Architectural Retiming:

A Technique for Optimizing Latency-Constrained Circuits

Soha Hassoun & Carl Ebeling

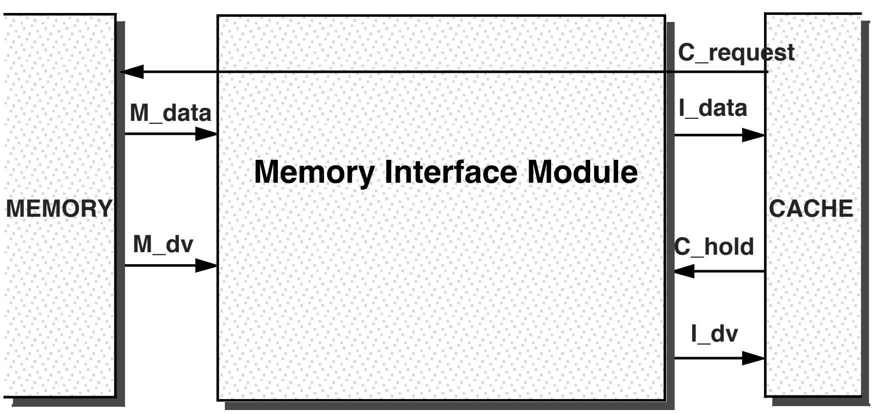
University of Washington
Computer Science and Engineering Department
Seattle, WA

Why Is It Hard
to
Design Fast Circuits?

Too Much Work,

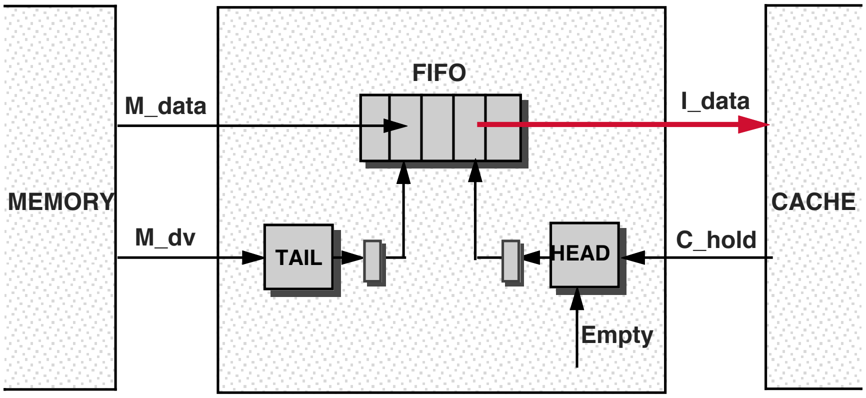
Too Little Time.

