

Branch Runahead

Branch Runahead: An Alternative to Branch Prediction for Impossible to Predict Branches

Stephen Pruett + Yale N. Patt @ MICRO'21

Presented by: Ignacio Bricchi

Executive Summary

Motivation

- Programmes in certain fields are increasingly data driven

Problem

- Current branch prediction is not well suited to deal with data based predictions

Goal

- Provide an alternative solution to branch prediction for these “hard to predict branches”

Idea

- Coroutines that execute dependency chains leading up to an “hard to predict branch”

Challenges

- Detecting dependency chains
- Keeping co-routines in sync with core processor

Results

- Branch Miss Prediction per Kilo Instruction (BMPKI) Reduction of 40%
- Instructions Per Cycle (IPC) increase of 8%

Presentation outline



Paper Summary

Background

Mechanism

Results



Analysis

Strengths

Weaknesses

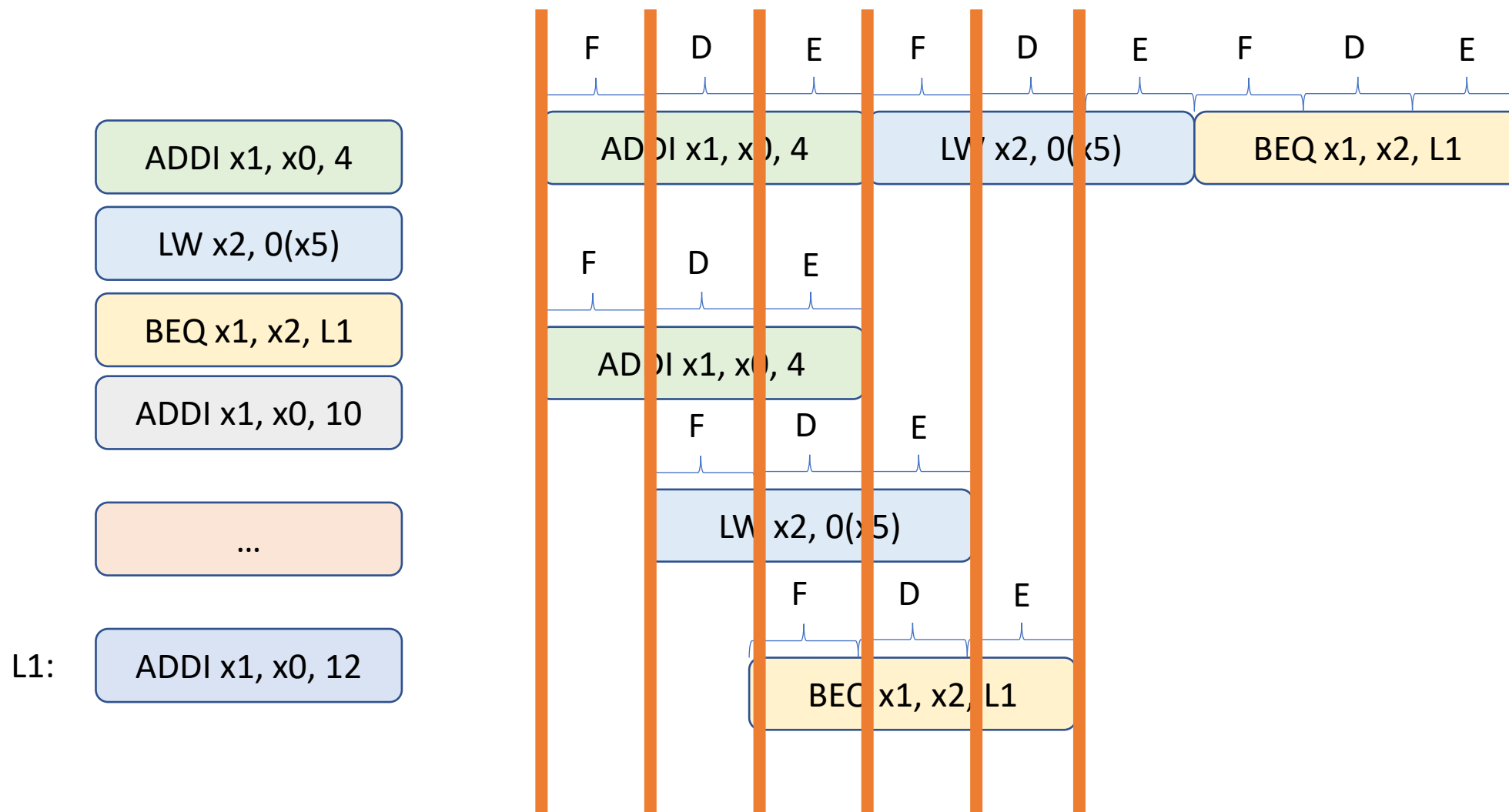
Thought/Ideas



Discussion/Questions

Background

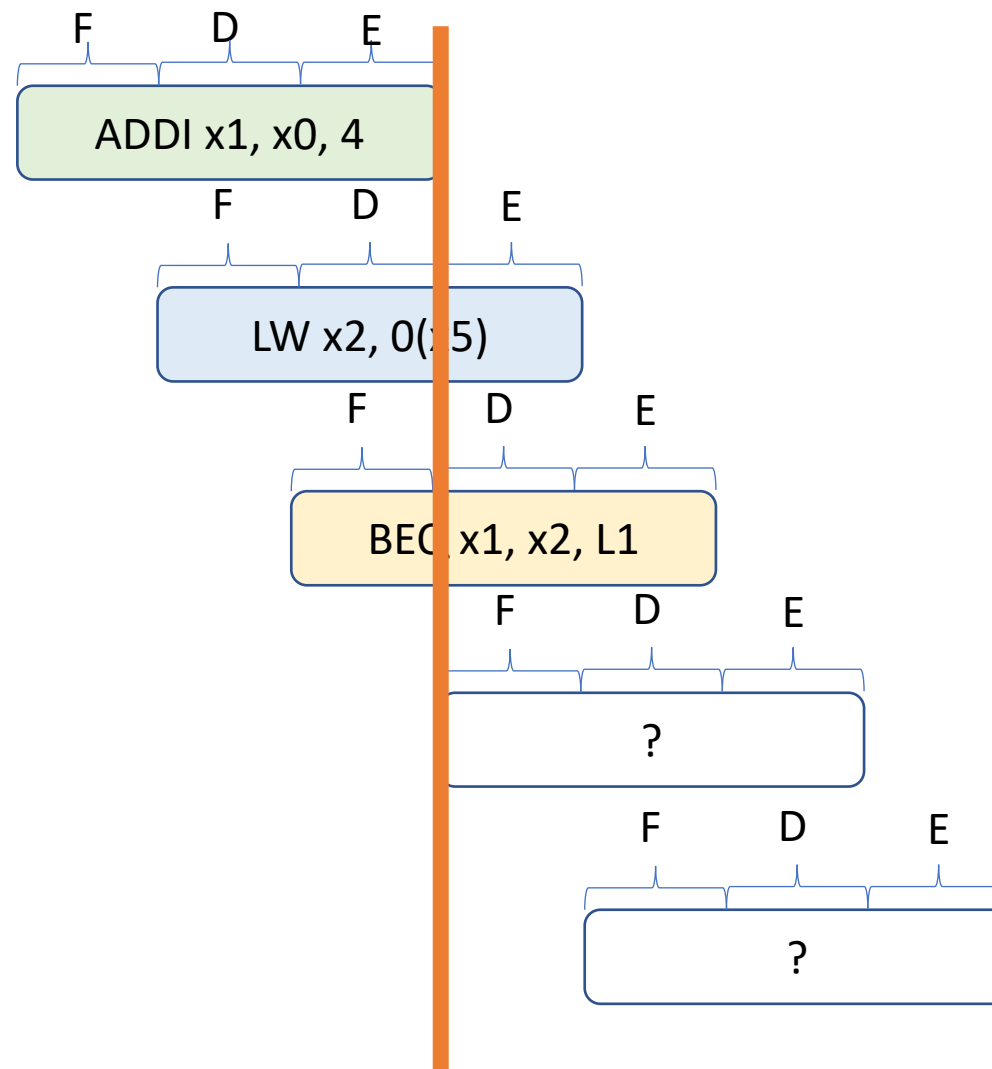
Pipelining introduced to increase throughput



