iRetILP : An efficient incremental algorithm for min-period retiming under general delay model

Debasish Das*, Jia Wang† and Hai Zhou
EECS, Northwestern University, Evanston, IL 60201
*Place and Route Group, Mentor Graphics, San Jose, CA 95131
†ECE, Illinois Institute of Technology, Chicago, IL 60616

January 19, 2010

Outline

Delay Models

- Simple Delay Model
  - Constant combinational delays are considered
- General Delay Model (Similar to Lalgudi et al.) considers 4 physical effects
  - Clock skews
  - Load dependent FF setup times
  - Combinational gate delays
  - Interconnect delays

Delay Models

- Simple Delay Model
  - Constant combinational delays are considered
- General Delay Model (Similar to Lalgudi et al.) considers 4 physical effects
  - Clock skews
  - Load dependent FF setup times
  - Combinational gate delays
  - Interconnect delays

Retiming

- Relocate flip-flops (FFs) w/o changing circuit functionality. [Leiserson and Saxe 83]
- Retiming under simple constant delay model
  - Polynomial time algorithm proposed by Leiserson et al.
  - An Efficient implementation proposed by Shenoy et al.
  - Algorithm considering hold conditions proposed by Papaethymiou et al.
  - Incremental algorithm proposed by Zhou et al., Lin et al.
- Retiming under general delay model
  - Branch and Bound Algorithm proposed by Soyata et al.
  - Integer linear programming based algorithm proposed by Lalgudi et al.

Retiming

- Relocate flip-flops (FFs) w/o changing circuit functionality. [Leiserson and Saxe 83]
- Retiming under simple constant delay model
  - Polynomial time algorithm proposed by Leiserson et al.
  - An Efficient implementation proposed by Shenoy et al.
  - Algorithm considering hold conditions proposed by Papaethymiou et al.
  - Incremental algorithm proposed by Zhou et al., Lin et al.
- Retiming under general delay model
  - Branch and Bound Algorithm proposed by Soyata et al.
  - Integer linear programming based algorithm proposed by Lalgudi et al.

Retiming

- Relocate flip-flops (FFs) w/o changing circuit functionality. [Leiserson and Saxe 83]
- Retiming under simple constant delay model
  - Polynomial time algorithm proposed by Leiserson et al.
  - An Efficient implementation proposed by Shenoy et al.
  - Algorithm considering hold conditions proposed by Papaethymiou et al.
  - Incremental algorithm proposed by Zhou et al., Lin et al.
- Retiming under general delay model
  - Branch and Bound Algorithm proposed by Soyata et al.
  - Integer linear programming based algorithm proposed by Lalgudi et al.

Circuit classification

General delay model classifies the circuit into two categories

- Begin/end extendible circuit

Property (Path Delay Monotonicity)

Delay of a register-to-register path decreases as the number of combinational elements on the path is decreased.

  - Efficiently solved by extension of algorithm proposed by Shenoy et al.
  - Efficiently solved by extension of incremental algorithm proposed by Zhou et al.
- Two-way extendible circuit
  - Can be solved by ILP based algorithm proposed by Lalgudi et al.
  - Inefficient in terms of performance and memory
  - Motivated us to develop iRetILP

Outline

Sequential System Optimization using Retiming

- Why General Delay Model ?
  - Considers prominent physical effects
  - More important for lower process nodes
- Min-period retiming:
  - Relocate FFs to minimize clock period.
  - Ignore cost – can increase FF area.
  - Lalgudi et al's algorithm solved min-period retiming.
- Min-area retiming:
  - Relocate FFs to minimize FF are under given clock period.
- Why study Min-period retiming ?
  - Min-area retiming needs a clock period.
  - Min-period retiming drives Min-area retiming.
- This paper focusses on Min-period retiming

