Dynamic Speculation and Synchronization of Data Dependences

Andreas Moshovos

Scott E. Breach

T. N. Vijaykumar

Guri Sohi

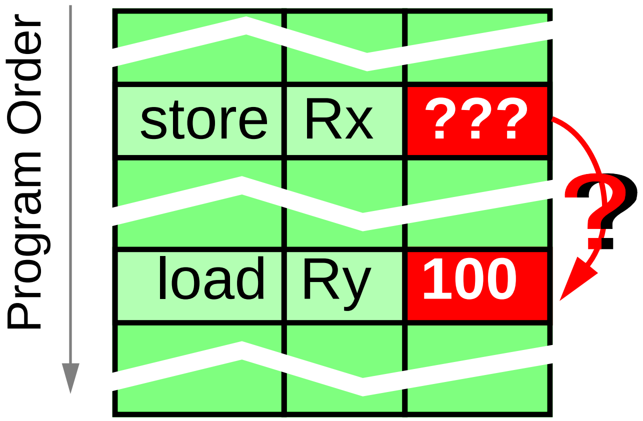
{moshovos, breach, vijay, sohi}@cs.wisc.edu

Computer Sciences Department

University of Wisconsin-Madison

The Problem

When to execute loads with unresolved dependences



Ideally...

load **waits for store**

only if dependent

Guess → Dependence Speculation

+ Gain when Right

− Penalty when Wrong

Wider Windows: Gain ↑ vs. Penalty ↑

Need for Intelligent Dependence Speculation

The Problem Revisited and The Solution

Intelligent Dependence Speculation

Q1. Which loads have dependences

Q2. How long to wait to:
(i). satisfy the dependence
(ii). maintain high gain

Dependence Speculation/Synchronization

A1. Predict dependences (load, store)

mis-speculation history

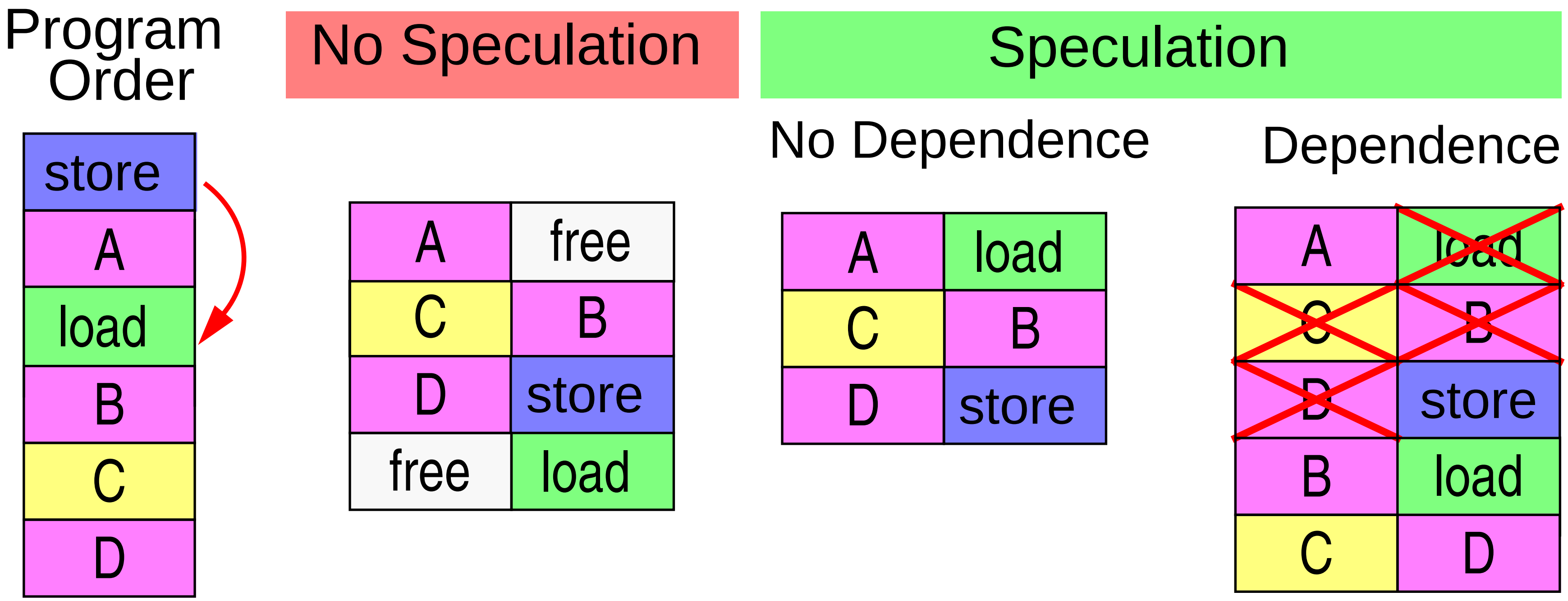
A2. Synchronize

dynamically assign full/empty bits

Roadmap

- ~~Overview~~
- **Dependence Speculation / Synchronization**
  - Dependence Speculation and Performance
  - Ideal Solution - Alternatives
  - Our Solution
- Evaluation
- Summary

