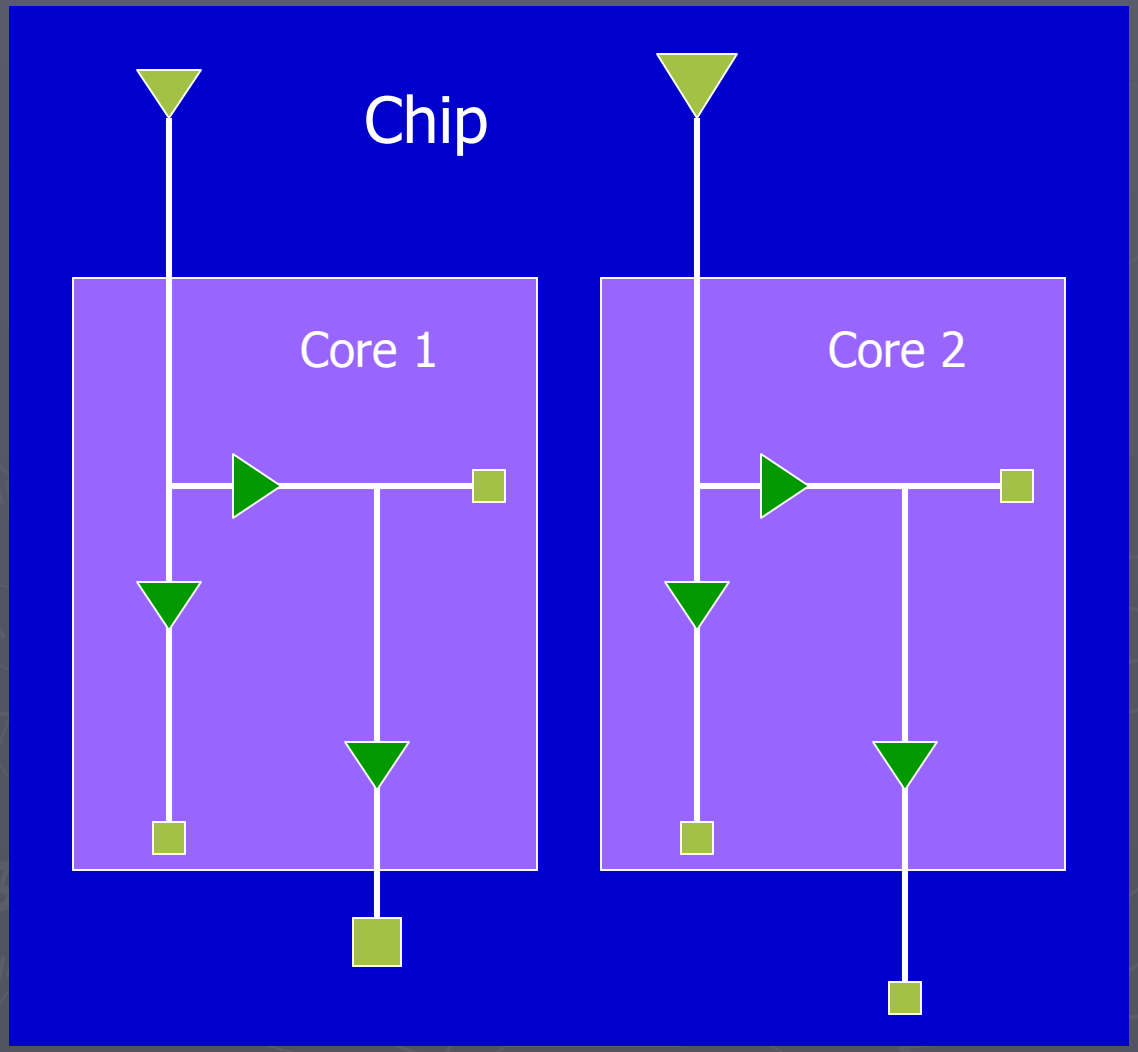
Multi-Scenario Buffer Insertion in Multi-Core Processor Designs