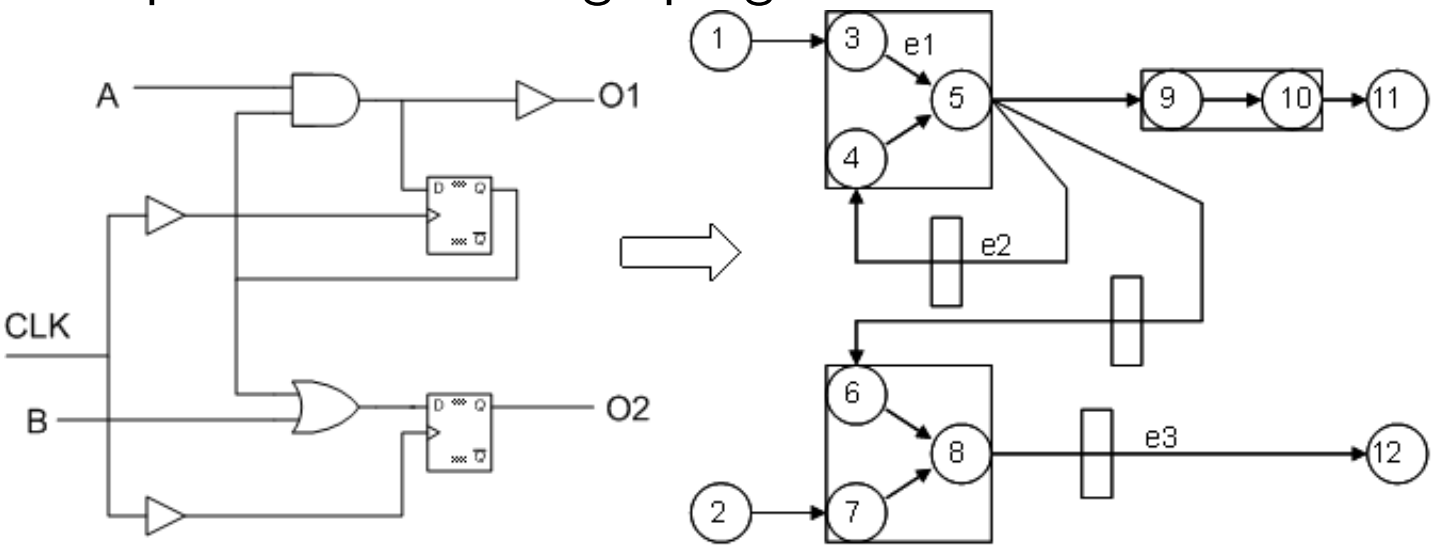
Circuit Graph Generation

Circuit graph $G = (V, E)$ of n vertices and m edges

- Each vertex $v \in V$ can be
  - primary input/output port
  - input/output port of combinational cell
- Each edge $e \in V$ can be
  - Combinational edge c with 2 labels
    - $\delta(c)$: Minimum load dependent gate delay
    - $\Delta(c)$: Maximum load dependent gate delay
  - Interconnect edge i with 4 labels
    - $d(i)$: Interconnect delay without FF
    - $\alpha(i)$: Interconnect delay driving FF input port
    - $\beta(i)$: Interconnect delay driven by FF output port
    - $w(i)$: Number of FFs on interconnect

Circuit Graph Example

Example of our circuit graph generation



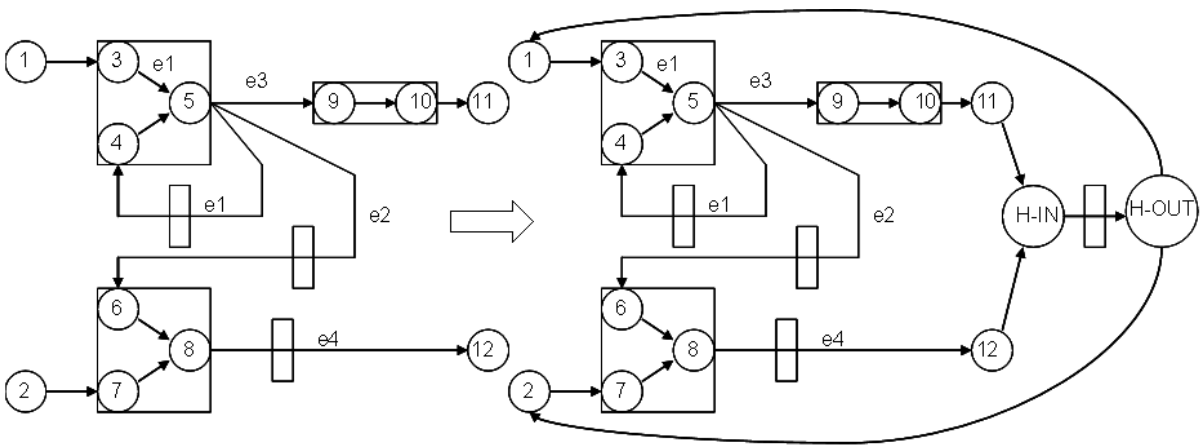
Extension of Circuit Graph

Circuit Extension for concise treatment of timing constraints

- Two vertices added : H-IN and H-OUT
- 0 delay edges added from
  - Primary outputs to H-IN
  - H-OUT to Primary inputs
  - H-IN to H-OUT
- One Virtual FF added on H-IN to H-OUT edge

Example of Extended Circuit Graph

Following figure shows the extended circuit graph

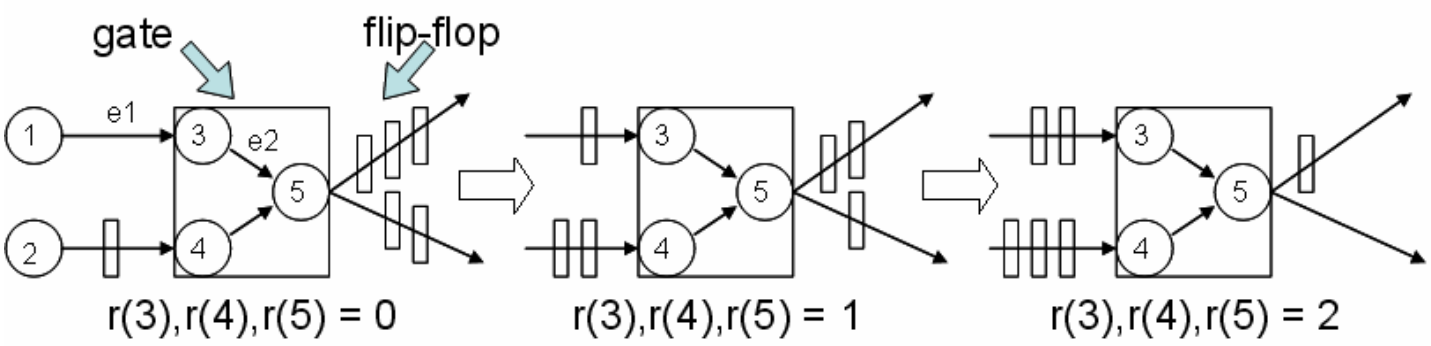


Retiming Feasibility Constraints

- Retiming is represented by an integer-valued vertex label

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

$r(v)$ is the # FFs moved from fanout interconnect edges to fanin interconnect edges of the ports of gate v



- Number of FFs on edge (i,j) is computed as

$$w_r(i,j) \triangleq w(i,j) + r(j) - r(i)$$

- Retiming feasibility constraints

$$P0(r) : w_r(i,j) \geq 0, \forall (i,j) \in E$$
$$w_r(i,j) = 0, \forall (i,j) \in C, w_r(\text{H-OUT}, \text{H-IN}) = 1$$

Timing Constraints in Retiming

- For ease of presentation we focus on setup constraints
- Use label $T : V \rightarrow \mathcal{R}^+$ to denote latest arrival time at each vertex
- Clock period of the circuit is ϕ
- Following inequalities model the timing constraints

$$P1(r, \phi) : \exists T \text{ such that:}$$
$$T(i) + \Delta(i,j) \leq T(j), \forall (i,j) \in C$$
$$T(i) + d(i,j) \leq T(j), \forall (i,j) \in I \wedge w_r(i,j) = 0$$
$$T(i) + \alpha(i,j) \leq \phi \wedge T(j) \leq \beta(i,j), \forall (i,j) \in I \wedge w_r(i,j) \geq 1$$

Theorem

The clock period of the circuit G after retiming r is given by

$$\phi(r) = \max\{T(i) + \alpha(i,j) : \forall (i,j) \in I \wedge w_r(i,j) \geq 1\}.$$

Timing Analysis Algorithm TA

Given an assignment of w_r on each edge, TA provides 3 procedures

- get-period :
  - Computes critical path $e \rightsquigarrow e'$
  - Returns ϕ as delay of critical path
- get-head-edge : returns e'
- get-tail-edge : returns e

Theorem

Given a graph G of n vertices and m edges, the algorithm TA runs in $O(m+n)$ time.