Dependence Speculation and Performance



Program Order

store
A
load
B
C
D

No Speculation

A	free
C	B
D	store
free	load

Speculation

No Dependence

A	load
C	B
D	store

Dependence

A	~~load~~
~~C~~	~~B~~
~~D~~	store
B	load
C	D

Speculation may affect performance either way

Balance: Gain vs. Penalty

Penalty: (a). work thrown away

(b). opportunity cost

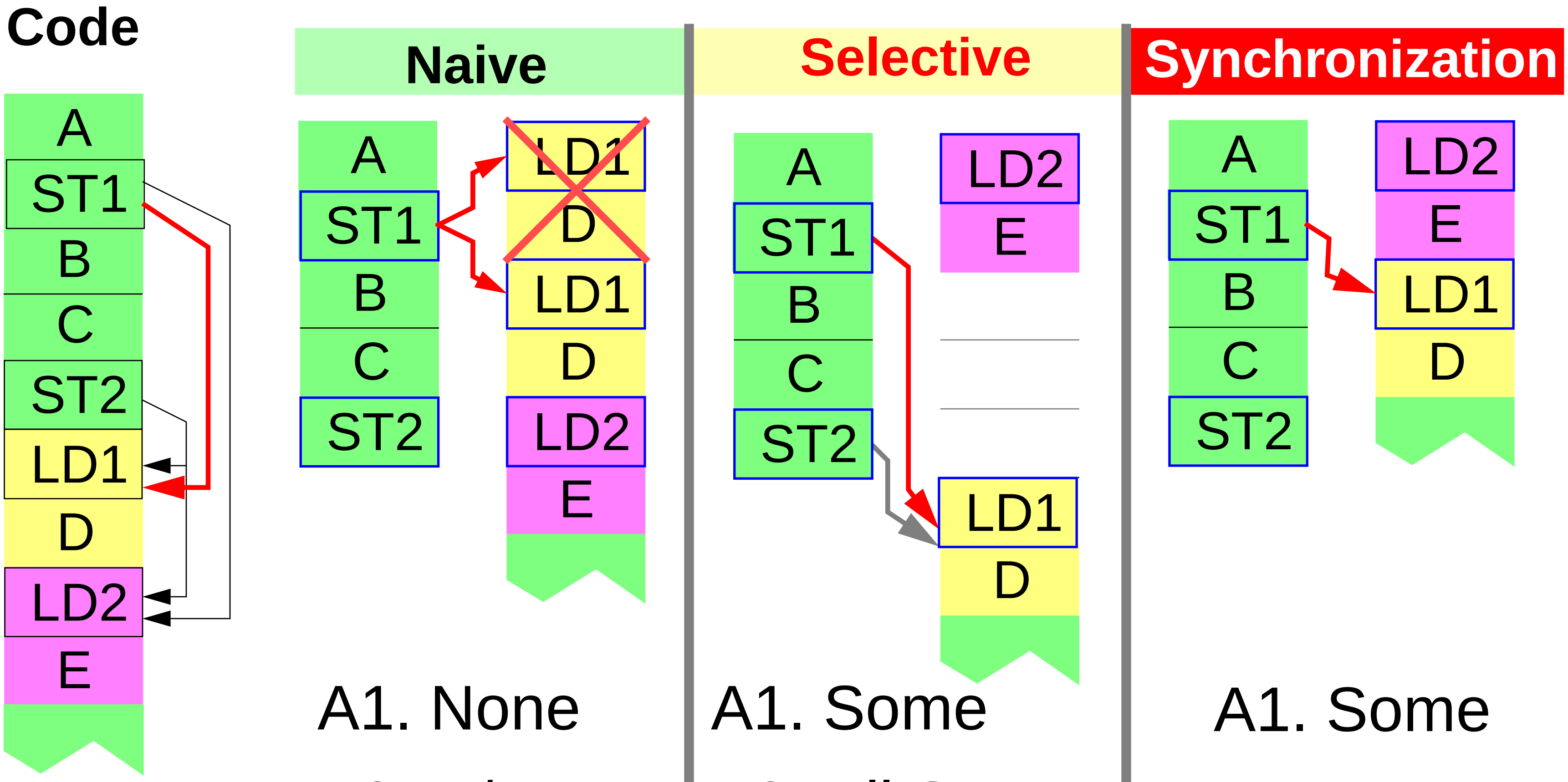
Dependence Speculation Policies

Q1. Which loads should wait

Q2. For how long

No Speculation	
A1. All	A2. For all previous stores
Naive	
A1. None	A2. N/A
Selective	
A1. Some	A2. For all previous stores
Synchronization	
A1. Some	A2. For the specific store

Speculation Policies - Examples



A1. None

A2. N/A

A1. Some

A2. All Stores

A1. Some

A2. Sync.