Example



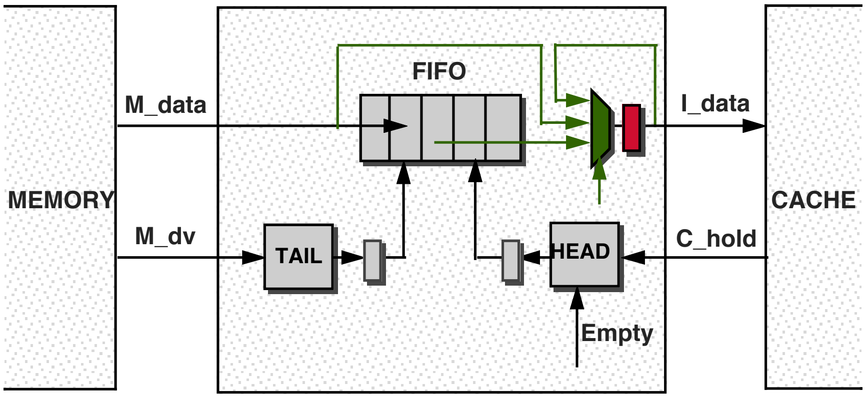
Specified Latency = 1 Cycle

Memory Interface Module

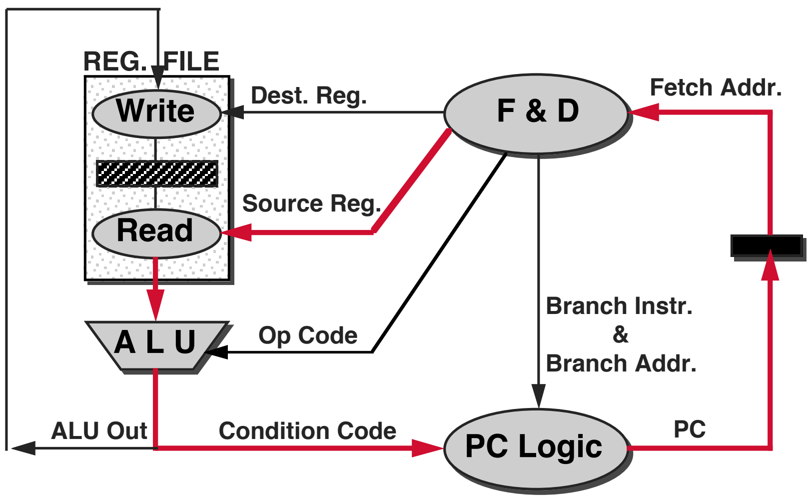


Problem: I_data Arrives Late at Cache

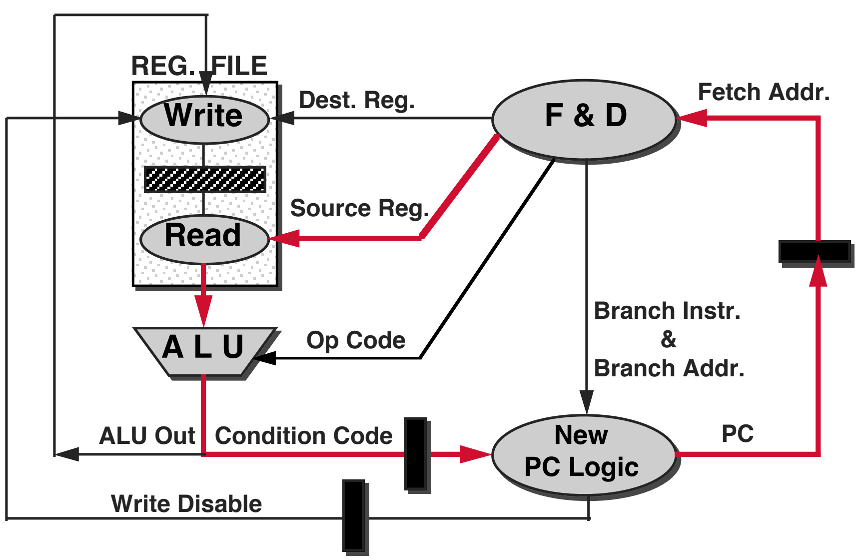
Re-designed Memory Interface Module



A Simplified Controller



An Improved Controller



Architectural Retiming

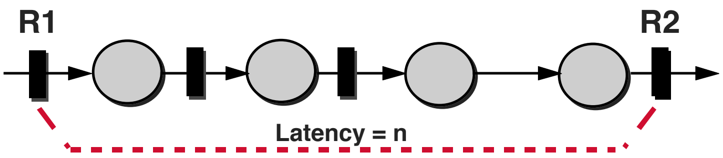
- Identify performance bottleneck
- Provide systematic way to perform optimizations
- Formalize and generalize ad hoc architectural transformations
- Explore design space
- Speed up the design cycle

Outline

- Problem
- Finding performance bottleneck
- Approach: Architectural Retiming
  - Example revisited
- Results
- Related work
- Conclusion