Background



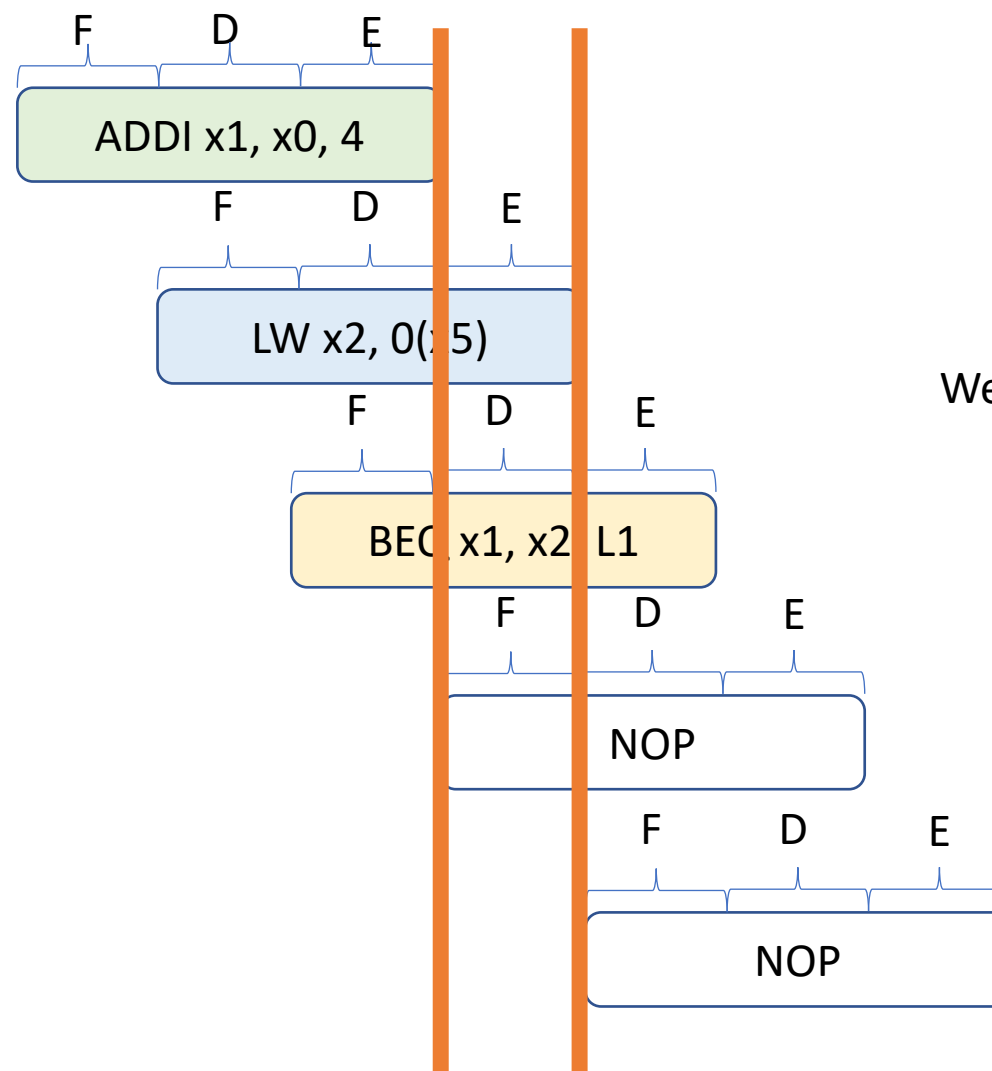
Introduces dependency hazards



Background

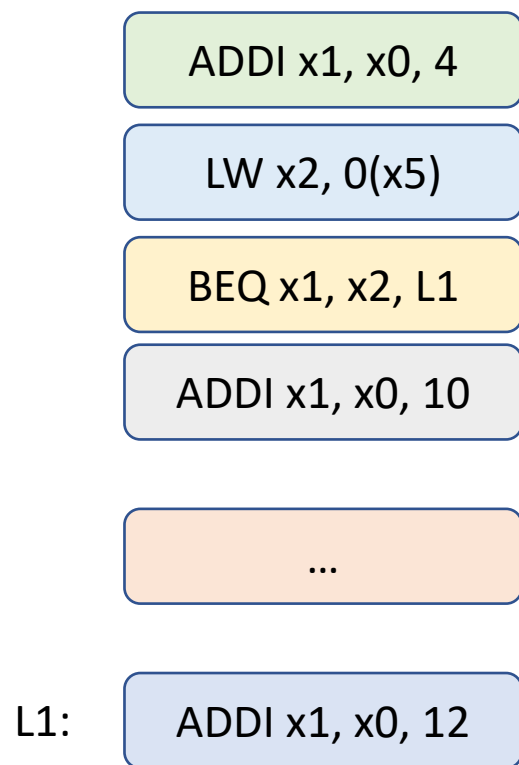


Introduces dependency hazards

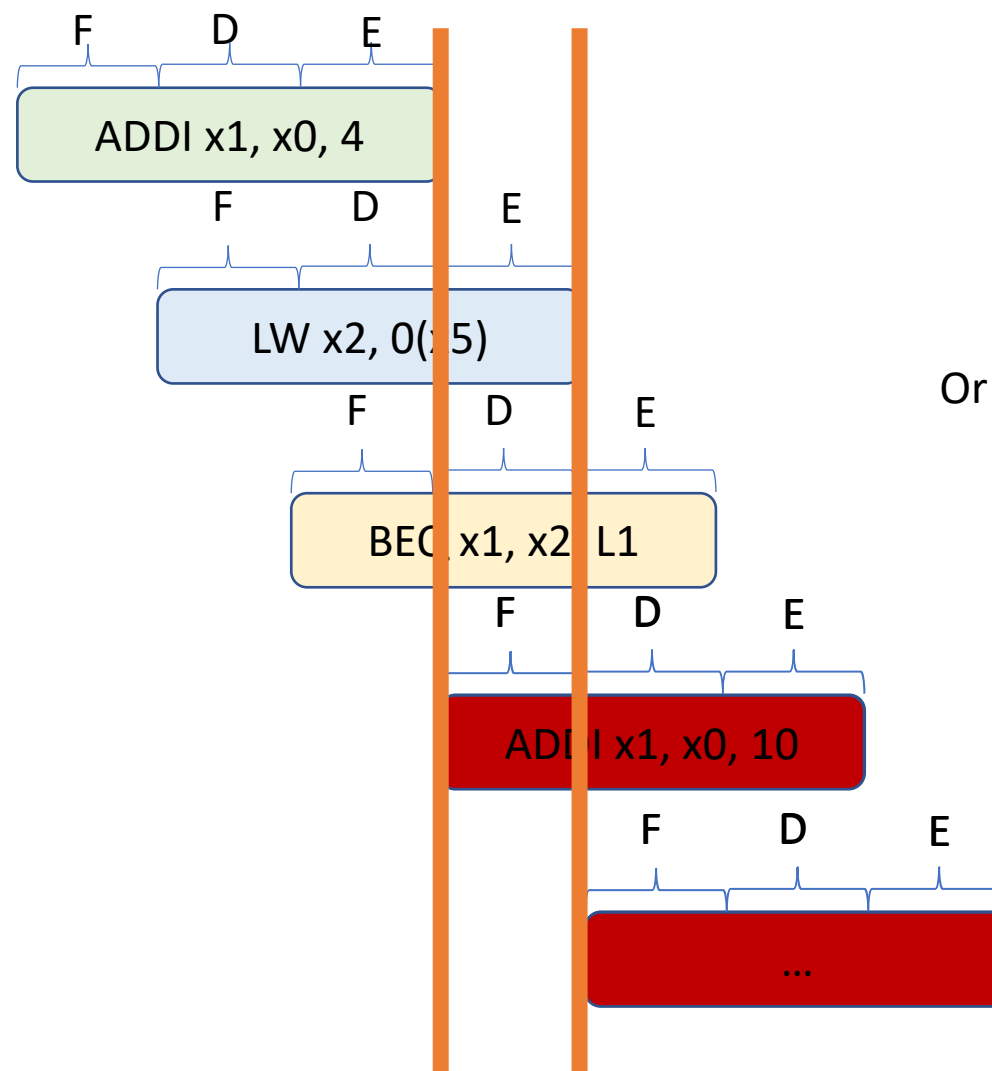


We can wait until data is ready

Background



Introduces dependency hazards



Background



Simple solution is to add bubbles, but this is slow and wasteful



Branch prediction solves this by making a prediction on what branch to take and flushing the pipeline in the event of an incorrect guess



Relies on the fact that most branches exhibit **predictable patterns** which can be detected as the programme runs

Motivation The problem

ADDI x1, x0, 4

LW x2, 0(x5)

BEQ x1, x2, L1

ADDI x1, x0, 10

...