Yifang Liu, Jiang Hu and Weiping Shi
Department of ECE
Texas A&M University

Outline

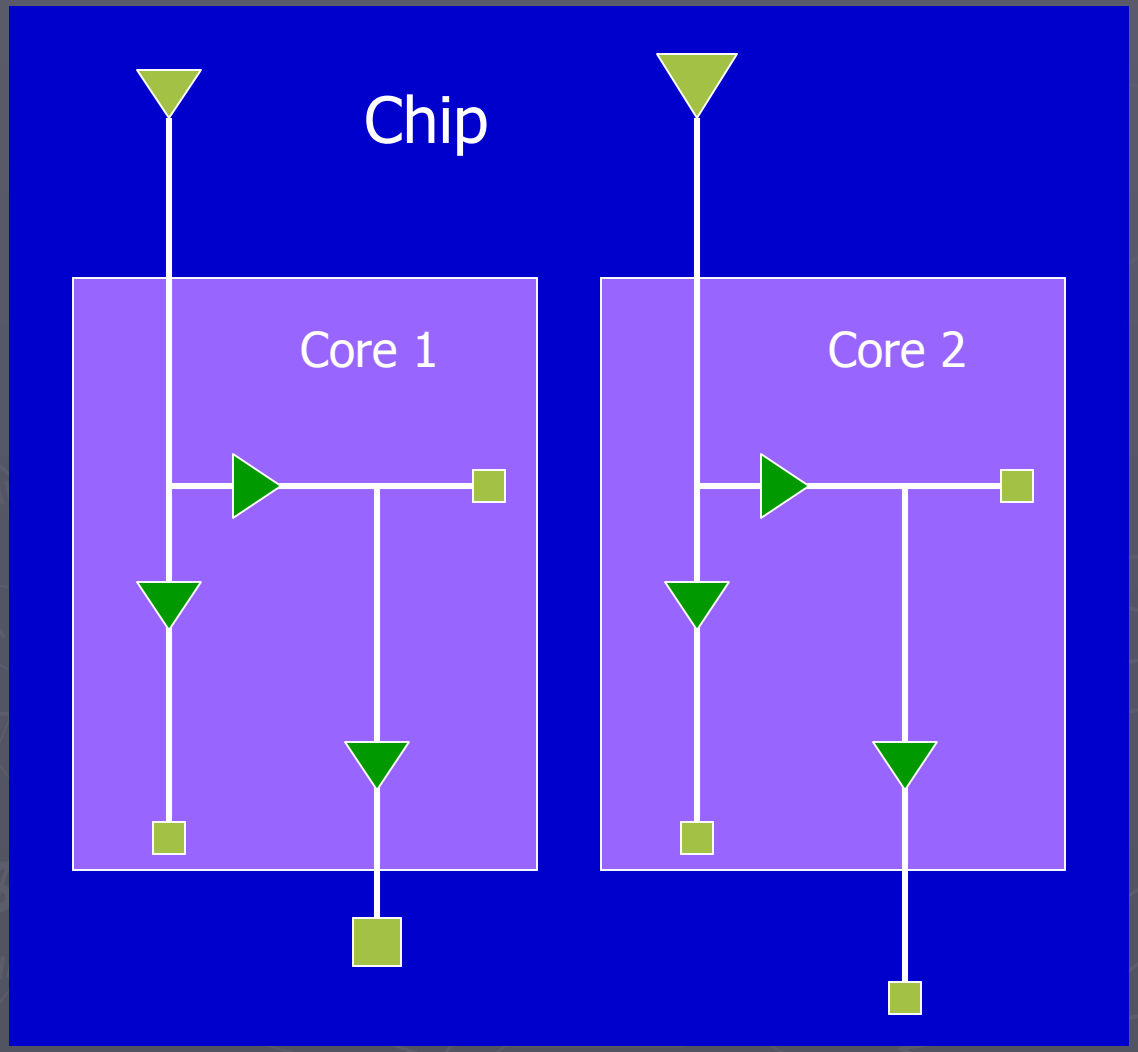
- Buffering for multi-core processors
- Algorithm
- Experimental results
- Conclusion

Buffering for Multi-core Processor



- Identical cores => identical in-core buffering solutions

Buffering for Multi-core Processor



- External context varies from core to core, differences:
  - Source strength
  - Source arrival time
  - Sink capacitance
  - Sink RAT
- A single in-core solution handles different scenarios

Definitions

- A signal net appears for multiple cores – one **instance** for each core
- **Critical slack**: the minimum slack among all instances

Problem Formulation

- Maximize critical slack
- Minimize total cost while timing constraints of all instances are satisfied
- Obtain a tradeoff curve of critical slack and total cost

Naive Approach

- Run Ginneken-like algorithm for each instance separately
- Often, these instances ends with different solutions
- Pick a solution from one instance and apply it to all instances
- A solution good for one instance may be poor for another

Our Approach

- Use **net solution** composed by identical solutions for all instances
- More precisely, only in-core part should be identical
- Propagate net solutions like Ginneken-Lillis algorithm
  - Insert the same buffer at the same node simultaneously for all instances

Net Solution Characterization

- If there are m cores, a net solution at node v_i is characterized by

$(c_{i,1}, c_{i,2}, \ldots c_{i,m}, q_{i,1}, q_{i,2}, \ldots q_{i,m}, w_i)$

c: load cap at node v_i

q: required arrival time at v_i

w: total cost for all instances

- $2m + 1$ dimension, hard to prune and slow!