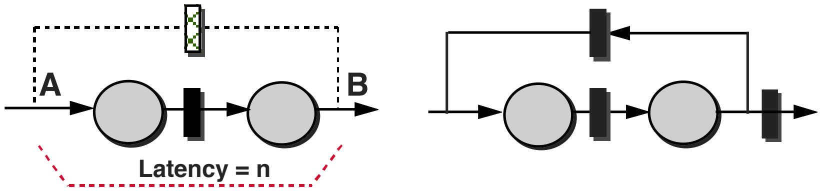
Problem: Latency Constraints

- Computation must complete in a fixed number of cycles



- Classification
  - Explicit: performance requirements
  - Implicit: cyclic dependencies

Finding Latency-Constrained Paths



- Model as cycles with fixed number of registers

- Find critical cycle,

  max (($\sum$ delay) / Number of Registers)

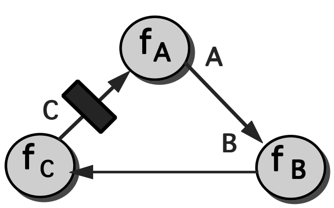
  [Hartmann & Orlin 91], [Burns 91]

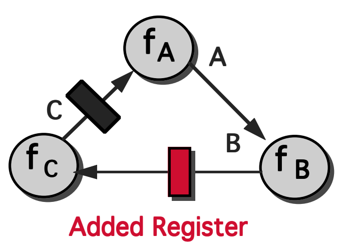
Architectural Retiming

For a Critical Cycle:

Decrease (($\sum$ delay) / Number of Registers)
by
adding a register
while
preserving circuit's functionality and latency

Pipelining Cycles





Original Operation

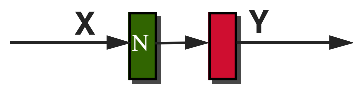
	t-1	t	t+1
A	a0	a1	a2
B	b0	b1	b2
C	c0	c1	c2

Modified Operation

	t-1	t	t+1	t+2
A	--	a1	–	a2
B	--	b1	–	b2
C	c0	--	c1	--

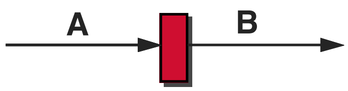
Negative Register/Pipeline Register

$Y^t = X^t$



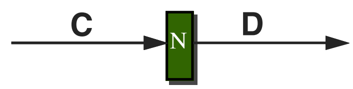
Pipeline Register

$B^{t+1} = A^t$

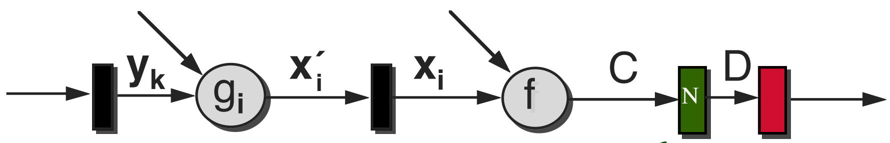


Negative Register

$D^t = C^{t+1}$



Implementing Negative Registers



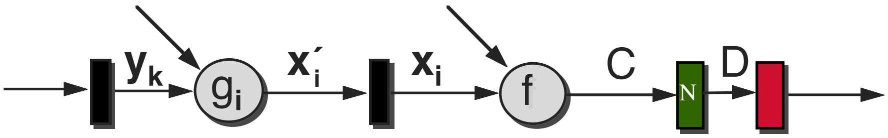
Precomputation

Find a function that computes D in terms of y_k

Prediction

Each cycle, verify previous prediction:
If correct, predict C again;
otherwise, correct misprediction

Implementing Negative Registers:
Precomputation



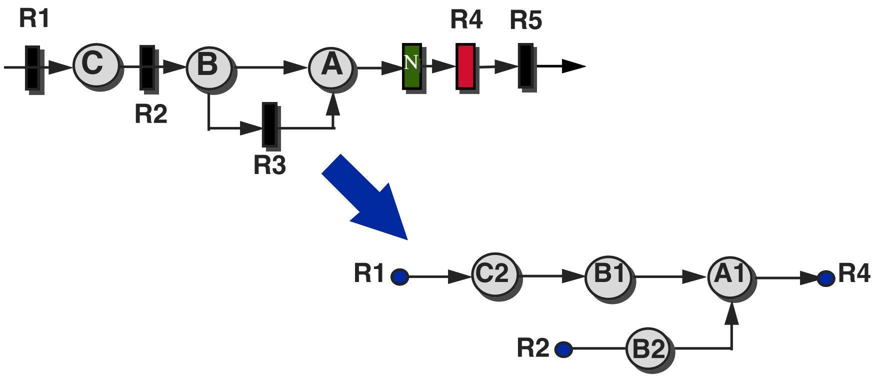
$$D^t = C^{t+1}$$
$$= f(\ldots, x_i^{t+1}, \ldots)$$