Q1. Which loads should wait

Q2. For how long

Our approach

Speculation/Synchronization, we need:

Identify → **(1).** Loads with dependences

Identify → **(2).** Relevant stores

(3). Enforce synchronization

How we do it:

- Parts **(1)** & **(2)**: **Predict load - store**

  Based on the history of mis-speculations

- Part **(3)**:

  Dynamically assigned synchronization variables

Dependence Prediction/Synchronization

Dependences as **(Load PC, Store PC):**

Small Working Set + Temporal Locality

A **small** table can:

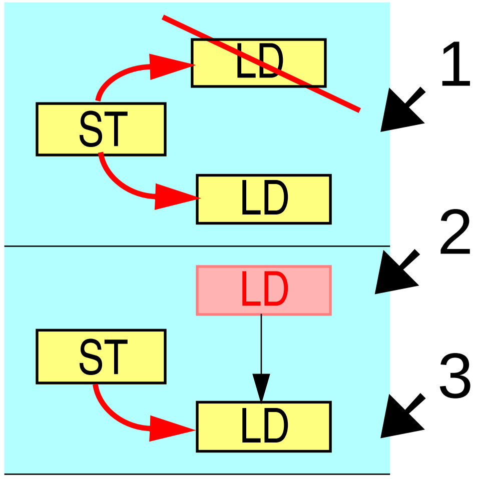
(1). Track recent mis-speculations

(2). Predict future load-store dependences

(3). Synchronize

- Eliminate most mis-speculations
- Aggresive speculation

Dependence Prediction/Synchronization

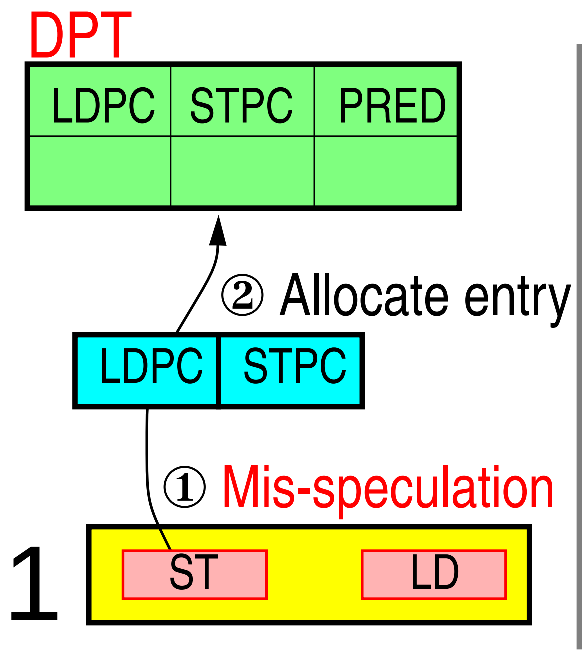


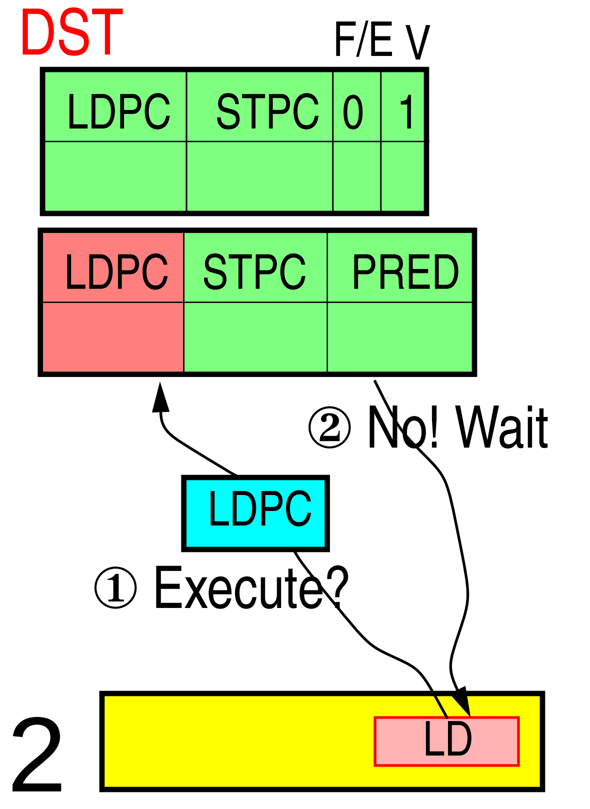
Dependence Prediction Table

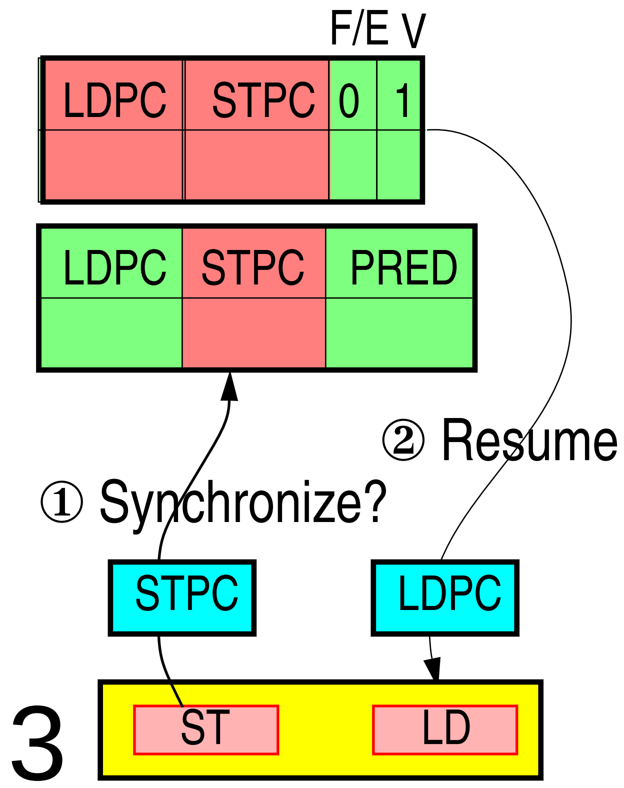
Predict Loads w/ Dependences

Dependence Synchronization Table

Enforce Synchronization







Other Issues

- Execution order varies
- Same dependence many times
    - Distinguish
    - Link load w/ appropriate store
    - Synchronization bits
- Multiple dependences per load or store