Dimension Reduction

- First consider easier cases – only sinks may be different
- All instance solutions satisfy timing constraints => the worst of all instances satisfies its timing constraint
- Solution characterization

$(c_{i,1}, c_{i,2}, \ldots c_{i,m}, q_{i,min}, w_i)$

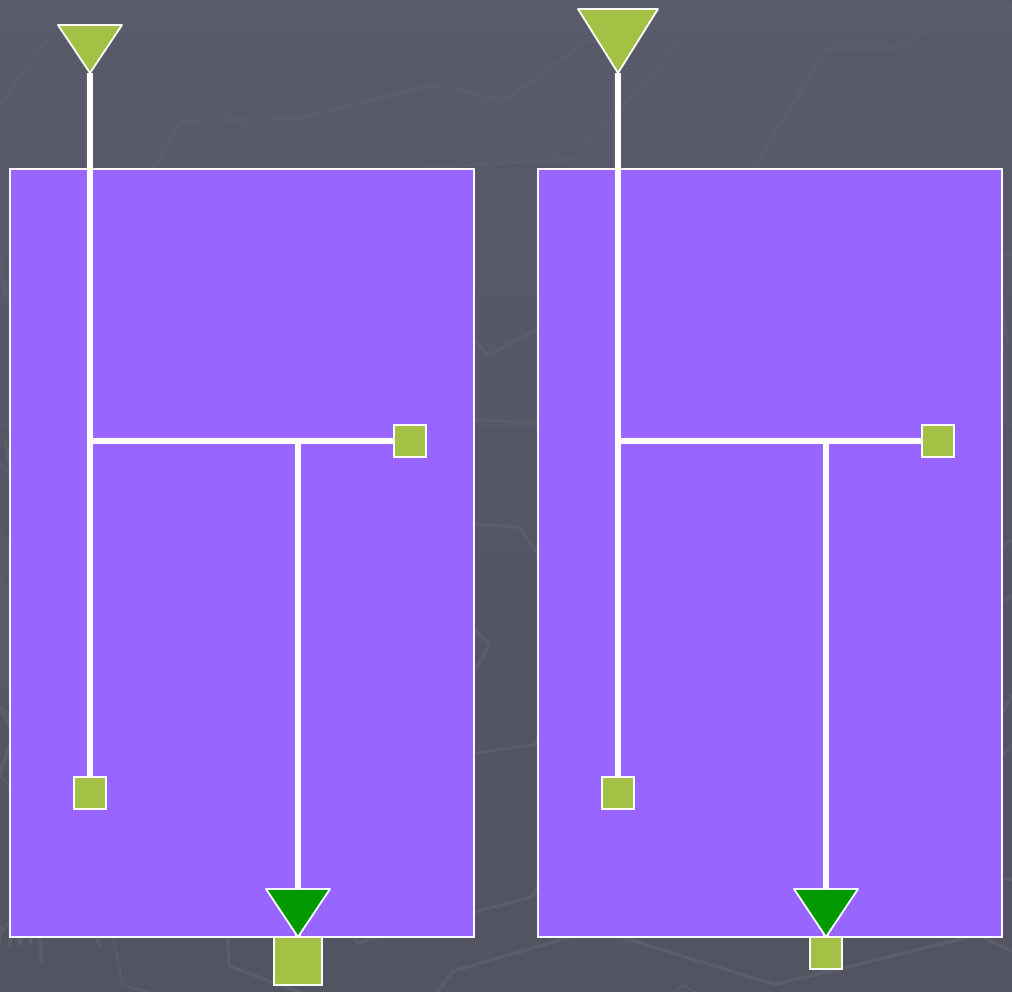
$q_{i,min} = \min (q_{i,1}, q_{i,2}, \ldots, q_{i,m})$