$$x_i^{t+1} = x'^{\,t}_i = g_i(\ldots, y_k^t, \ldots)$$

$$D^t = f(\ldots, g_i(\ldots, y_k^t, \ldots), \ldots)$$
$$= f'(\ldots, y_k^t, \ldots)$$

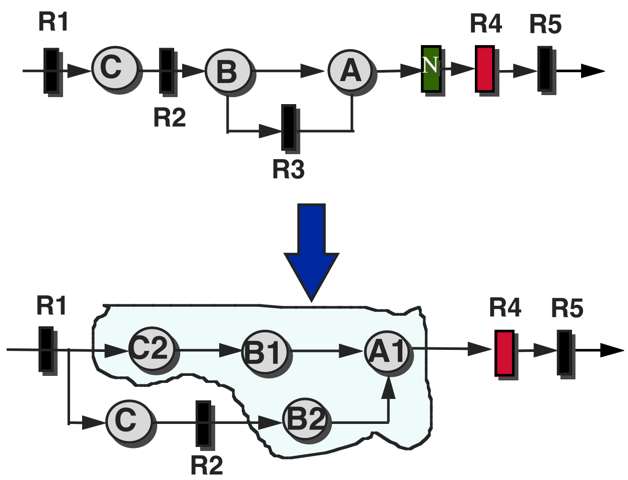
An Algorithm For Precomputation

1. Extract precomputation function

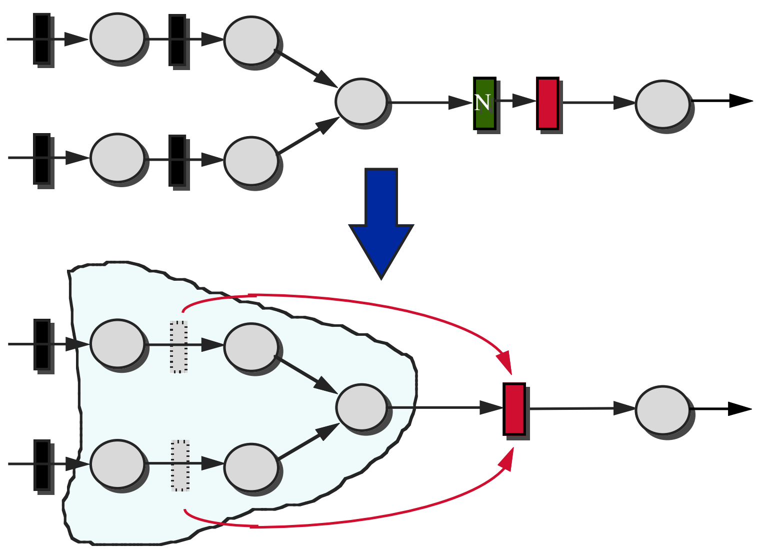


2. Optimize function
3. Add function back to circuit
4. Remove excess logic

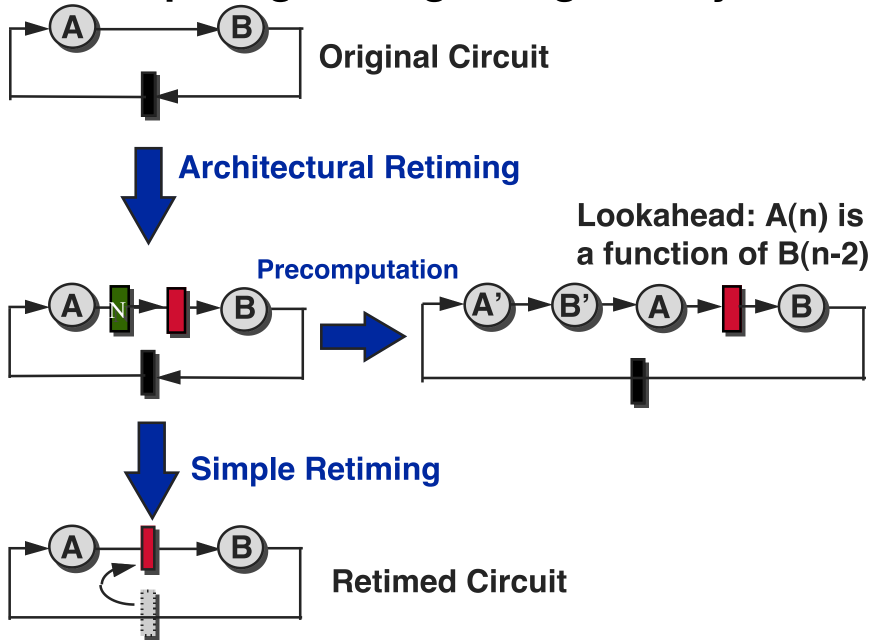
Precomputation



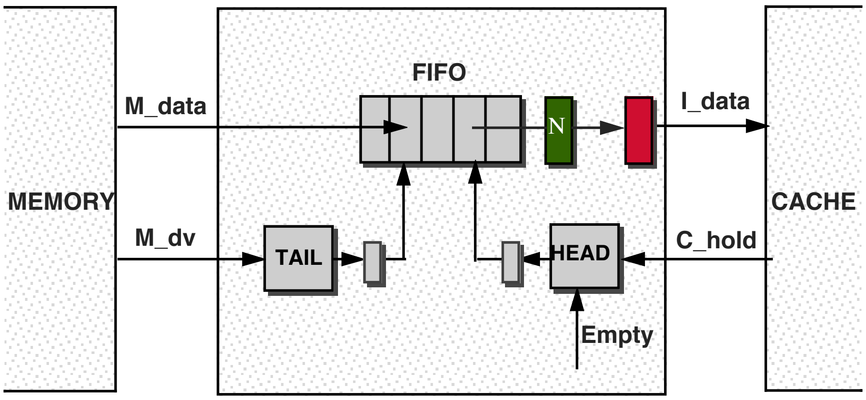
Simplest Precomputation:
A Local Forward Retiming Move



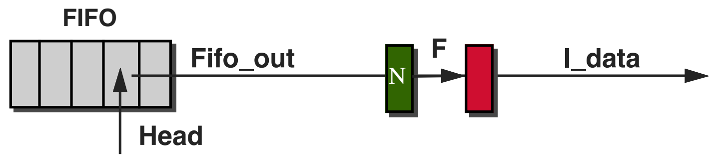
Precomputing in Single-Register Cycles



Precomputing
In The Memory Interface Module

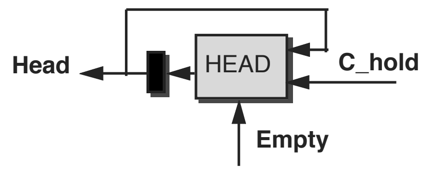


Applying Architectural Retiming



- $F^{t} = \text{Fifo\_out}^{t+1} = \text{FIFO}^{t+1}[\text{Head}^{t+1}]$

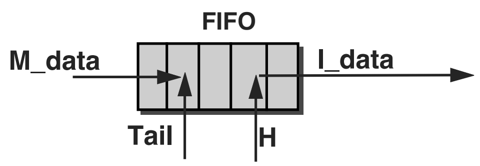
Precompute Head



- $\text{Head}^{t+1} = \begin{cases} \text{Head}^{t} & \text{if } (\text{Empty}^{t} \text{ or } \text{C\_hold}^{t}) \\ \text{Head}^{t}+1 & \text{if } !(\text{Empty}^{t} \text{ or } \text{C\_hold}^{t}) \end{cases}$

Architectural Retiming (cont.)