L1: ADDI x1, x0, 12

Data driven programs tend to have branches that do not follow a history based pattern

Data dependant branch

Genome Analysis
Image Processing
Big Data problems

Motivation Alternatives

Highly parallelized architectures (e.g. Cray MTA)

- With enough parallelization you can switch between threads skipping those which are waiting for data dependencies
- Not possible for all kinds of workloads

Execute all paths (Never implemented commercially as far as I can tell)

- Wasteful in terms of area, and energy
- Exponential growth with pipeline depth

Use a coprocessor to compute **dependency chains** for resolving branches

Presentation outline



Paper Summary

Background

Mechanisms

Results



Analysis

Strengths

Weaknesses

Thought/Ideas



Discussion/Questions

Key Ideas

Software

- Threads are spawned to compute relevant instructions in parallel with main execution
- **High overhead** limits when co-processing can occur

Hardware

- Hardware approach, **run time** detection of dependency chains and special purpose compute unit
- No need for source code modification

Key Mechanisms

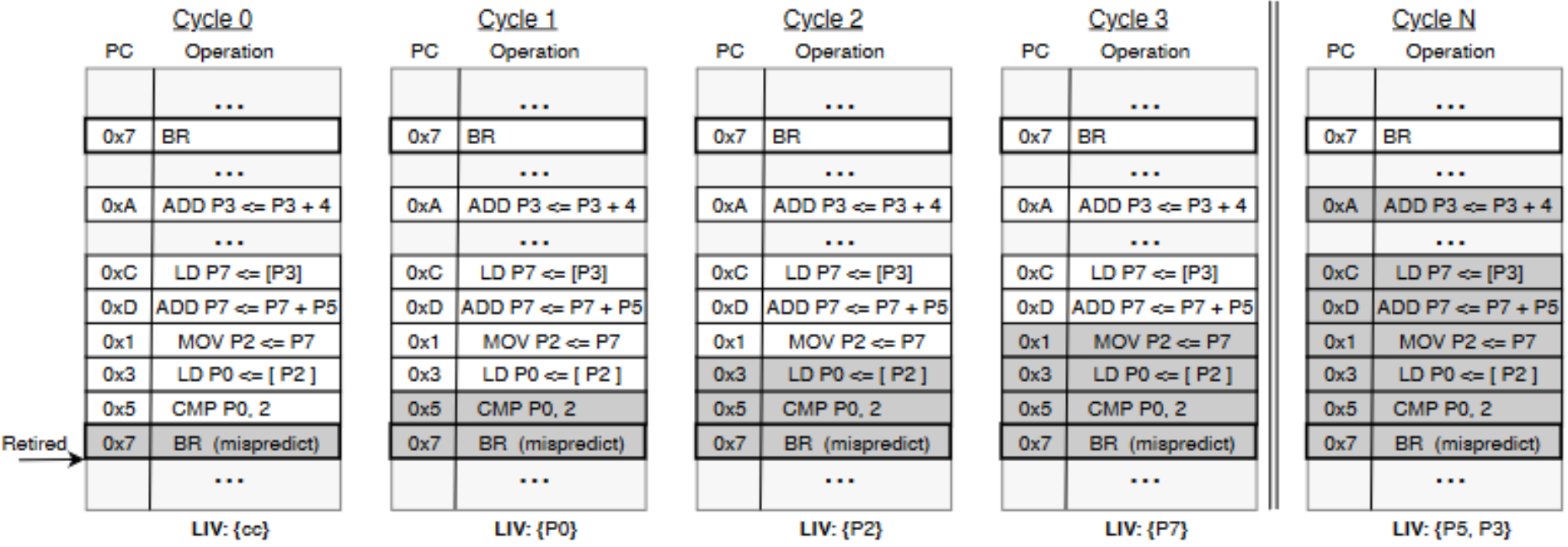
Detecting Hard To Predict Branches (HTB)

- Introduce a Hard Branch Table (HBT)
- Keep track of misprediction counts
- Once counter is saturated consider Branch HTP
- Counter is periodically decreased

HBT	
0x1:	00
0x2:	02
0x3:	07
0x4:	16

Key Mechanisms Dynamically detecting dependency chains

- Backwards dataflow walk
- Loop over a circular buffer of most recently retired uops
- Store live in registers and instructions in Data Dependence Cache (DDC)