- Prediction
- Support for Control Speculation
- Distributed vs. Centralized

Addressed In The Paper...

Roadmap

- ~~Overview~~
- ~~Dependence Speculation/Synchronization~~
- **Evaluation**
    - Comparison of Speculation Policies
    - Accuracy of prediction
    - Performance
- Summary

Evaluation - Methodology

Machine model

Multiscalar

- 2-way OoO units, 8 PU

- Instruction driven - timing simulation
- Simulate all as realistically as possible

Benchmarks

- SPEC ‘95 for most (train/test up to 2 Billion instructions)
- SPECint ‘92 for some
- gcc 2.7.2, -O3 compiled

Evaluation

(1). Speculation Policies

Assumption: Perfect Prediction

Goal: Do we need Synchronization?

Is Selective good enough?

(2). Dependence Prediction Accuracy

Assumption: Real Prediction

Goal: Can we predict dependences?

(3). Speedup

Assumption: Real Prediction/Synchronization

Goal: What is the impact on performance.

Comparison of Speculation Policies

If we had Perfect Dependence Prediction...

Compared to No Speculation:

Dependence Speculation wins

speedup: 25% - 140%

Compared to Naive Speculation:

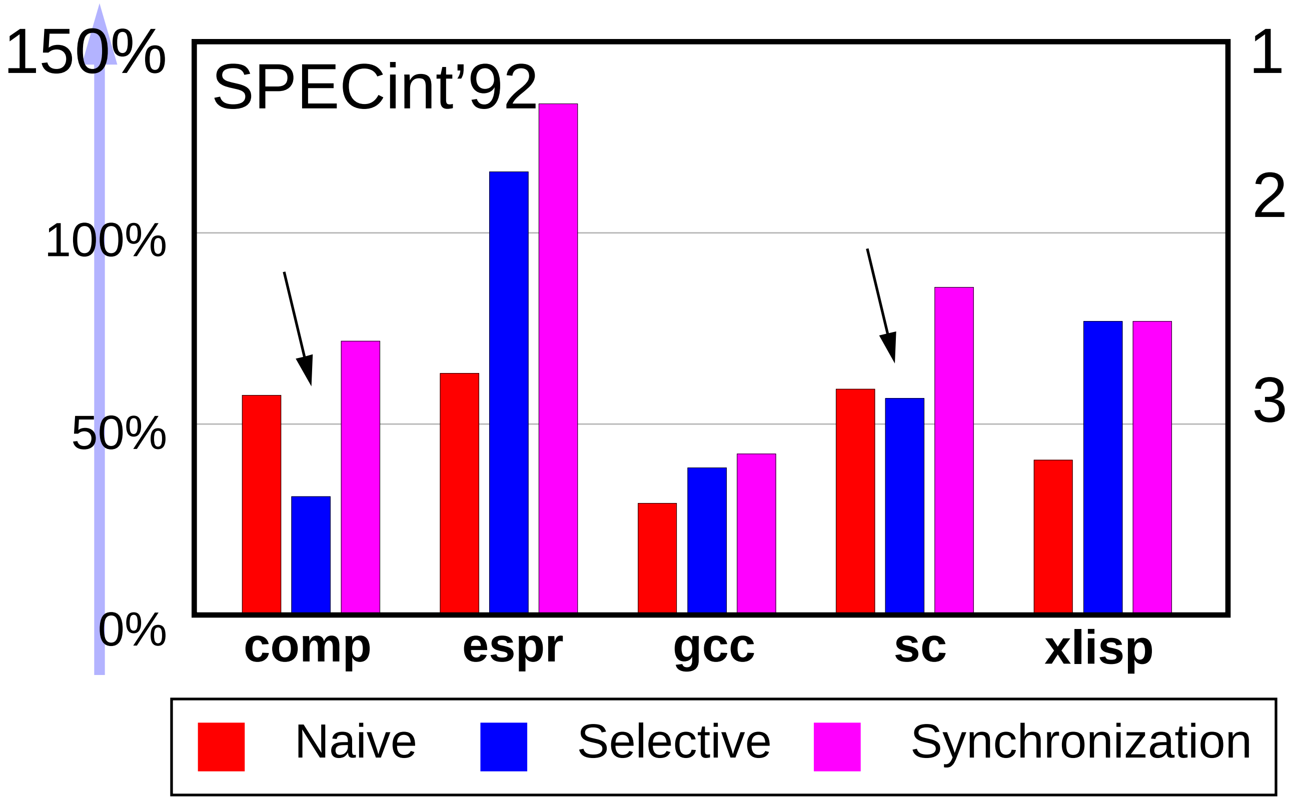
- Selective may do **much** worse
- Synchronization always improves