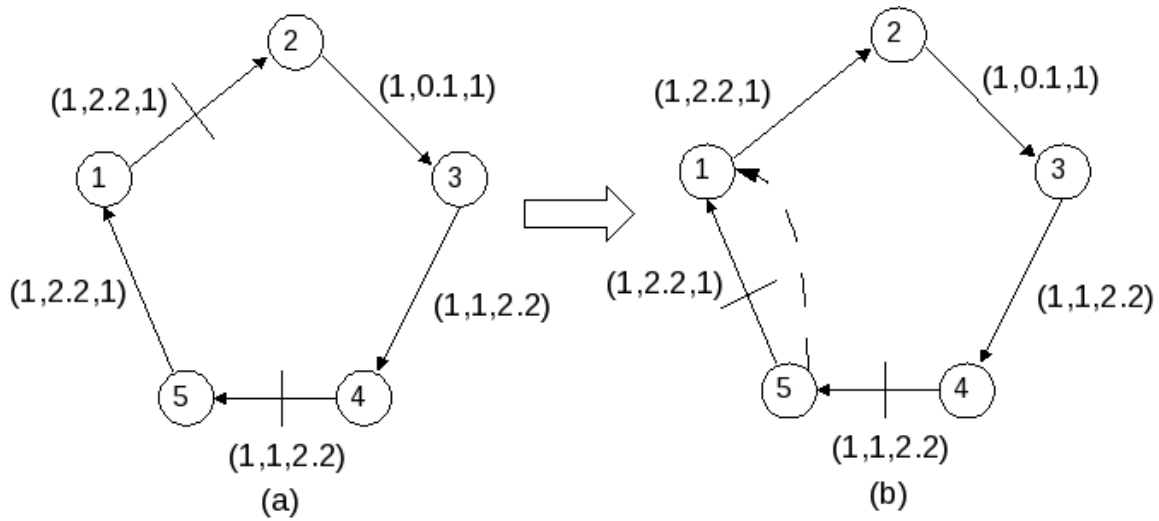
Problem Formulation

We formulate the following *Setup Retiming* problem

Problem (Generalized Setup Retiming)

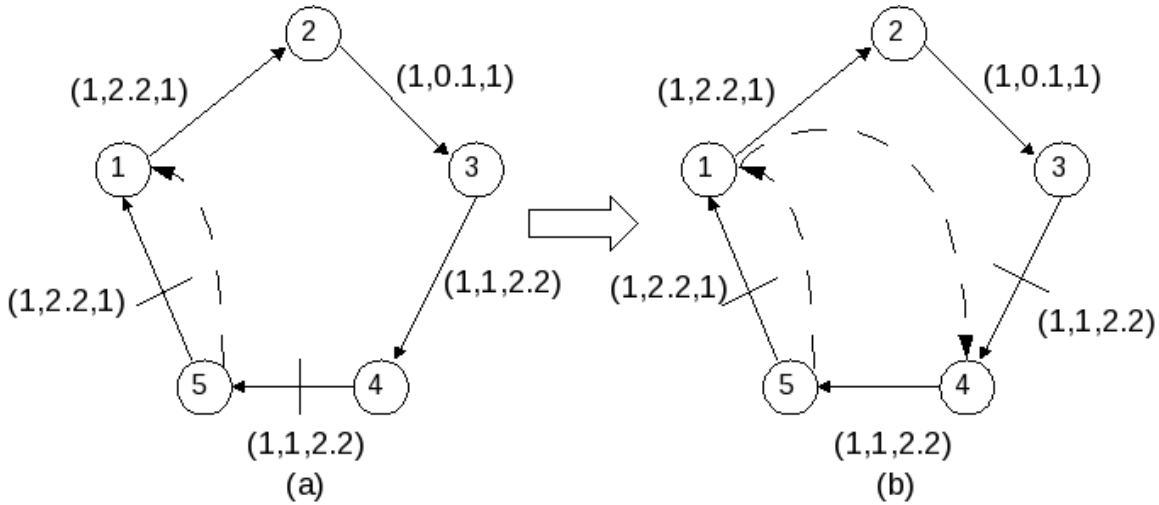
Given a circuit $G = (V, E)$ and the number of FFs w on the edges, find a retiming r such that the retiming validity constraints $P0(r)$ and timing feasibility constraints $P1(r, \phi)$ can be satisfied with the minimum clock period ϕ.