Precompute FIFO State

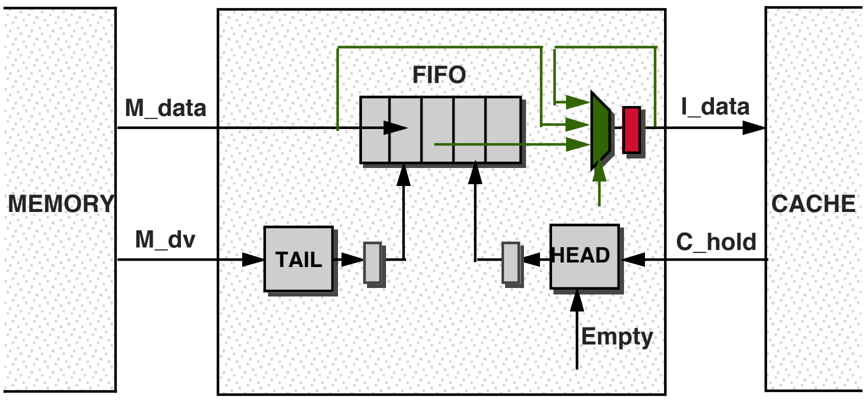


- $\text{FIFO}^{t+1}[\text{H}^{t}] = \begin{cases} \text{FIFO}^{t}[\text{H}^{t}] & \text{if } (\text{H}^{t} \mathrel{!}= \text{Tail}^{t}) \\ \text{M\_data}^{t} & \text{if } (\text{H}^{t} = \text{Tail}^{t}) \end{cases}$

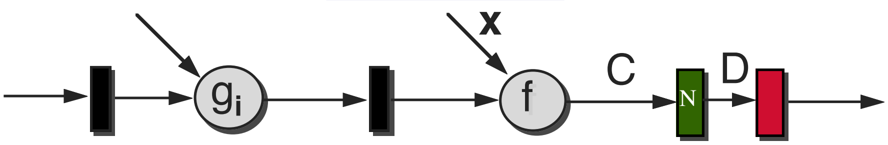
Additional Optimization

- Combinational logic optimization
- $\text{FIFO}^{t}[\text{Head}^{t}] = \text{I\_data}^{t}$

Architecturally-Retimed Memory Interface Module



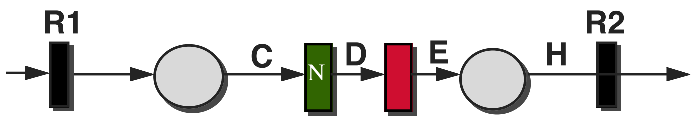
Implementing Negative Registers
Prediction



- Predict C^{t+1} if precomputation is unsatisfactory, or can't precompute
- Issues:
  - Prediction
  - Verification & correction
  - Nullifying mispredictions

The Effects of Mispredictions

	t-1	t	t+1	t+2
C	c0	c1	c2	c3
H	h0	h1	h2	h3



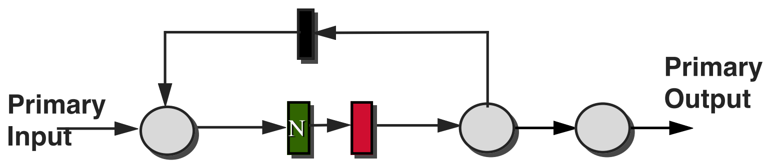
	t-1	t	t+1	t+2
C	c0	c1	c2	c3
D	c1*	c2*	c3*	c3
E	c0	c1	c2	c3*