Need: What to Speculate + How Long to Wait

Comparison of Speculation Policies

If Perfect Dependence Prediction was available...

Speedups Relative to No Speculation



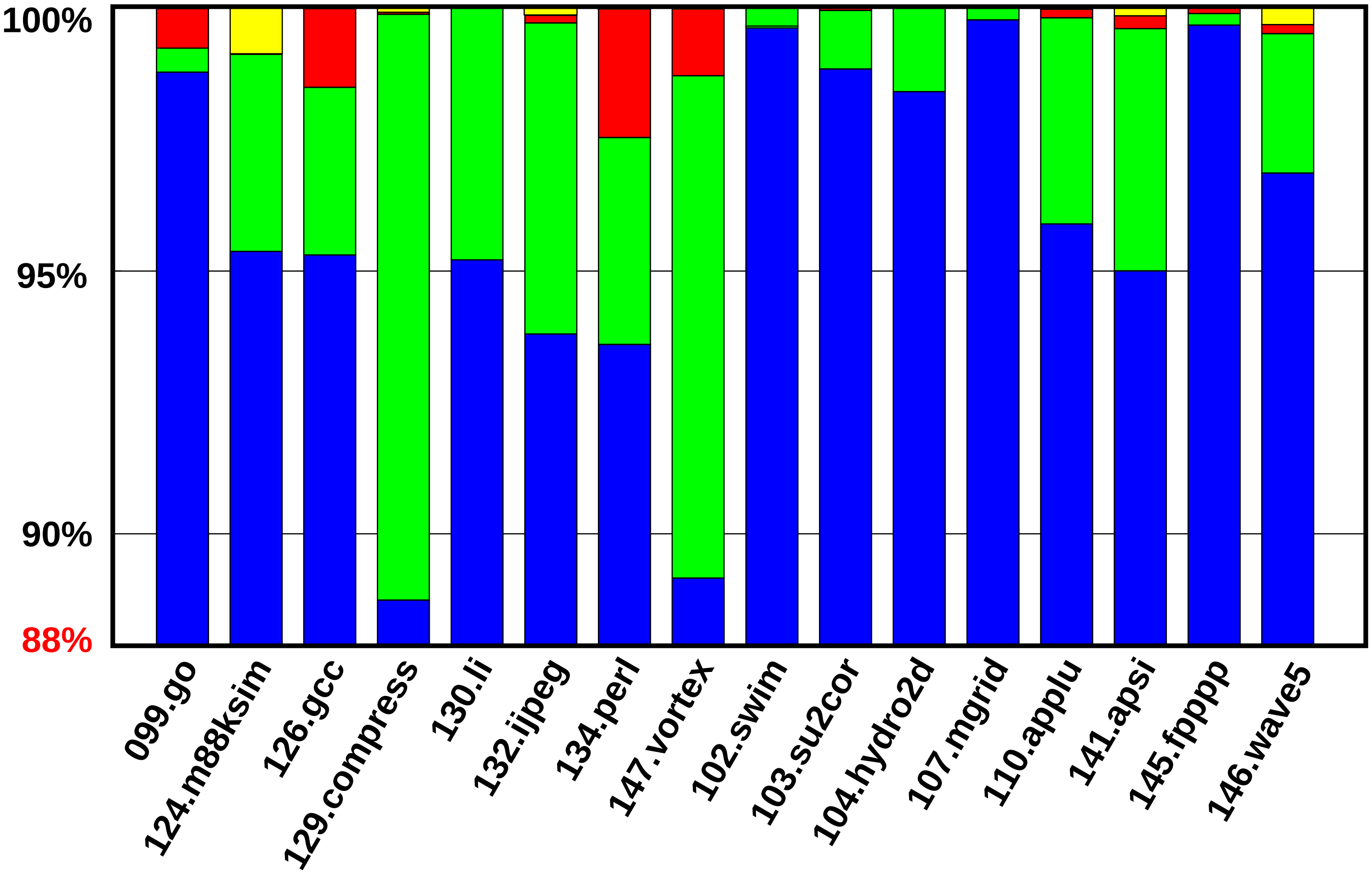
1. Speculation Wins
2. Selective may do worse than Naive
3. Synchronization Robust