- $m + 2$ dimension now

Further Dimension Reduction

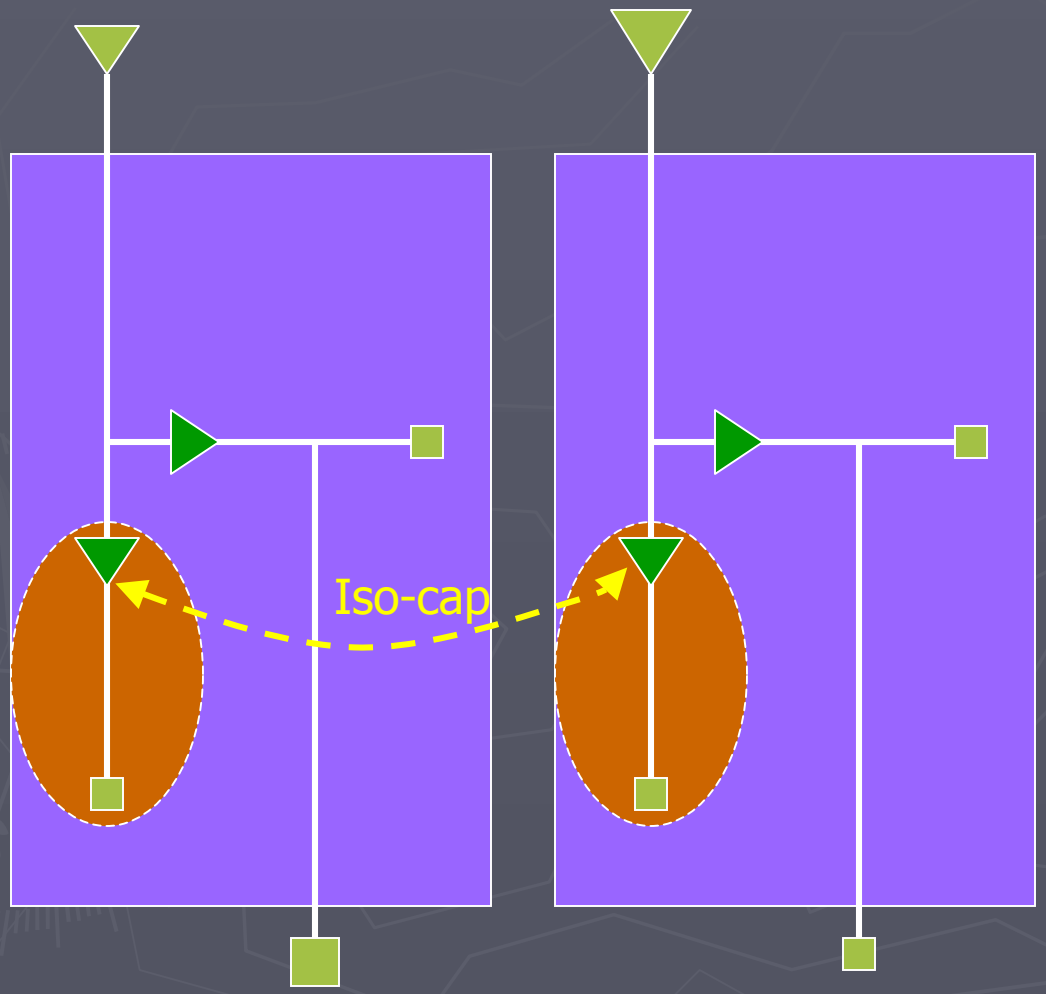
- If cap is identical for all instances – **iso-cap solution**: $(c_i, q_{i,min}, w_i)$
- Now 3D! Propagate like Lillis' algorithm

How to Make It Iso-cap?



- Quick and dirty
- Pre-insert identical buffers at boundary
- Too much buffer cost!

A Better Approach



- Once a buffer is inserted at a node, the solution at that node becomes iso-cap
- Due to slew constraint, solutions become iso-cap sooner or later

Before Iso-cap

- **Critical component** ($c_{i,max}$, $q_{i,min}$, w_i)
  $c_{i,max} = \max(c_{i,1}, c_{i,2}, \ldots, c_{i,m})$
- Solution pruning is approximated using critical component