Sub-optimality of Polynomial algorithms



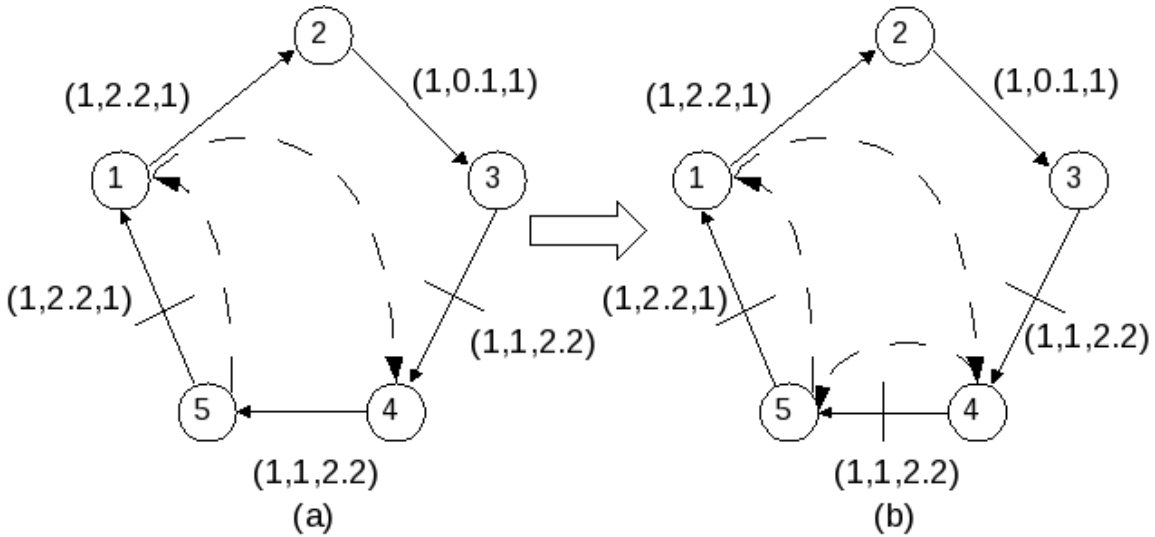
Polynomial Retiming Algorithm : Step 1

Sub-optimality of Polynomial algorithms



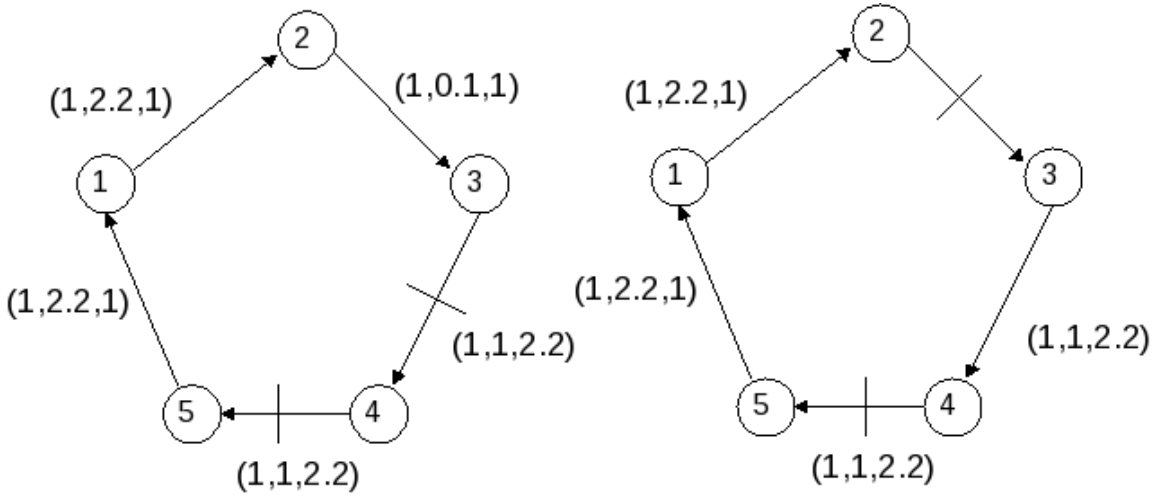
Polynomial Retiming Algorithm : Step 2

Sub-optimality of Polynomial algorithms



Polynomial Retiming Algorithm : Step 3

Sub-optimality of Polynomial algorithms



Sub-optimality of Polynomial Retiming Algorithm

Outline

Traditional Algorithm Preprocessing

We call the algorithm proposed by Lalgudi et al. as Retime-General

- Extension of Leisserson and Saxe's Algorithm.
- Two matrices : W (size n $\times$ n) and D (size m $\times$ m) used
- W[u][v] indicates minimum register count among all paths between vertices u and v

$$W(u,v) \quad = \quad min\{w(p) : u \rightsquigarrow_p v\}$$

- Given pair of edges $e = (\hat{u}, u)$, $\hat{e} = (v, \hat{v})$, D(e,$\hat{e}$) is computed as

$$D(e,\hat{e}) \quad = \quad max\{\Omega(e,p,\hat{e}) : \hat{u} \rightarrow_e u \rightsquigarrow_p v \rightarrow_{\hat{e}} \hat{v}, \\ w(p) = W(u,v)\}$$

- Ω is longest propagation delay from e to $\hat{e}$ under minimum register count constraint

Matrix D stores all possible clock periods

Clock Period Feasibility Theorem

- Matrix D and W is a preprocessing stage for Retime-General
- Given graph G, G_r is obtained using retiming transformation r
- For period c, conditions that satisfy setup constraints in G_r

Theorem (ILP Constraint)

Let G_r be a graph with retiming transformation $r : V \rightarrow \mathcal{Z}$ and c be a positive real number. c is a feasible clock period for G_r if and only if for every edge $u \rightarrow_e v \in I$, we have

$$w_r(e) \geq 0$$

and for every edge pair $e, \hat{e} \in I$ such that $\hat{u} \rightarrow_e u \rightsquigarrow v \rightarrow_{\hat{e}} \hat{v}$ and $D(e,\hat{e}) > c$, we have

$$W_r(u, v) = 0 \Rightarrow (w_r(e) = 0 \vee w_r(\hat{e}) = 0)$$

Traditional Algorithm Overview

1. Generate a clock period bound ϕ_{ub} using TA
2. Linear search to generate period $c < \phi_{ub}$
3. Feasibility of c is checked using ILP Constraint Theorem
4. Clock period updated by TA on the feasible retimed graph G_r
5. Retime-General decreases period until it becomes infeasible
6. Period larger than the infeasible period is declared optimal.