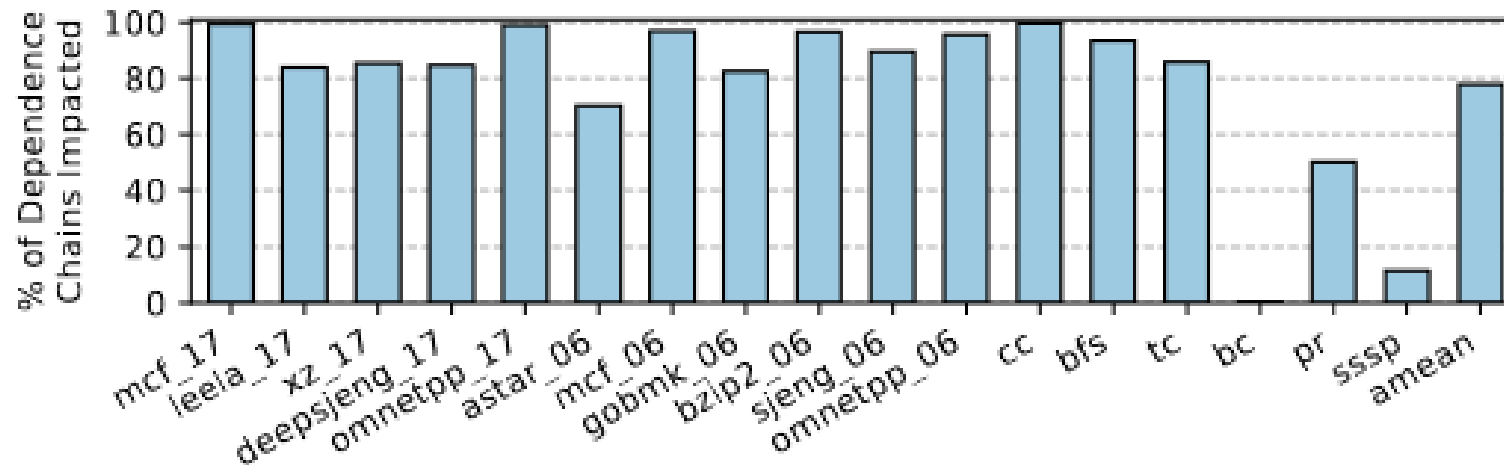
Key Mechanisms

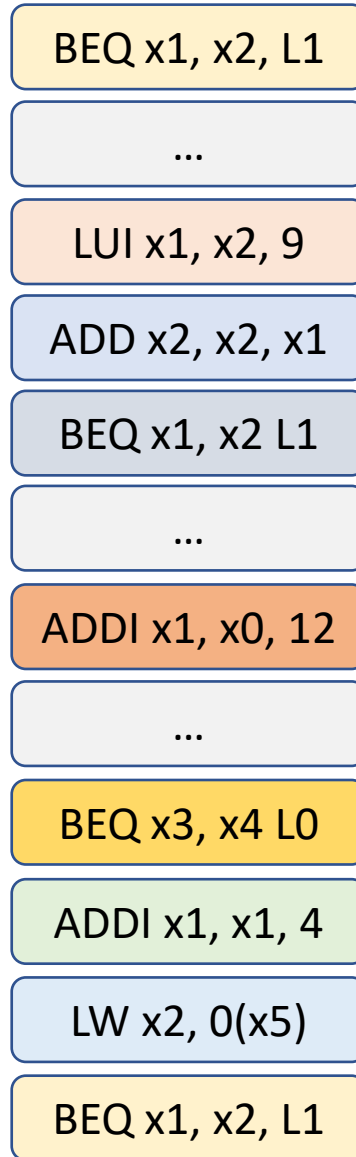
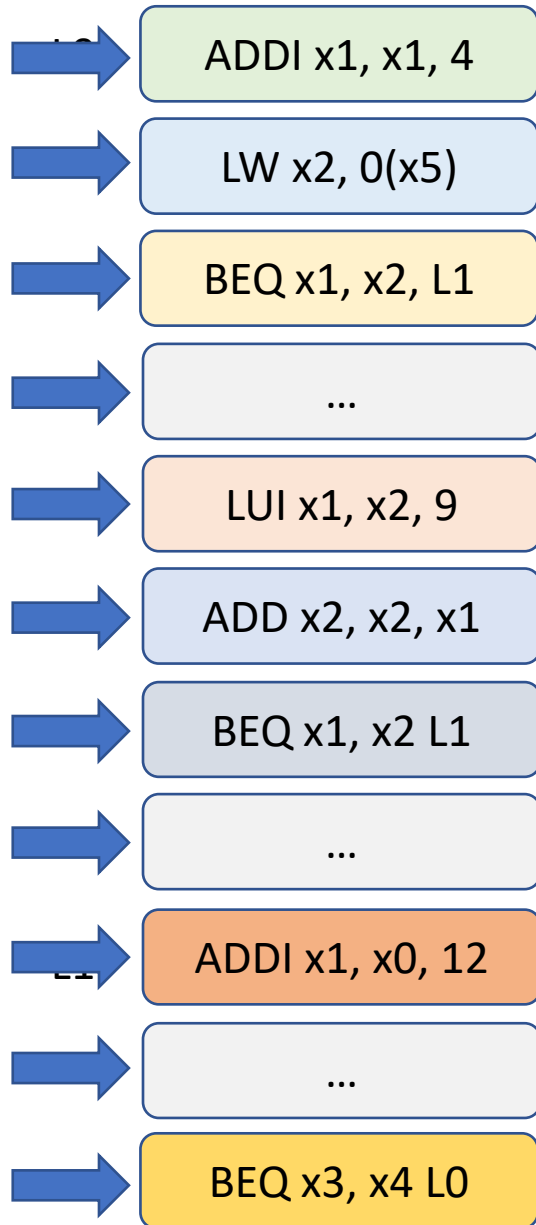
Triggering Execution

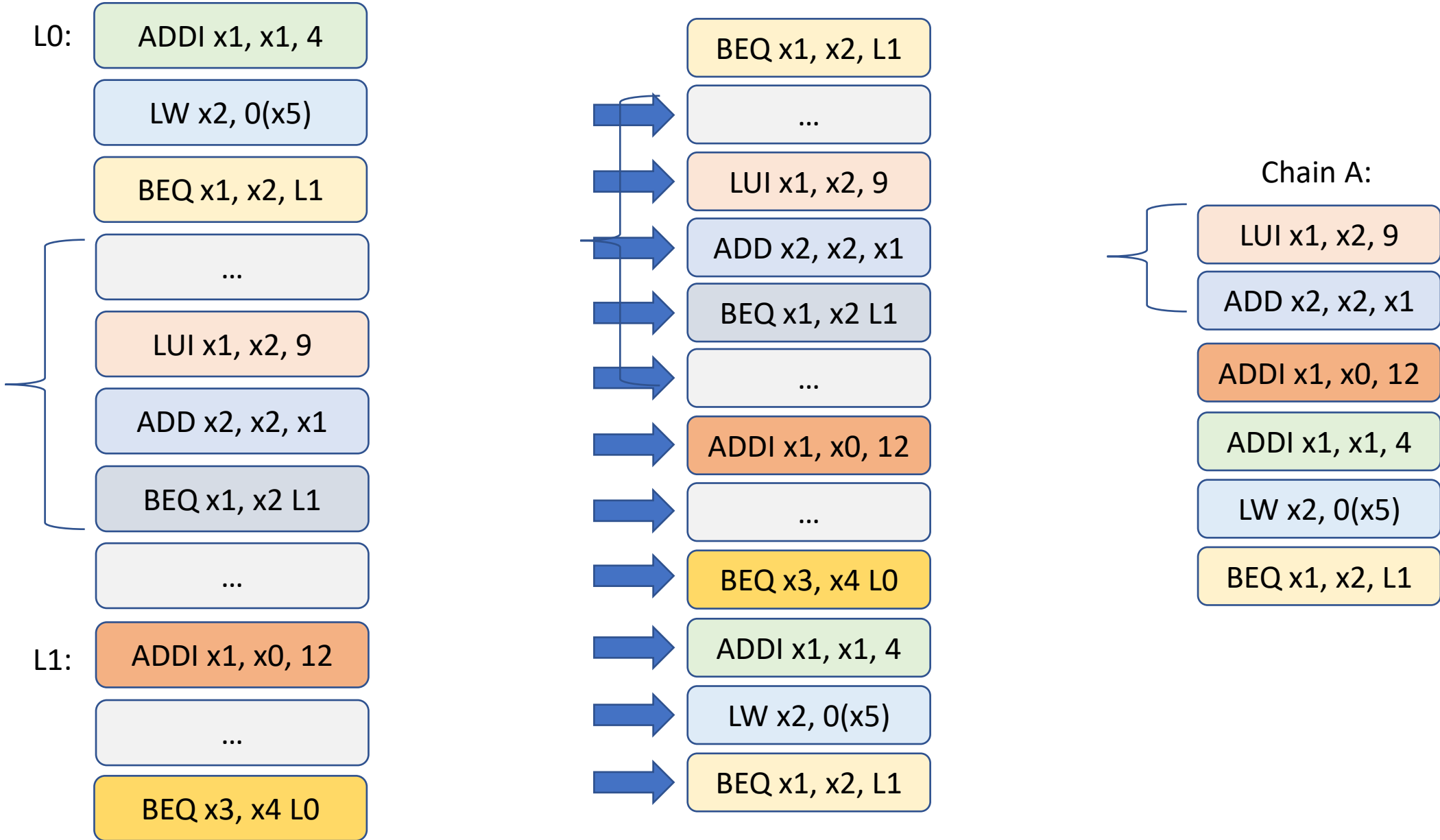
- When PC is reached and live in registers are ready copy them in and begin executing chain continuously
- Store prediction results in a queue
- When processor reaches branch read result from queue
- Keep track of accuracy of run ahead predictions to modulate between it and regular branch predictions