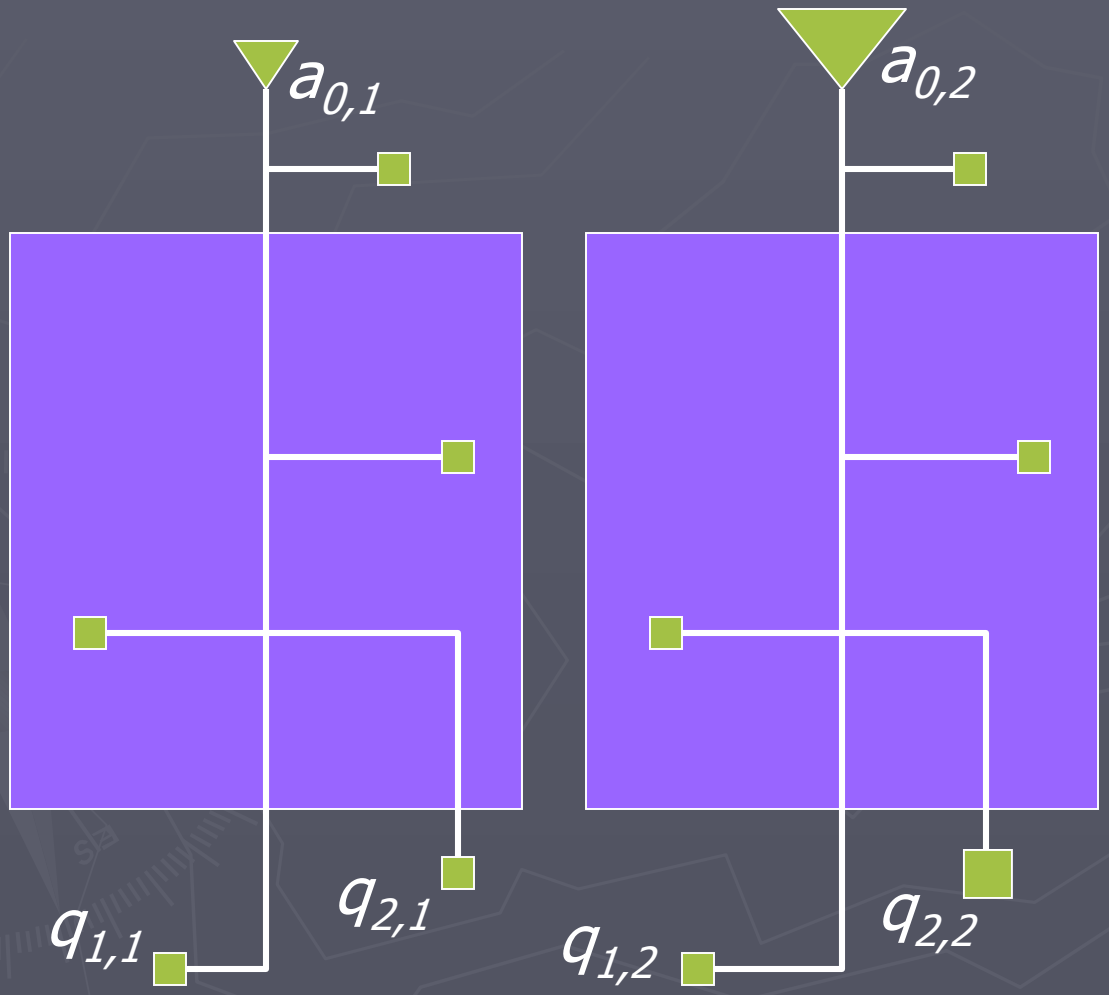
Detect Iso-cap

- Once all candidate solutions are iso-cap
  - Cap of net solution == cap of instance solution
- Prefer to propagating unbuffered solutions first
  - Iso-cap solutions can be reached earlier
- Detect iso-cap by linear time labeling
  - Iso-cap labels are propagated ahead of solutions

On the Source Side

- Discussions so far depend on assumptions
  - Same driving strength at source of each instance
  - Same arrival time (AT) at source of each instance
- How to handle differences at the source side?