ILP Formulation

- We use a modification to $PO(r)$

$$PO^*(r) \quad \triangleq \quad \forall (i,j) \in E : 0 \leq w_r(i,j) \leq 1 \qquad (1)$$

  - Chuan et al's idea for hold constraint violation
  - Practical condition that simplifies ILP formulation
- ILP formulation for feasibility checking

Corollary

Let G_r be a graph with retiming transformation r and c be a positive real number. c is a feasible clock period if for every edge pair $e = (\hat{u}, u)$, $\hat{e} = (v, \hat{v}) \in I$ such that such that $\hat{u} \rightarrow_e u \rightsquigarrow v \rightarrow_{\hat{e}} \hat{v}$ and $D(e,\hat{e}) > c$, we have

$$w_r(e) + w_r(\hat{e}) <= 1 + W_r(u, v)$$

- 4 integer variables, Bellman-Ford algorithm can't be used

ILP Formulation

- We use a modification to $PO(r)$

$$PO^*(r) \triangleq \forall (i,j) \in E : 0 \leq w_r(i,j) \leq 1 \qquad (1)$$

  - Chuan et al's idea for hold constraint violation
  - Practical condition that simplifies ILP formulation
- ILP formulation for feasibility checking

Corollary

Let G_r be a graph with retiming transformation r and c be a positive real number. c is a feasible clock period if for every edge pair $e = (\hat{u}, u)$, $\hat{e} = (v, \hat{v}) \in I$ such that such that $\hat{u} \rightarrow_e u \rightsquigarrow v \rightarrow_{\hat{e}} \hat{v}$ and $D(e,\hat{e}) > c$, we have

$$w_r(e) + w_r(\hat{e}) <= 1 + W_r(u, v)$$

- 4 integer variables, Bellman-Ford algorithm can't be used

ILP Formulation

- We use a modification to $PO(r)$

$$PO^*(r) \triangleq \forall (i,j) \in E : 0 \leq w_r(i,j) \leq 1 \qquad (1)$$

  - Chuan et al's idea for hold constraint violation
  - Practical condition that simplifies ILP formulation
- ILP formulation for feasibility checking

Corollary

Let G_r be a graph with retiming transformation r and c be a positive real number. c is a feasible clock period if for every edge pair $e = (\hat{u}, u)$, $\hat{e} = (v, \hat{v}) \in I$ such that such that $\hat{u} \rightarrow_e u \rightsquigarrow v \rightarrow_{\hat{e}} \hat{v}$ and $D(e,\hat{e}) > c$, we have

$$w_r(e) + w_r(\hat{e}) <= 1 + W_r(u, v)$$

- 4 integer variables, Bellman-Ford algorithm can't be used

Traditional Algorithm Shortcomings

- Too many constraints in ILP formulation as D is dense
- Generating D and W matrix requires all pair shortest path algorithm
- Generating D and W matrix needs $O(n^2)$ and $O(m^2)$ memory
- Needs an artificial clock period decrease factor

Proposed Algorithm

iRetILP uses only critical constraints to generate the optimal clock period, eliminate the need to generate matrix D and W and also removes the need to employ artificial clock period decrease factor.

- *iRetILP solves a number of ILP formulations.*
- *Each formulation has few constraints.*
- *Total runtime is significantly improved.*

Outline

Initialization

- We use 6 variables in our algorithm
  - Optimal clock period ϕ^* and register vector F^* of size m
  - Intermediate clock period ϕ and register vector F of size m
  - Critical constraint vector CV initialized to $\emptyset$
  - Retiming label r for each vertex
- TA is used to generate initial clock period ϕ
- F is populated with the current register count $w(e)$

$$0 \leq w(e) \leq 1$$

- r label on each vertex initialized to 0

Iterations

- Iterations in iRetILP are used to either
  - Generate optimal period ϕ^*
  - Provide proof that ϕ is optimal
- Each iteration of iRetILP generates a retimed graph G_r
  - Identifies a unique critical constraint, populates to CV
  - If the critical period improves present clock period ϕ
    - Register assignment of G_r is updated to F^*
    - ϕ^* is updated to present critical period
- Critical constraint at each iteration generated using ILP

Iterations illustrated

Loop

1. For all e = (u,v) $\in$ E: F[e] = w[e] + r[v] - r[u]
2. Invoke TA:
   - ϕ = get-period()
   - e = get-head-edge(), $\hat{e}$ = get-tail-edge()
3. Add (e,$\hat{e}$) to CV
4. If $\phi < \phi^*$: $\phi^* = \phi$, $F^* = F$
5. Formulate and Solve ILP to generate next r assignments
6. Terminate if ILP is infeasible or there is critical cycle

ILP for Unique Constraint Generation

Consider an arbitrary iteration of iRetILP

- r be the retiming transformation associated with this iteration
- Generate an ILP where constraints from CV do not appear as clock period of G_r
  - Each entry $c \in CV$ has edges $e, \hat{e}$ associated with it
  - Retiming associated with constraint c be r_c

$$w_{r_c}(e), w_{r_c}(\hat{e}) = 1$$

  - For retiming r, critical path $e \rightsquigarrow \hat{e}$ is equivalent to

$$w_r(\hat{u}, u) = 1 \wedge w_r(v, \hat{v}) = 1 \Rightarrow W_r(u, v) = 0$$

ILP for Unique Constraint Generation (contd.)

- Lemma to be satisfied so that $e \rightsquigarrow \hat{e}$ does not exist in G_r

Lemma (G_r ILP formulation)

Formation of G_r used the following integer linear constraints for every edge $e \in I$, $\hat{e} \in C$, we have

$$0 \leq w_r(e) \leq 1, w_r(\hat{e}) = 0$$

for every 2-tuple $\{ (e, \hat{e}), w_{uv} \} \in CV$

$$w_r(e) + w_r(\hat{e}) <= w_{uv} + 1$$

- How to generate w_{uv} for each constraint $c \in CV$?
  - w_{uv} is equivalent to $W(u, v)$ + $r_c(v)$ - $r_c(u)$
  - Traditional algorithm had W matrix, we do not have
  - We have a property, when c was critical, $W_r(u, v) = 0$

$$W(u, v) = r_c(u) - r_c(v)$$

ILP for Unique Constraint Generation (contd.)