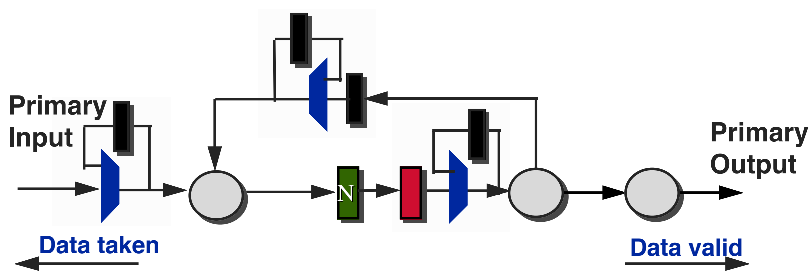
Nullifying Mispredictions

- Need a recovery strategy
  - Change I/O interface
  - Discard a result for each node
  - Prevent invalid data from propagating indefinitely in cycles
- Two Approaches
  - (A) Fix immediately
    - As Soon As Possible (ASAP) Restoration
  - (B) Allow invalid data to propagate - fix when needed
    - As Late As Possible (ALAP) Correction

ASAP Restoration

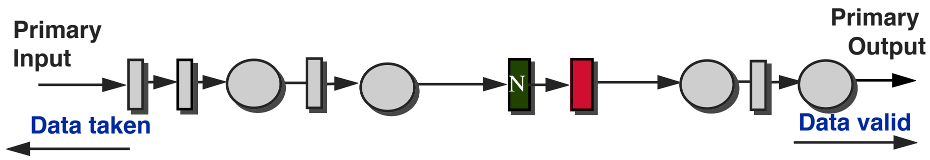
- **Idea:** "Shouldn't have done that!", but cannot turn back time
- Instead, replay previous conditions



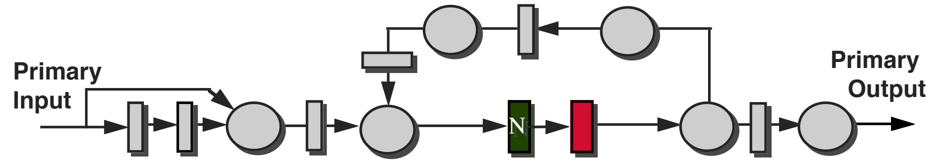


ALAP Correction

- Allow mispredictions to propagate and modify circuit elements
  - A simple pipeline