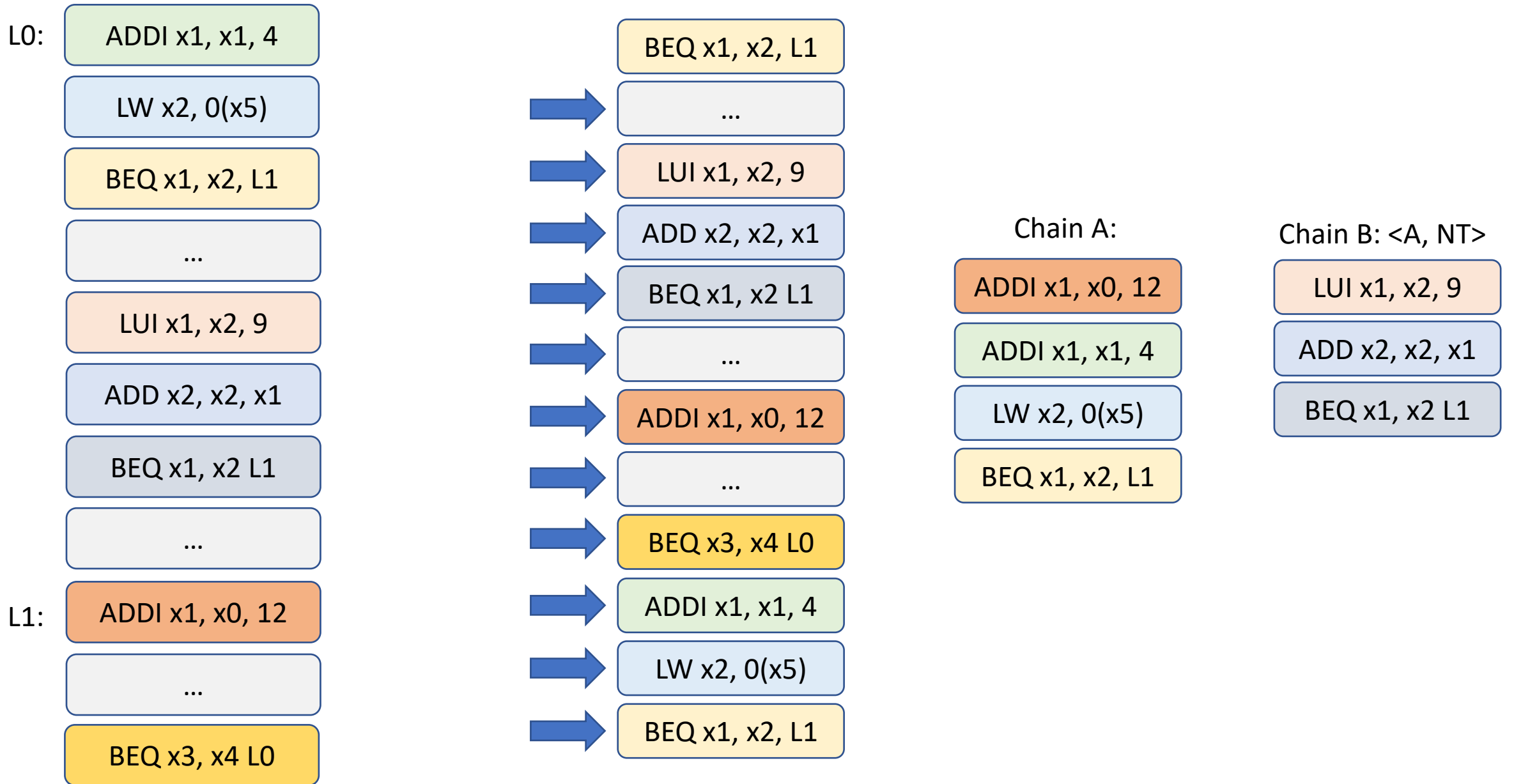
Key Insights/Innovations

- The code within a dependency chain may contain smaller frequently **changing branches** which require synchronisation limiting time a co-processor can remain **independent**









Key Insights/Innovations

- Detect these frequently changing branches and generate chains for them
- Use these to generate **inter chain dependencies** so that when one chain ends a second is immediately triggered as a continuation based on it's result
- Improves the time the programme can **execute independently** without the need for synchronisation