- Lemma to be satisfied so that $e \rightsquigarrow \hat{e}$ does not exist in G_r

Lemma (G_r ILP formulation)

Formation of G_r used the following integer linear constraints for every edge $e \in I$, $\hat{e} \in C$, we have

$$0 \leq w_r(e) \leq 1, w_r(\hat{e}) = 0$$

for every 2-tuple { (e, ê), w_{uv} } $\in CV$

$$w_r(e) + w_r(\hat{e}) <= w_{uv} + 1$$

- How to generate w_{uv} for each constraint $c \in CV$?
  - w_{uv} is equivalent to $W(u, v)$ + $r_c(v)$ - $r_c(u)$
  - Traditional algorithm had W matrix, we do not have
  - We have a property, when c was critical, $W_r(u, v) = 0$

$$W(u, v) = r_c(u) - r_c(v)$$

Correctness and Termination

Theorem for algorithm correctness

Theorem (Critical Constraints)

Each iteration of our algorithm finds a entry from D matrix which is one of the timing feasibility constraint for the ILP formulation used by Retime-General with c as the optimal clock period ϕ^*.

Invariants used in our iterations

Theorem (Loop Invariant)

Loop invariant of iRetILP is that the integer linear program generated by Lemma 1 is either feasible or there are no critical cycle in the graph G_r.

Complexity Analysis

- Let size of CV be k upon termination.
- Let size of CV be s_i at i'th iteration
- Let complexity of solving ILP $= \eta \cdot s_i$ (Linear function)
- iRetILP's runtime T is dominated by ILP solver's runtime

$$T = \sum_{i=0}^{k} \eta \cdot s_i$$

- T $= O(k^2)$: quadratic function of total critical constraints
- Peak memory consumption M $= O(k)$
- iRetILP generates significantly lesser constraints than Retime-General

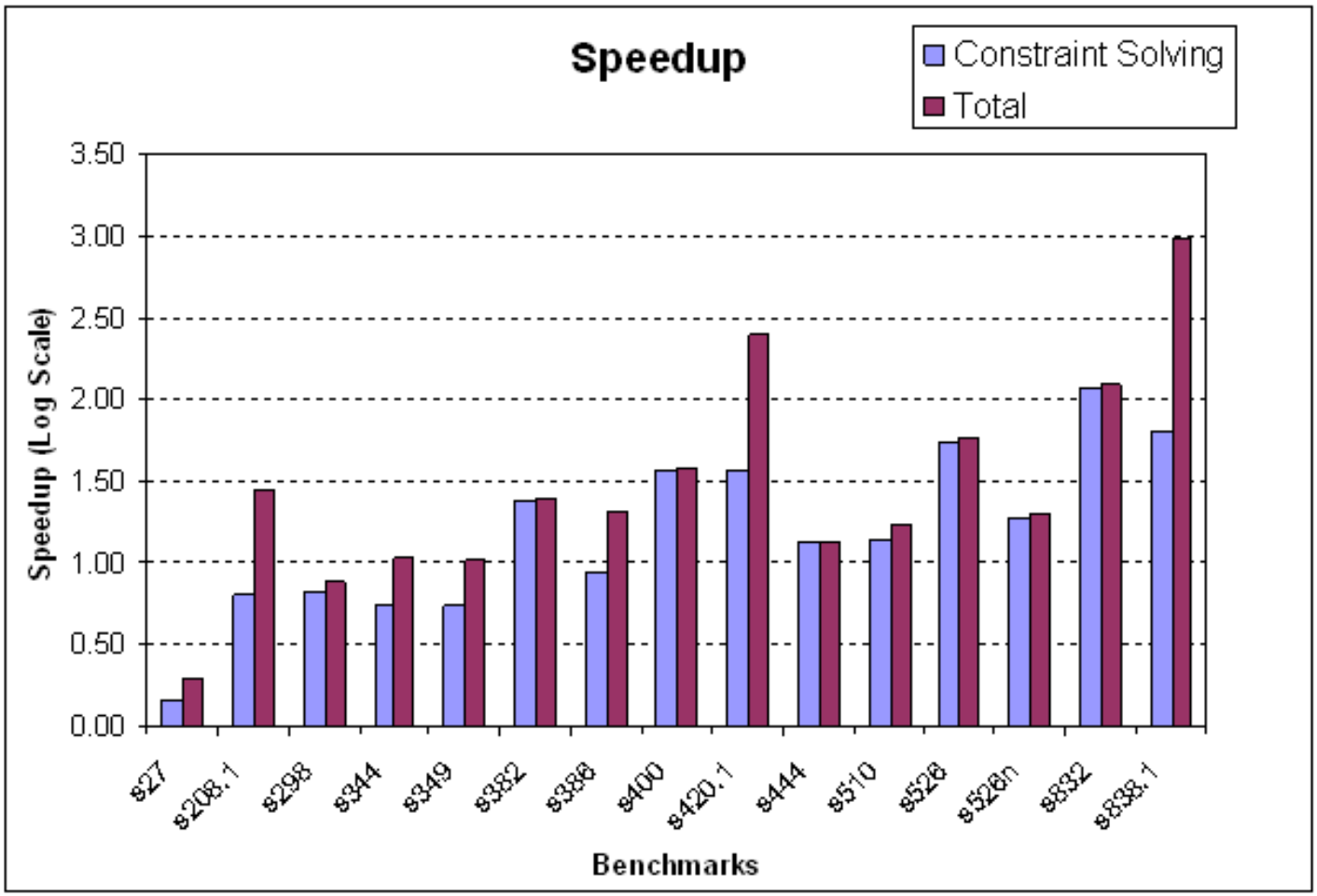
Outline

Experimental Setup

- Random delay parameter generation
- Floyd-Warshall to generate D,W matrix for Retime-General
- ILP solved using CPLEX C++ API level integration
- Divided ISCAS benchmarks into two sets, small and big
  - Small benchmarks vertex count: 1K
  - Big benchmarks vertex count: 52K
- Running Retime-General till infeasibility is slow
  - Mentioned in Lalgudi et al's work, they run 34 vertices
  - s298 (368 vertices) didn't terminate in 6 hours
  - iRetILP on s298 terminated in 91.12 secs
- iRetILP is used to generate optimal period
  - Retime-General is terminated at optimal period
  - Comparisons exclude infeasibility proof runtime