Align Arrival Time at Source



a: arrival time

$q'_{1,1} <= q_{1,1} - a_{0,1}$

$q'_{2,1} <= q_{2,1} - a_{0,1}$

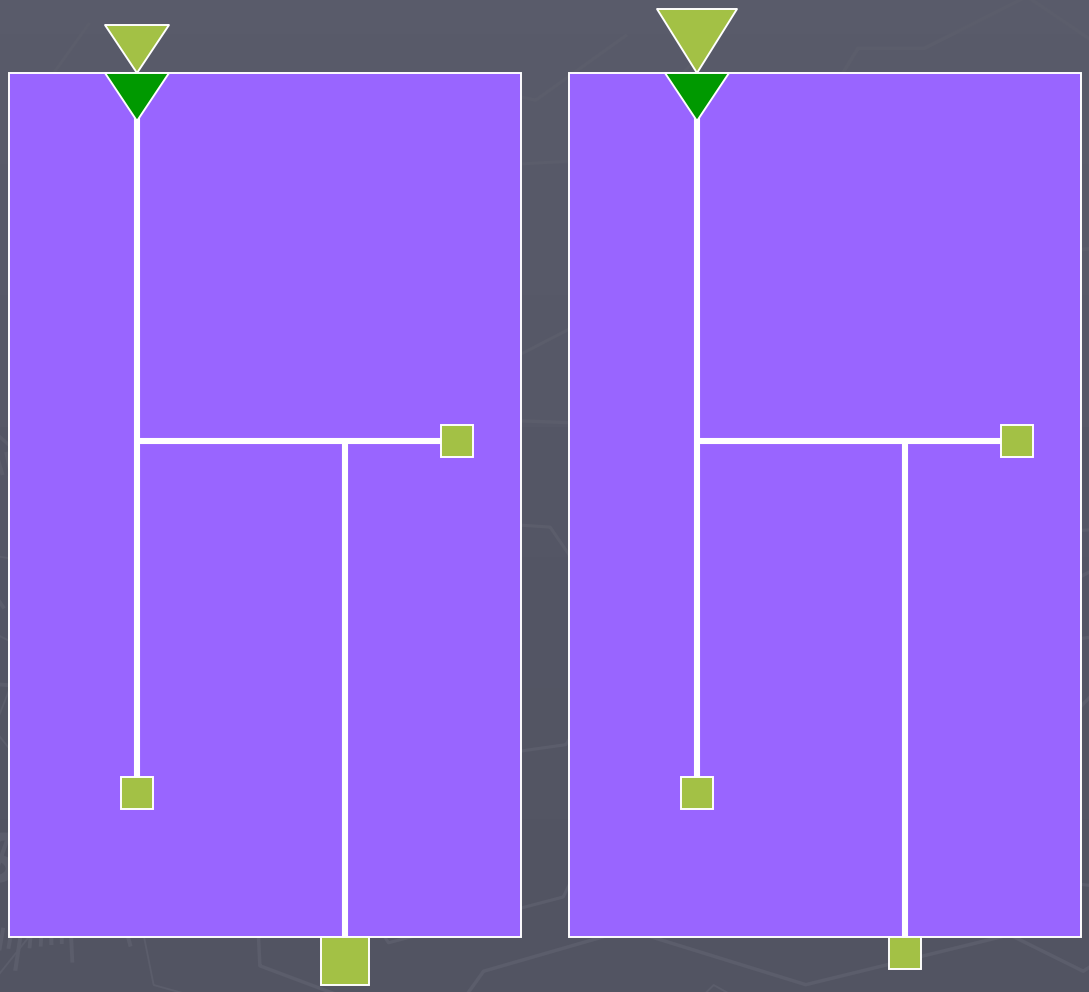
$a'_{0,1} <= 0$

$q'_{1,2} <= q_{1,2} - a_{0,2}$

$q'_{2,2} <= q_{2,2} - a_{0,2}$

$a'_{0,2} <= 0$

Handling Different Driving Strength



If differences are large, pre-fix buffers at boundary

Otherwise, ignore the difference

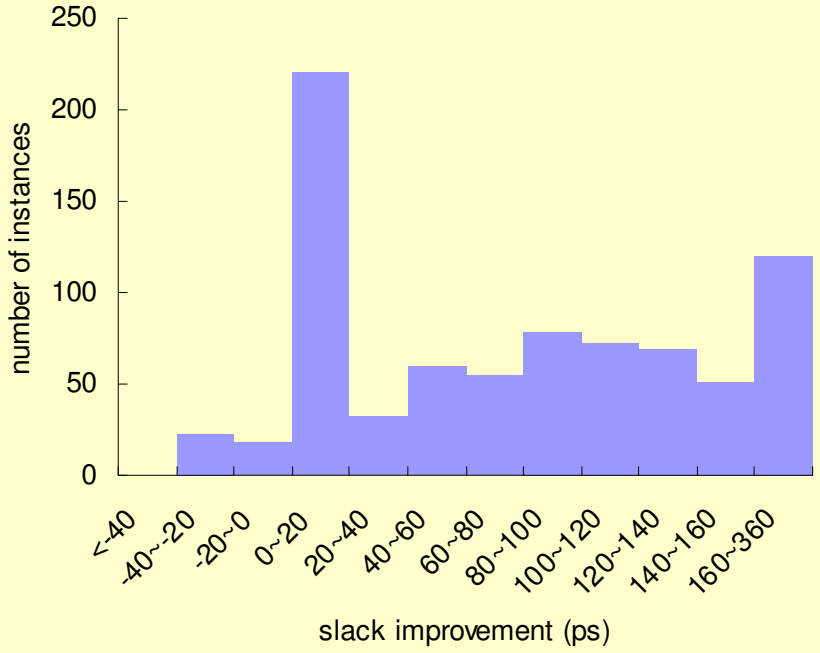
Experiment Setup

- 200 nets, 4 cores – 800 instances
- Compare:
  - Our heuristic
  - Instance based method:
    - Run Lillis' algorithm separately for each instance
    - Apply solution from a critical instance to all instances
- Two formulations:
  - Max slack: maximizing critical slack
  - Min cost: minimizing cost without negative slack

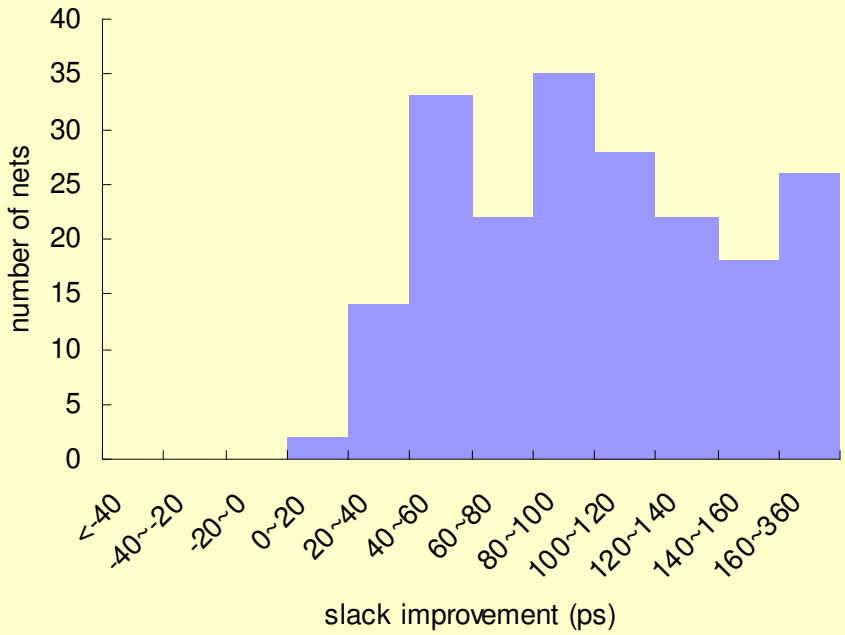
Max Slack Formulation

	Instance Based Baseline	Our Heuristic
Avg slack improvement/net (*ps*)	0	102.08
Avg slack improvement/instance (*ps*)	0	77.69
Total buffer cap (*fF*)	2503.66	2514.65
Total CPU time (*s*)	6408	849