Need: What to Speculate + How Long to Wait

Evaluation - Parameters

- 64 entries
- Fully associative
- Single sync bit per stage
- Predictor:
  - 3-bit counter based (threshold of 3)
  - minimal control path information

Dependence Prediction Accuracy



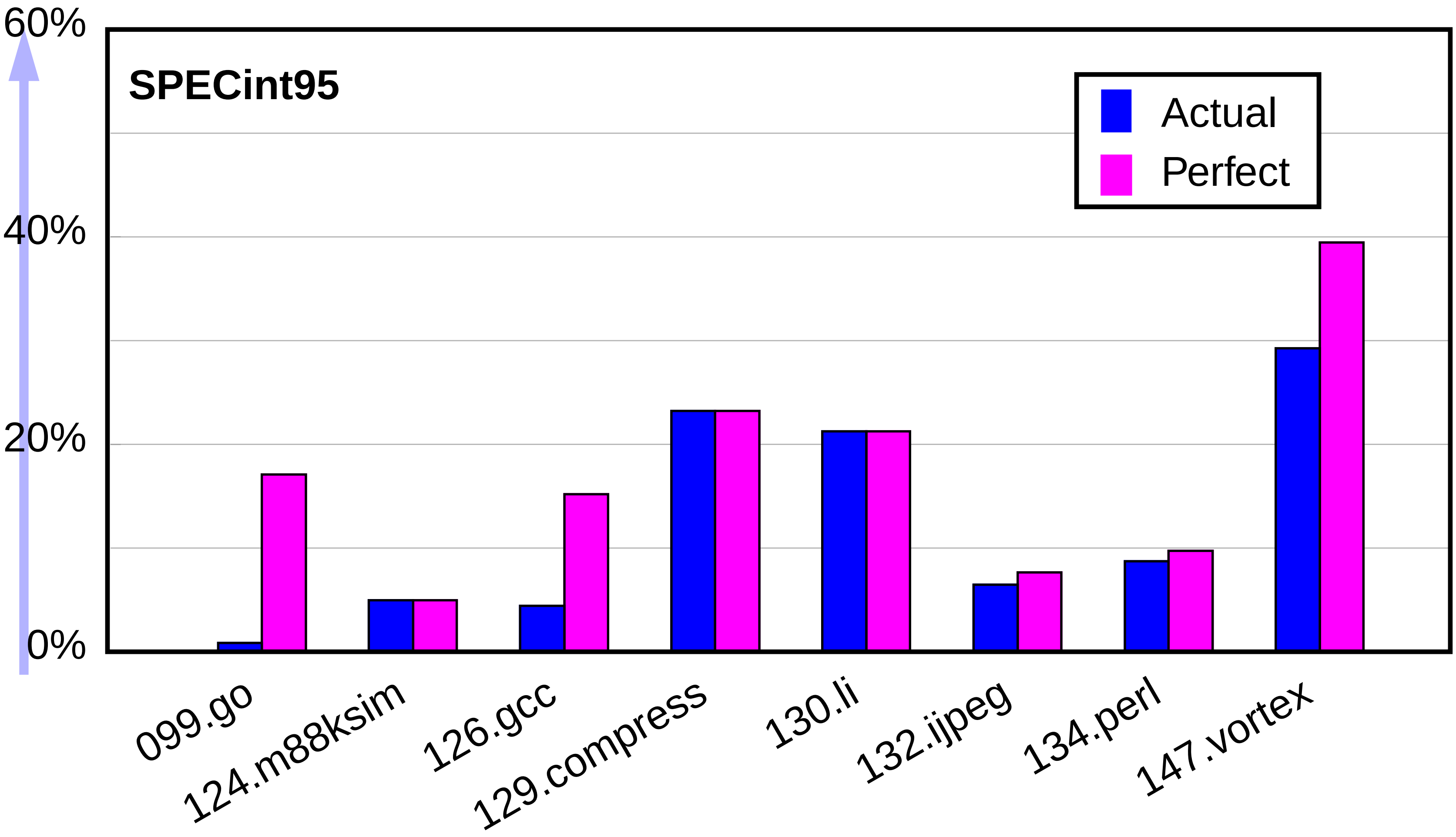
Predicted/Actual

🟦 N/N 🟩 Y/Y 🟥 N/Y 🟨 Y/N

Correct: 🟦 + 🟩

Incorrect: 🟥 + 🟨

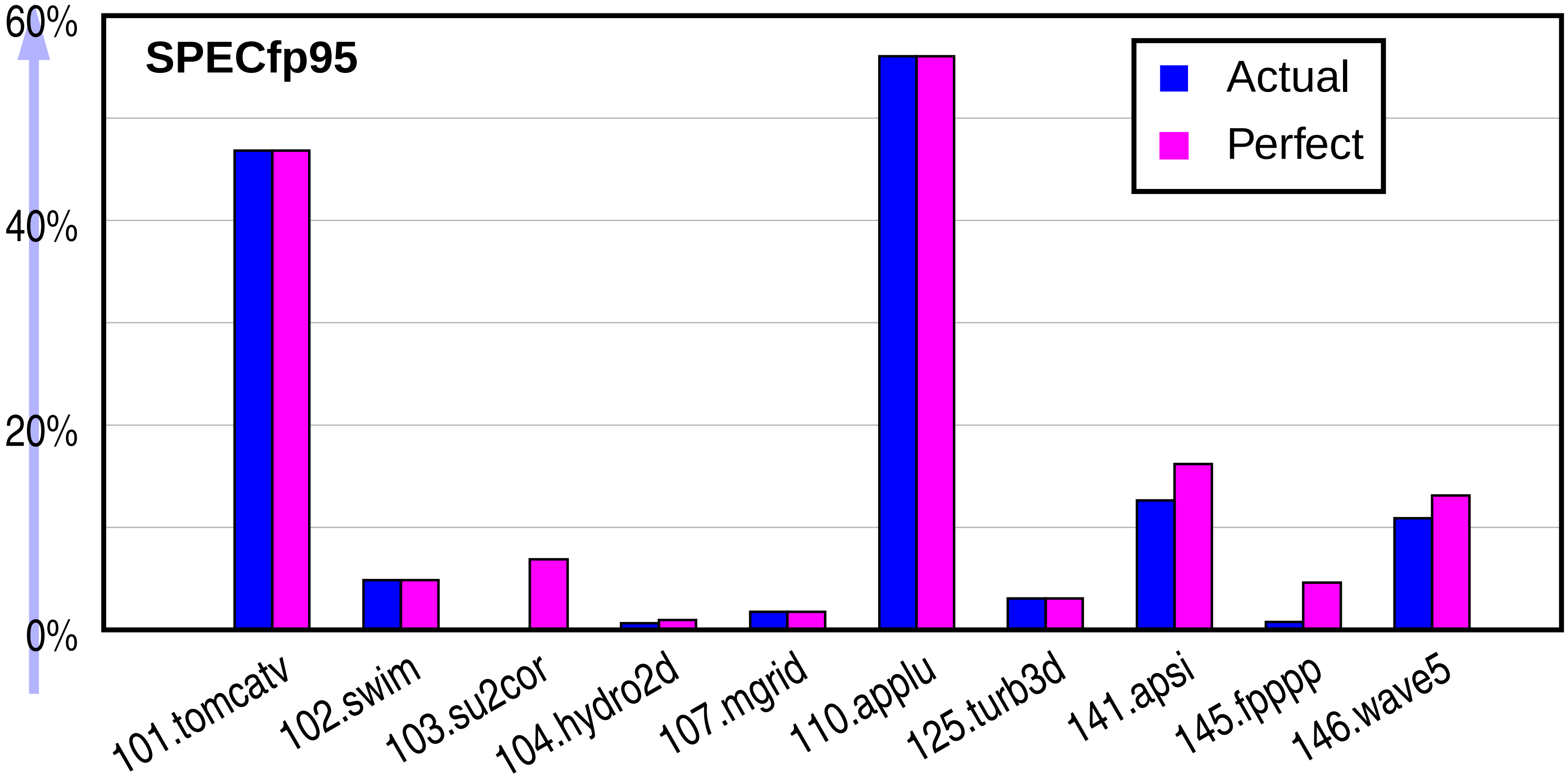
Speedup - SPECint95



- Speedups: relative to Naive speculation

Often close to perfect

Speedup - SPECfp95



- Speedups: relative to Naive speculation

Often close to perfect

Summary

Unresolved data dependences obscure parallelism

Solution: Dependence Speculation

State-of-the-art: Naive Speculation

Wider windows

High opportunity for speculation

Naive Speculation ➜ net penalty significant

Ideally

Load waits for store, only if dependent

Summary

Dependence **prediction** and **synchronization**

Dependences are predictable

Temporal Locality + Small Working Set

Overall

Mis-speculation rate: order of magnitude reduction

Performance close to perfect mechanism

improvements of up to 55%

Why not Memory Dependence Speculation?

Applicable to registers too...

Can the Compiler do it?