Performance Comparison : Small Benchmarks

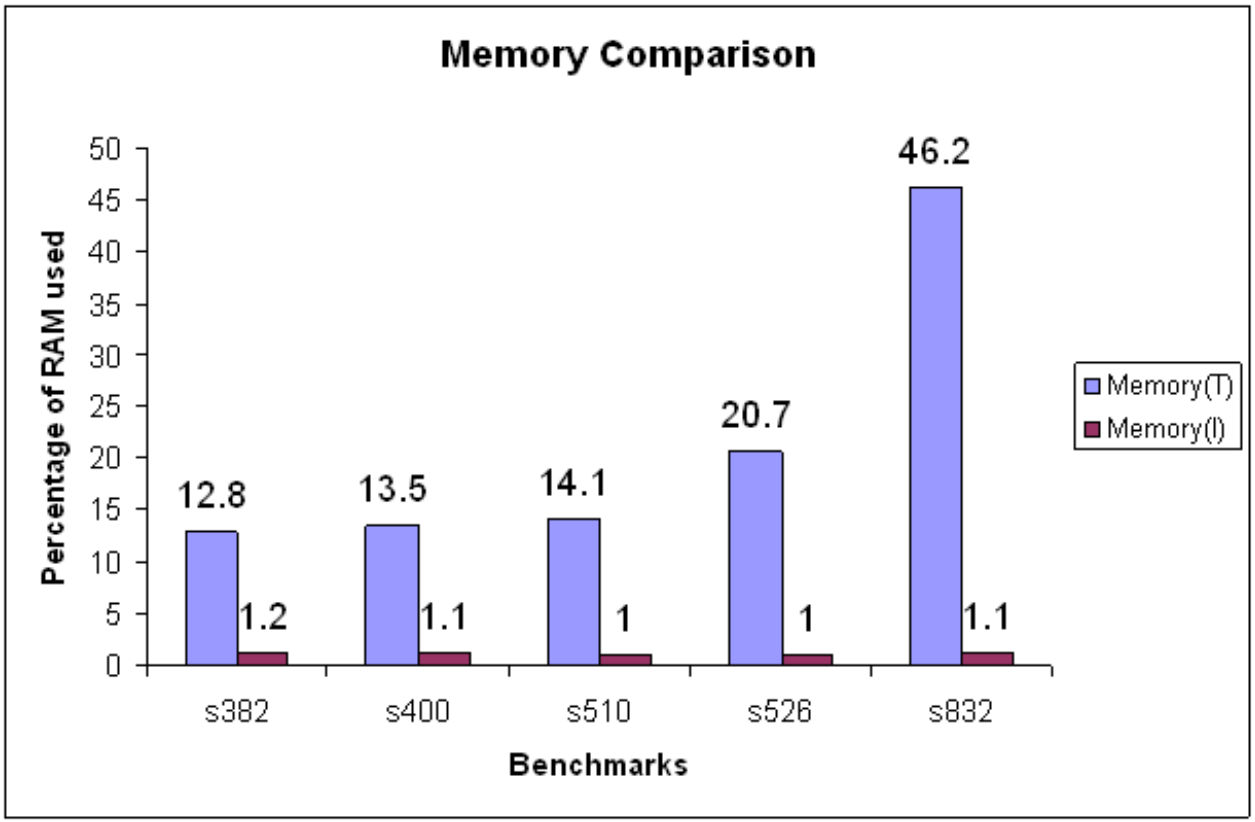
- Blue bar : Constraint solving speedup
- Red bar : Total speedup (includes Floyd-Warshall runtime)



Memory Comparison : Small Benchmarks

Peak Memory consumption of selected benchmarks

- Larger than 200 critical constraints
- Blue bar indicates Retime-General
- Red bar indicates iRetILP