Slack Improvement: Max Slack



Ours vs. instance based in term of instances

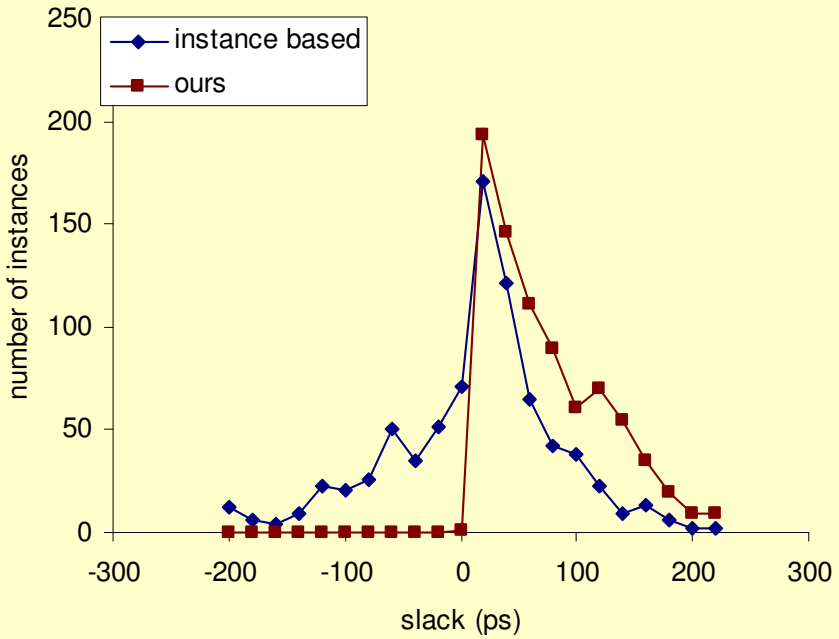


Ours vs. instance based in term of nets

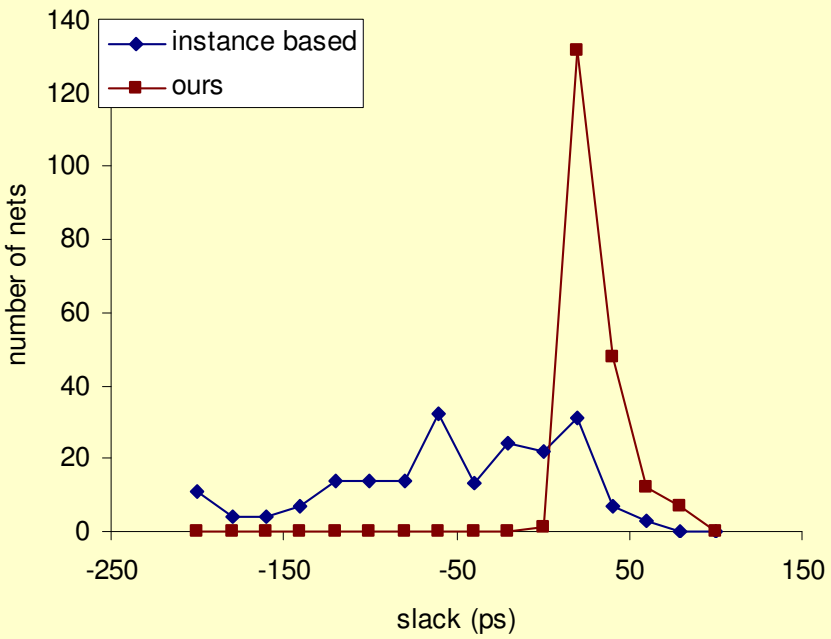
Min Cost with Timing Constraint

	Instance Based Baseline	Our Heuristic
Total # of nets with timing violations (*ps*)	155 (77:50% of 200 nets)	0
Total # of instances with timing violations (*ps*)	282 (35:25% of 800 instances)	0
Total buffer cap (*fF*)	2312.12	2314.77
Total CPU time (*s*)	6412	851