Presentation outline



Paper Summary

Background
Mechanisms
Results



Analysis

Strengths
Weaknesses
Thought/Ideas



Discussion/Questions





Results

Baseline Configuration

- Simulated using cycle accurate x86 simulator with custom extensions for the run ahead

Core	4-Wide Issue, 256-Entry ROB, 92-Entry Reservation Station, 3.2 GHz, 64KB TAGE-SC-L Branch Predictor [32]. Modeled by Scarab [2].
WPB	128-entry, 4-way, max merge point distance 256 uops.
L1 Caches	32 KB I-Cache, 32 KB D-Cache, 64 Byte Lines, 2 Ports, 3-Cycle Hit Latency, 8-Way, Write-Back.
L2 Cache	2 MB 12-Way, 18-Cycle Latency, Write-Back.
Memory Controller	64-Entry Memory Queue.
Prefetchers	Stream: 64 Streams, Distance 16. Prefetch into LLC.
DRAM	DDR4, 8Gb, x8, 2400R, Modeled by Ramulator [20].

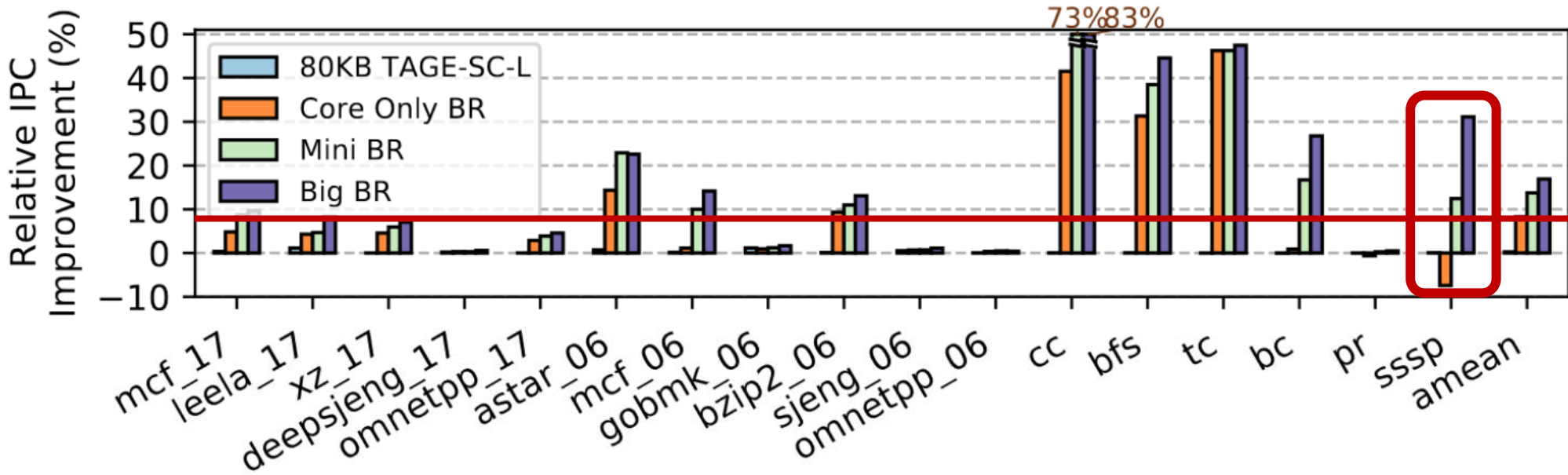
Results Branch Configuration

	80KB TAGE-SC-L
	Core Only BR
	Mini BR
	Big BR

	Core-Only (9KB)	Mini (17KB)	Big (Unlimited)
uOps	Integer: add/multiply/subtract/mov/load. Logical:and/or/xor/not/shift/sign-extend.		
Chain Cache	32-entry (2KB) 1 uop per entry, 4B per uop.		1024-entry
PRF	0 (0KB)	64x 8-entry (4KB)	1024x 8-entry
RSV	0 (0KB)	64x 32-entry (4KB)	1024x 32-entry
MSHRs	48-entry	48-entry	64-entry
Prediction Queue	16x 256-entry (4KB)		1024-entry
HBT	64-entry (1KB)		
CEB	512-entry (2KB)		2048-entry