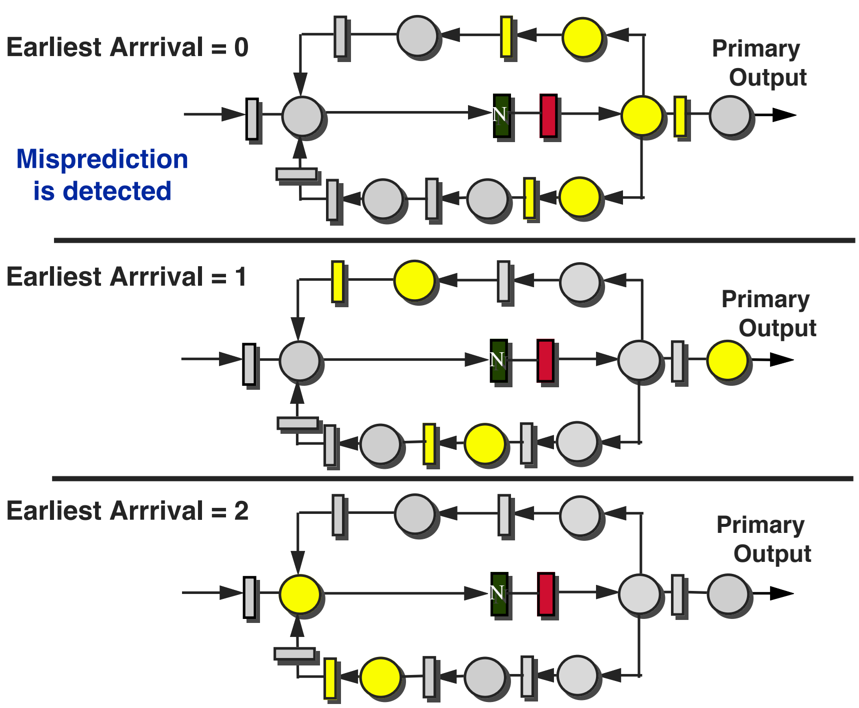
  - A more complex pipeline



Wavefronts

- An intuitive way to model propagation of invalid data

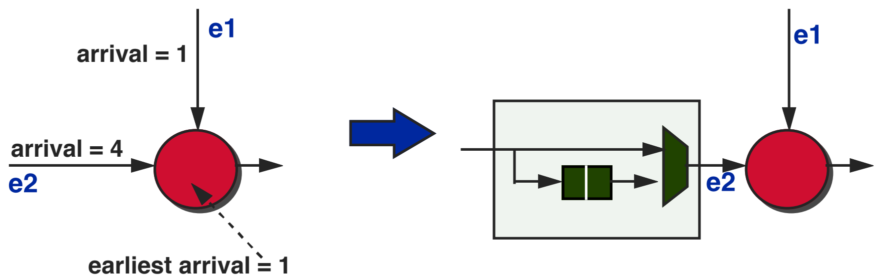
- Properties
  - Originate at output of added register & at primary inputs
  - Propagate to next pipeline stage each cycle, but don't propagate indefinitely
  - Predictable movement
    - Use shortest-path algorithm to calculate earliest arrival at each node



Modifying Combinational Logic

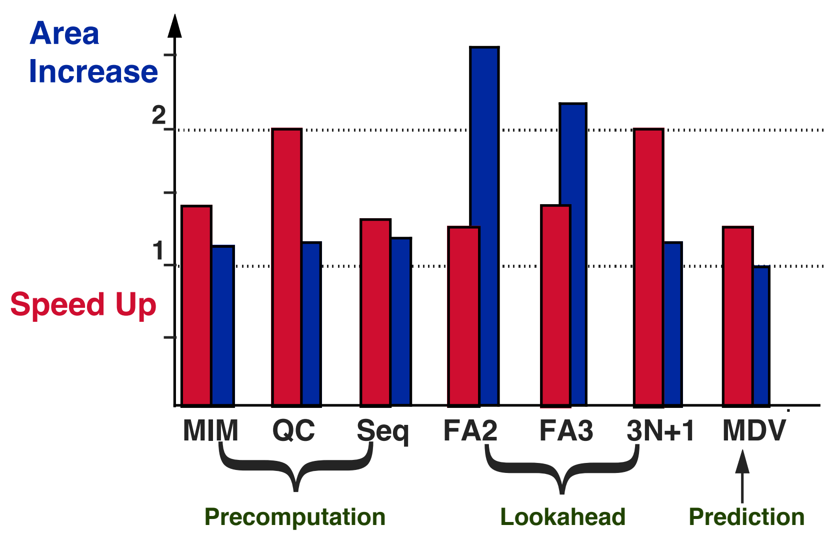
- Add FIFO queues along edges if needed
- Queue size (e) =

$$\left\lceil \frac{\text{arrival (e) - earliest\_arrival (target(e))}}{\text{Worst case separation}} \right\rceil$$



Results

Speed Up vs. Area Increase



Related Work

- Speculative computing
    - Generalize branch prediction [Holtmann & Ernst, 1993]
- Lookahead
    - Transform recursion to one dependent on earlier iteration [Kogge, 1981], [Parhi & Messerschmidtt, 1989]
- Precomputation
    - Optimize for low power [Alidina et. al., 1994]
    - Logic retiming [Ciesielski & Bommu, 1996]

Status

- Prediction using FSM
- Circuit implementation issues
    - Utilize handshaking signals
- *ART* (Architectural Retiming Tool)
- Evaluate more examples

Future Directions

- Concurrent optimization of Boolean and mathematical functions
- Bandwidth vs. latency
- Incremental performance evaluation

Summary

- New and general conceptual approach to improving performance of latency-constrained circuits
- Generalize, unify, and automate ad hoc architectural transformations
  - bypass, branch prediction, lookahead
- Good initial results