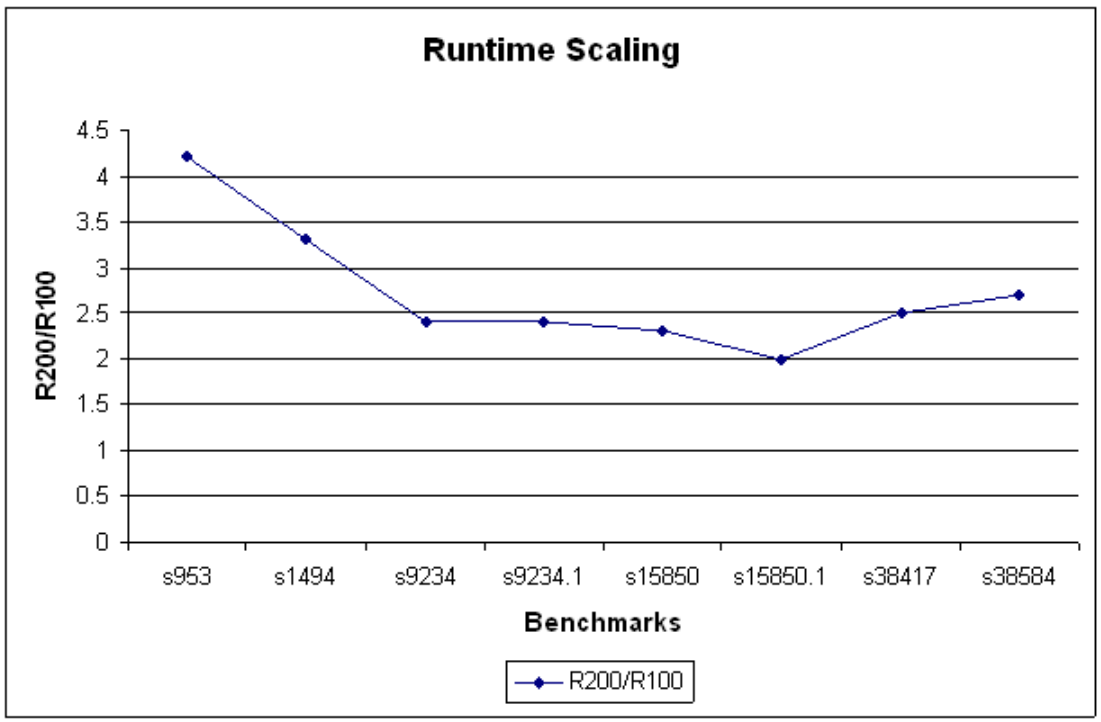


iRetILP incremental run : Large Benchmarks

Incremental mode iRetILP runs on ISCAS89 Large benchmark

- R100: Time taken for 100 iterations
- R200: Time taken for 200 iterations

Runtime scaling in incremental runs



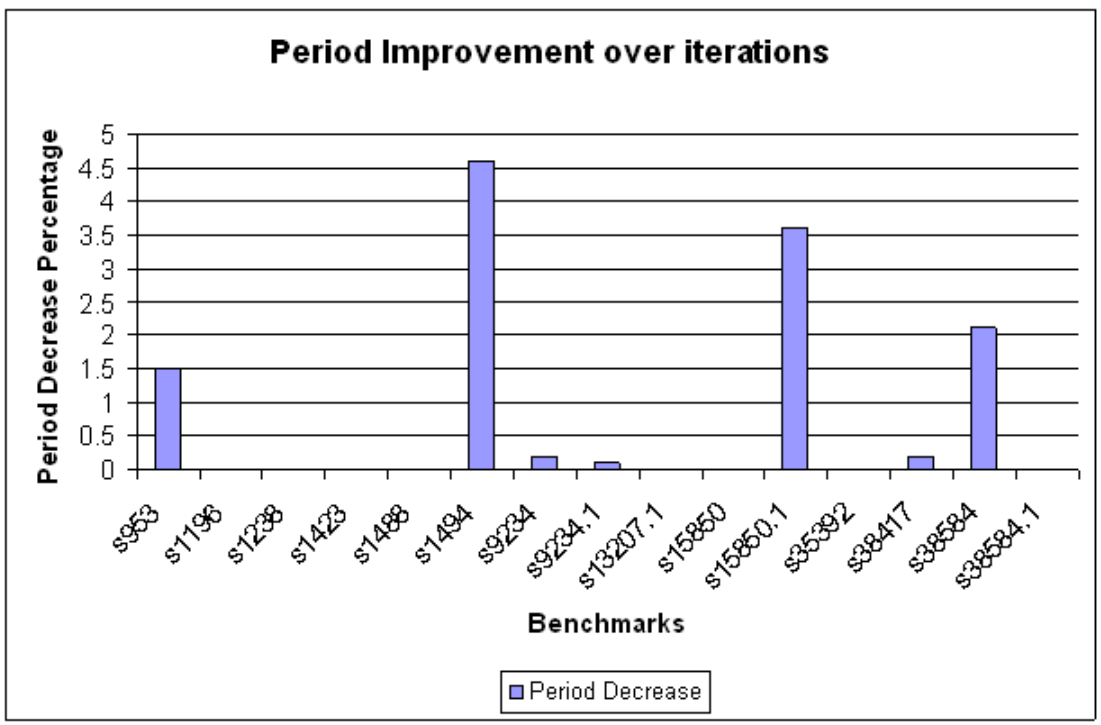
Clock period improvement

iRetILP is run in incremental mode, initial clock period ϕ_{init}

- ϕ_{100} : Optimized period after 100 iterations
- ϕ_{200} : Optimized period after 200 iterations
- Period decrease (PD) computed as

$$\text{PD} = \frac{(\phi_{200} - \phi_{100})}{\phi_{init}} \cdot 100$$

Period decrease over incremental runs



Outline

Conclusions and Future work

- Efficient incremental min-period retiming algorithm
  - On an average 100X faster than Retime-General
  - Upto 40X less peak memory consumption than Retime-General
  - Infeasibility proof should be avoided for practical usage
- For bigger benchmarks, incremental algorithm can be stopped any time, generating a feasible upper bound of optimal clock period
- Minimum area retiming for general delay models
  - Optimization version of the same problem
  - Experimenting with extended iRetILP for min-area retiming
- Open questions ?
  - Complexity class of 4 variable ILP formulation
  - Can we generate a polynomial time algorithm ?

Conclusions and Future work

- Efficient incremental min-period retiming algorithm
  - On an average 100X faster than Retime-General
  - Upto 40X less peak memory consumption than Retime-General
  - Infeasibility proof should be avoided for practical usage
- For bigger benchmarks, incremental algorithm can be stopped any time, generating a feasible upper bound of optimal clock period
- Minimum area retiming for general delay models
  - Optimization version of the same problem
  - Experimenting with extended iRetILP for min-area retiming
- Open questions ?
  - Complexity class of 4 variable ILP formulation
  - Can we generate a polynomial time algorithm ?

Conclusions and Future work

- Efficient incremental min-period retiming algorithm
  - On an average 100X faster than Retime-General
  - Upto 40X less peak memory consumption than Retime-General
  - Infeasibility proof should be avoided for practical usage
- For bigger benchmarks, incremental algorithm can be stopped any time, generating a feasible upper bound of optimal clock period
- Minimum area retiming for general delay models
  - Optimization version of the same problem
  - Experimenting with extended iRetILP for min-area retiming
- Open questions ?
  - Complexity class of 4 variable ILP formulation
  - Can we generate a polynomial time algorithm ?

Conclusions and Future work

- Efficient incremental min-period retiming algorithm
  - On an average 100X faster than Retime-General
  - Upto 40X less peak memory consumption than Retime-General
  - Infeasibility proof should be avoided for practical usage
- For bigger benchmarks, incremental algorithm can be stopped any time, generating a feasible upper bound of optimal clock period
- Minimum area retiming for general delay models
  - Optimization version of the same problem
  - Experimenting with extended iRetILP for min-area retiming
- Open questions ?
  - Complexity class of 4 variable ILP formulation
  - Can we generate a polynomial time algorithm ?

Q & A

Thank you!