Results

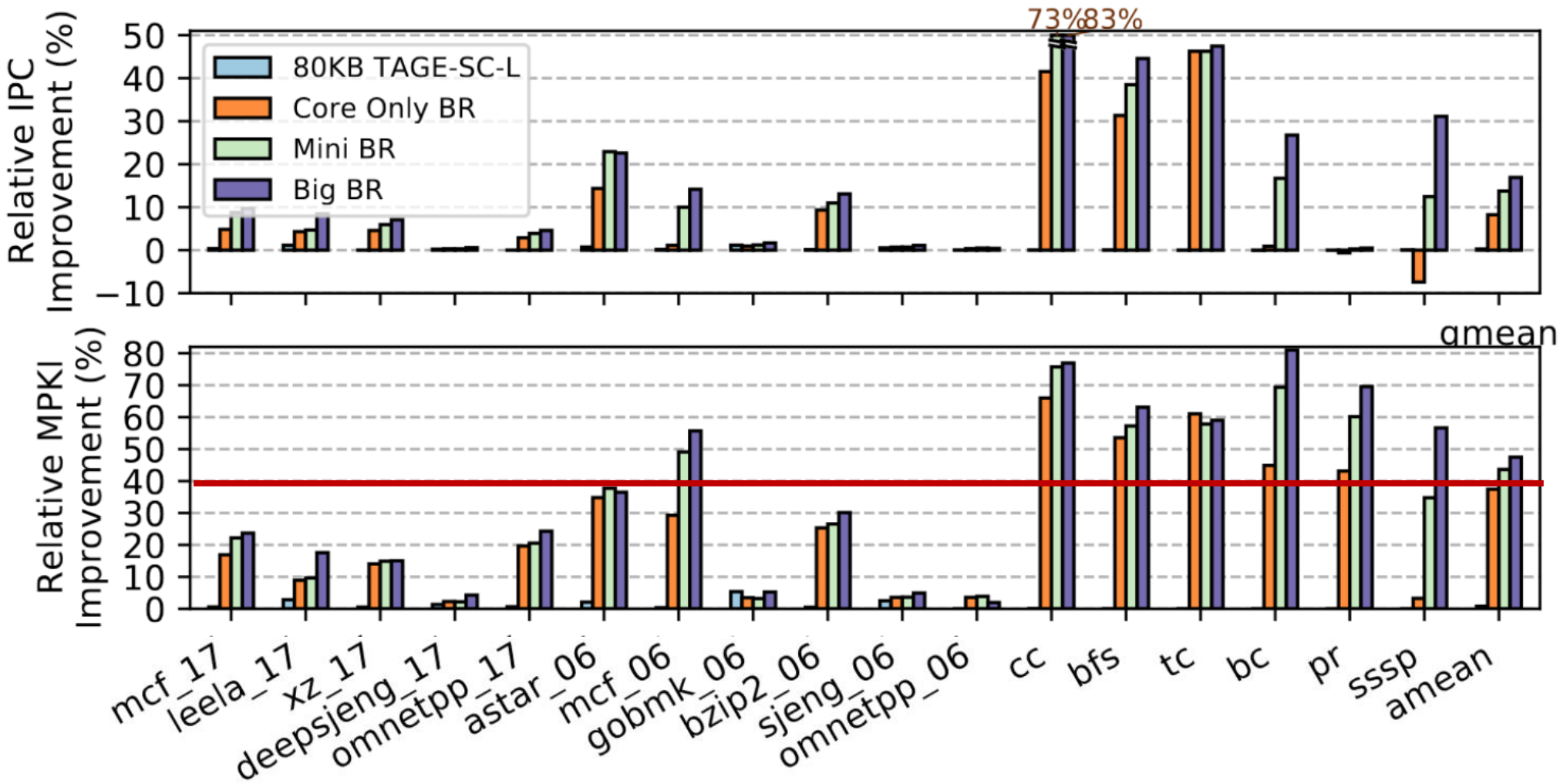
IPC increases by 8%



Results

IPC increases by 8%

BMPKI decrease by 40%



Conclusion

Motivation

- Programmes in certain fields are increasingly data driven

Problem

- Current branch prediction is not well suited to deal with data based predictions

Goal

- Provide an alternative solution to branch prediction for these “hard to predict branches”

Idea

- Coroutines that execute dependency chains leading up to an “hard to predict branch”

Challenges

- Detecting dependency chains
- Keeping co-routines in sync with core processor

Results

- BMPKI Reduction of 40%
- IPC increase of 8%

Presentation outline



Paper Summary

Background
Mechanism
Results



Analysis

Strengths
Weaknesses
Thought/Ideas



Discussion/Questions

Strengths

- Does not require changes to ISA's, this makes it possible to implement in current generation of processors
- Results suggest significant improvements
- Paper and design are well laid out and easy to follow

Novelty

- Other papers have already introduced runahead for various uses
 - (Multu +, Micro'03) Runahead execution: An effective alternative to large instruction windows
 - Proposes run ahead as an alternative to pre-fetching to reduce cache misses
 - (Chapell+ ISCA '02) Difficult-path branch prediction using subordinate microthreads
 - Proposes a very similar concept to this paper
- The key insight of adding interdependency between branches is novel and improves upon previous ideas by allowing run ahead to go further

Weaknesses

- Doesn't compare results against previous run-ahead models
- Not really an “alternative” for branch prediction, more of a supplementary feature

Conclusion

Motivation

- Programmes in certain fields are increasingly data driven

Problem

- Current branch prediction is not well suited to deal with data based predictions

Goal

- Provide an alternative solution to branch prediction for these “hard to predict branches”

Idea

- Coroutines that execute dependency chains leading up to an “hard to predict branch”

Challenges

- Detecting dependency chains
- Keeping co-routines in sync with core processor

Results

- MPKI Reduction of 40%
- IPC increase of 8%

Presentation outline



Paper Summary

Background
Mechanism
Results



Analysis

Strengths
Weaknesses
Thought/Ideas



Discussion

Discussion Highly parallelized architectures

- How does this solution compare to parallelized architectures?
 - Throughput
 - Latency
 - Complexity

Discussion Security

- What are the security implications of this design?
- Spectre V1 works by training the branch predictor to speculatively execute malicious code that leaves observable changes in the state of cache
- Can an attacker inject malicious dependency chains?
- What to do in the event of an exception?
- Could you use run-ahead to prevent Spectre V1?

Discussion HW + SW solution

- The author disregards SW solutions as they require computationally expensive threads, but is that the only way?
 - Extend the ISA to have explicit dependency chain markers on instructions
 - Take advantage of context available in source code that get's lost at the assembly level
 - Simplify the process of extracting dependency chains
 - Might miss some opportunities similar to how compiler schedulers are not as effective as out of order processors

Discussion

No branch prediction

- The author implements a way of detecting branch merge points in hardware, why not populate branch predictions with instructions from there?
 - Can we ensure no data dependencies from branch arms
 - Removes the need for prediction if enough independent instructions exist
 - Do independent instructions even exist?

Presentation outline



Paper Summary

Background
Mechanism
Results



Analysis

Strengths
Weaknesses
Thought/Ideas



Discussion/Questions

Executive Summary

Motivation

- Programmes in certain fields are increasingly data driven

Problem

- Current branch prediction is not well suited to deal with data based predictions

Goal

- Provide an alternative solution to branch prediction for these “hard to predict branches”

Idea

- Coroutines that execute dependency chains leading up to an “hard to predict branch”

Challenges

- Detecting dependency chains
- Keeping co-routines in sync with core processor

Results

- MPKI Reduction of 40%
- IPC increase of 8%