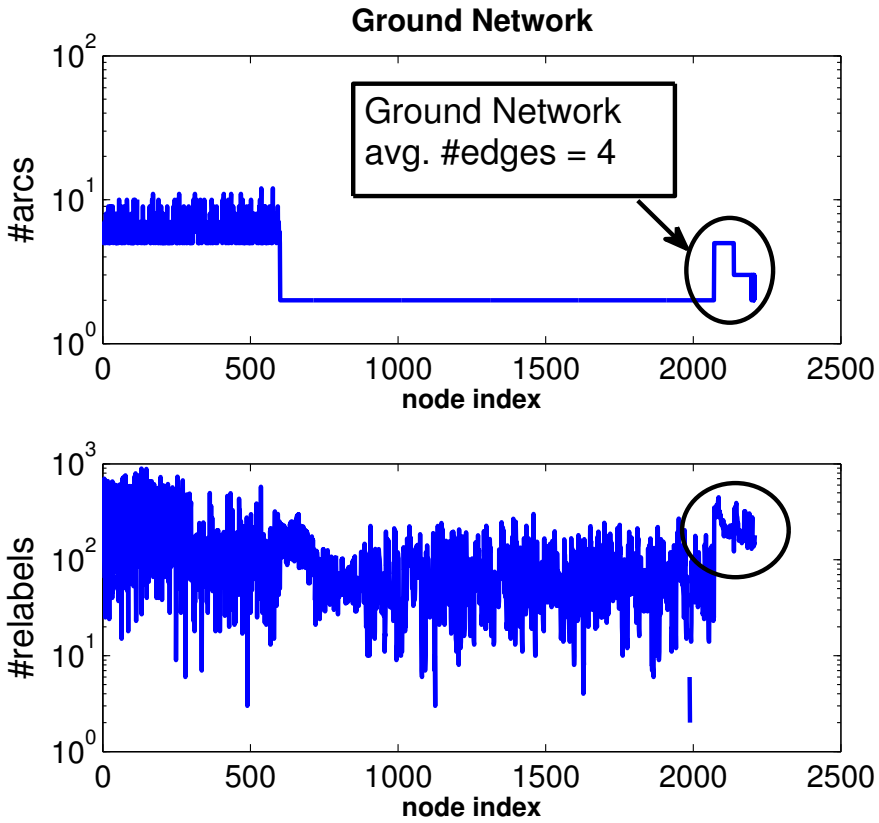


Performance Improvement

- Convert ground node to a ground network



Performance Improvement



Experiment Setup

- Implemented in multithreaded (TBB) C++.
- Compiled once and runs for different number of cores

Experiment Setup

- Implemented in multithreaded (TBB) C++.
- Compiled once and runs for different number of cores
- Application: voltage island assignment [Ma and Young ICCAD08]

$$
\begin{aligned}
\textit{Min} \quad & \sum_{(i,j)\in E} power_{ij}(v(i,j)) \\
s.t. \quad & \forall (i,j) \in E : p(i) + d_{ij}(v(i,j)) \leq p(j) \\
& \forall i \in V : 0 \leq p(i) \leq \phi \\
& \forall (i,j) \in E : v(i,j) \in \textit{Voltage}
\end{aligned}
$$

Experiment Setup

- Implemented in multithreaded (TBB) C++.
- Compiled once and runs for different number of cores
- Application: voltage island assignment [Ma and Young ICCAD08]

$$
\begin{aligned}
Min \quad & \sum_{(i,j)\in E} power_{ij}(v(i,j)) \\
s.t. \quad & \forall (i,j) \in E : p(i) + d_{ij}(v(i,j)) \leq p(j) \\
& \forall i \in V : 0 \leq p(i) \leq \phi \\
& \forall (i,j) \in E : v(i,j) \in Voltage
\end{aligned}
$$

- Linux server with two dual-core 3.0GHz CPUs and 2GB RAM, up to 4 cores.

Effectiveness of Ground Network

Cases	Single Ground		Ground Network	
	#Contentions	4C Speedup	#Contentions	4C Speedup
n10	0.00	1.25	0.00	0.93
n30	58.50	1.03	4.50	1.25
n50	196.75	1.25	5.00	1.42
n100	908.75	1.31	51.75	1.46
n200	6111.00	1.07	94.75	2.26
n300	8809.00	1.02	116.50	1.90

Speedup Rates on Voltage Island Assignment

Cases	$\|V\|/\|E\|$	Speedup Rate of 2C			Speedup Rate of 4C		
		AVG	MIN	MAX	AVG	MIN	MAX
n200	1344/2329	1.61	1.40	1.81	2.26	1.99	2.96
n300	2209/3834	1.44	1.17	1.84	1.90	1.31	2.44
n600	4414/7662	1.46	1.26	1.60	2.24	1.87	2.64
n800	5376/9322	1.73	1.52	1.99	2.78	2.32	3.31
n900	6619/11490	1.44	1.15	1.97	2.15	1.65	2.51
n1000	6720/11653	1.76	1.51	2.02	2.92	2.36	3.30
n1200	8824/15318	1.53	1.27	1.95	2.54	2.17	3.41
n1400	9410/16319	1.83	1.67	2.03	3.16	2.86	3.44
n1600	10752/18646	1.57	1.47	1.69	2.72	2.30	3.05
AVG	-	1.59	1.38	1.88	2.52	2.09	3.01

Summary

- Parallel CAD unavoidable under multicore revolution
- Parallelism better explored in Nondeterministic Transactional algorithms
- A systematic multicore implementation based nondeterministic transactional algorithm
- Min-cost flow solver with application on voltage assignment demonstrates effectiveness
- Extending to other CAD applications

Thank you

Any questions?