Slack Histogram: Min Cost Solutions



Ours vs. instance based in term of instances



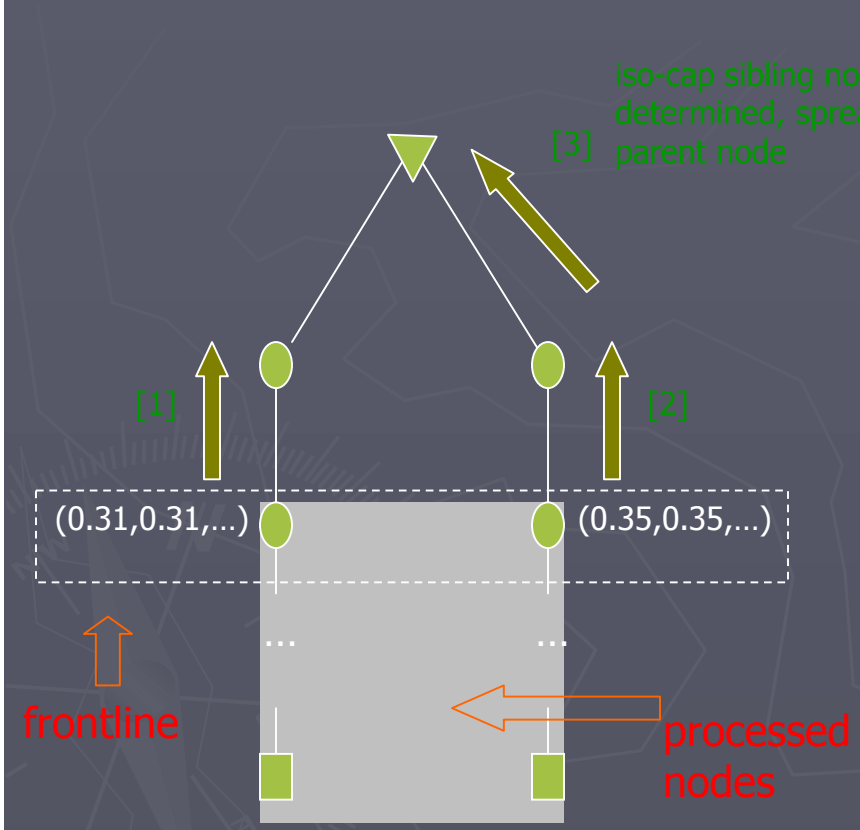
Ours vs. instance based in term of nets

Conclusion

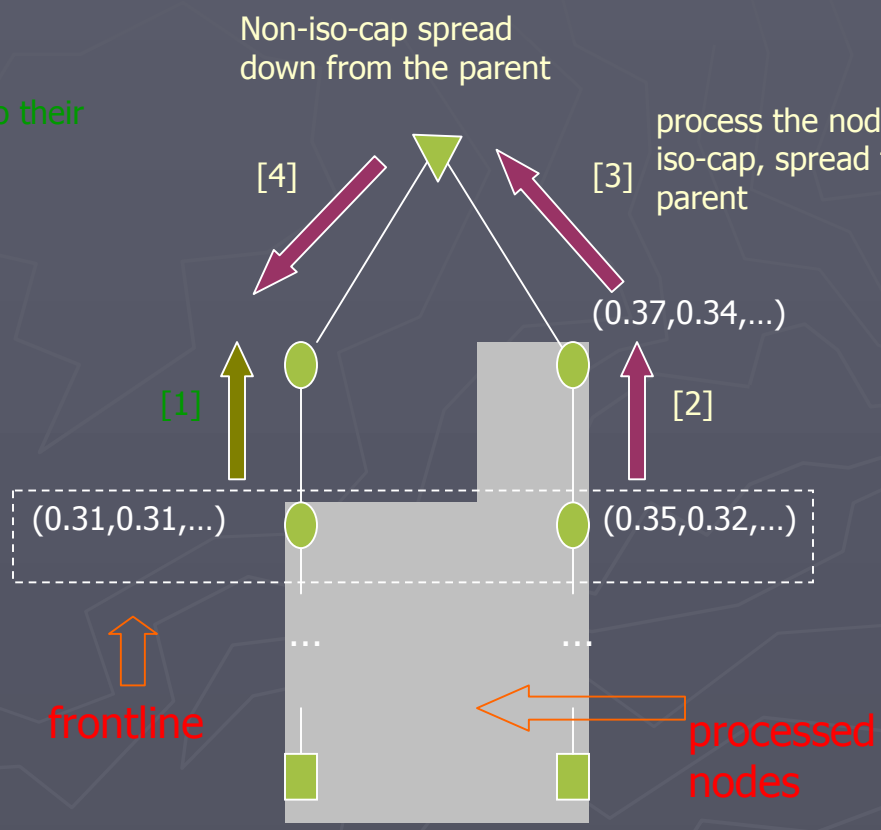
- A heuristic for multi-scenario buffering is proposed
- Compared to naïve application of Lillis' algorithm, our heuristic
  - increases slack significantly
  - avoids timing violations
  - reduces computation runtime
  - about the same buffer cost

Thank You!

Iso-Cap Detection – an example



Iso-cap spread example



Non-iso-cap spread example