YES! However:

(1). Identifying dependences

- Not all dependences need be synchronized
- Dependence behavior may vary:
  - over time
  - with data set

(2). How to synchronize

- Mechanism is needed. Likely, fine-grain.
- Allocation?
  - Static Names: have to convert all dep/s. to distance 1

Let the compiler do its best

Rely on our mechanism for all other cases

SuperScalar Environment?

Loads w/ dependences:

@ 64 instructions: 1 of 3

@ 256 instructions: 1 of 2

- Probability of mis-speculation is high

Many speculative loads:

Pick the right one at the right time

Selective Invalidation:

Opportunity Cost

Implementation?

Split Window? Multiscalar, Hydra

Isn't this I-Structures?

I-Structures:

Program & Language support is needed

Write Once Semantics

Dependence Speculation/Synchronization

No program support

Built on the fly

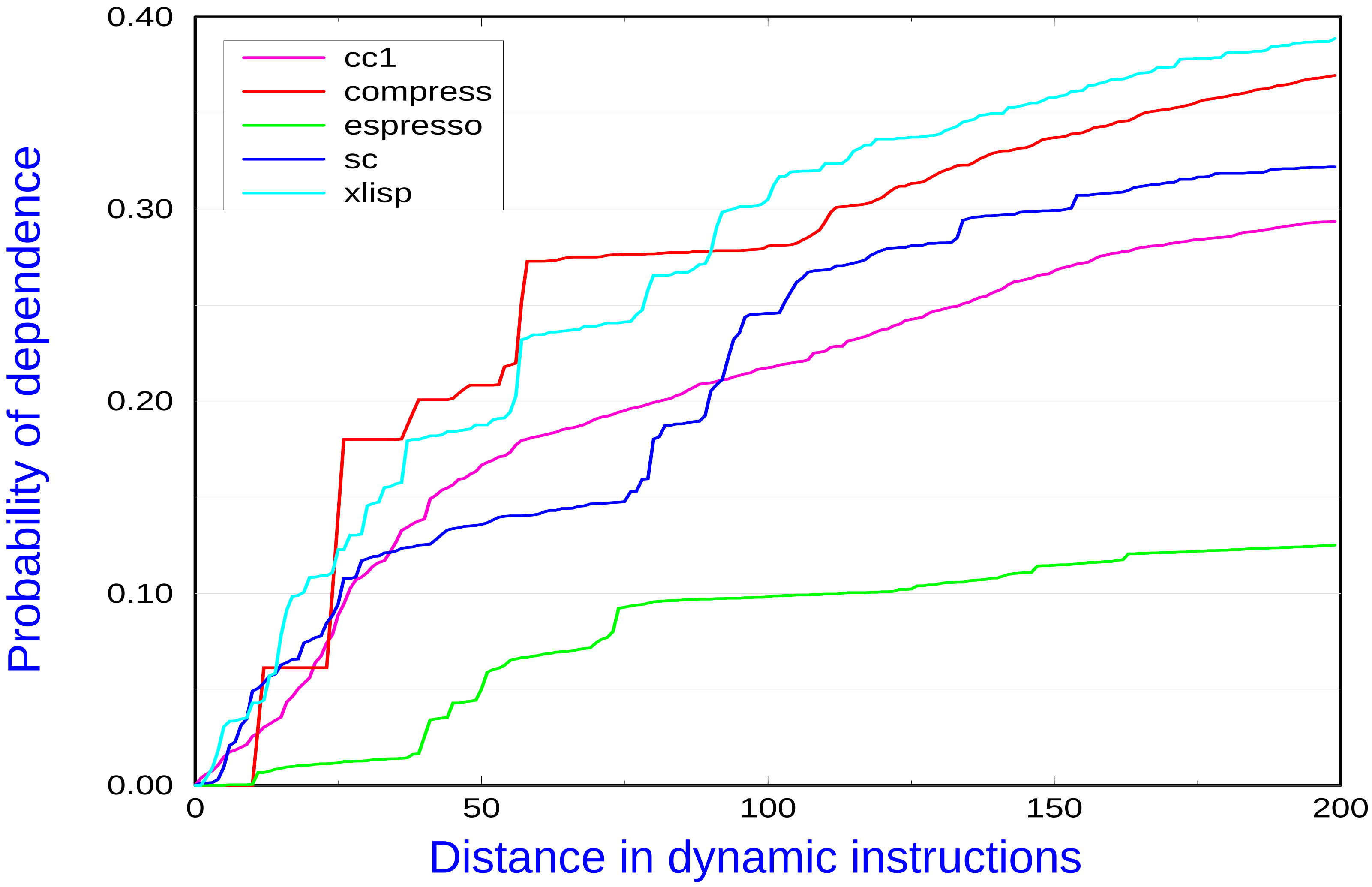
Speculative

No correctness issues

Only for some of the dependences

No write once semantics

Dependences vs. Window Size



Frequency of loads with Dependences within the Window

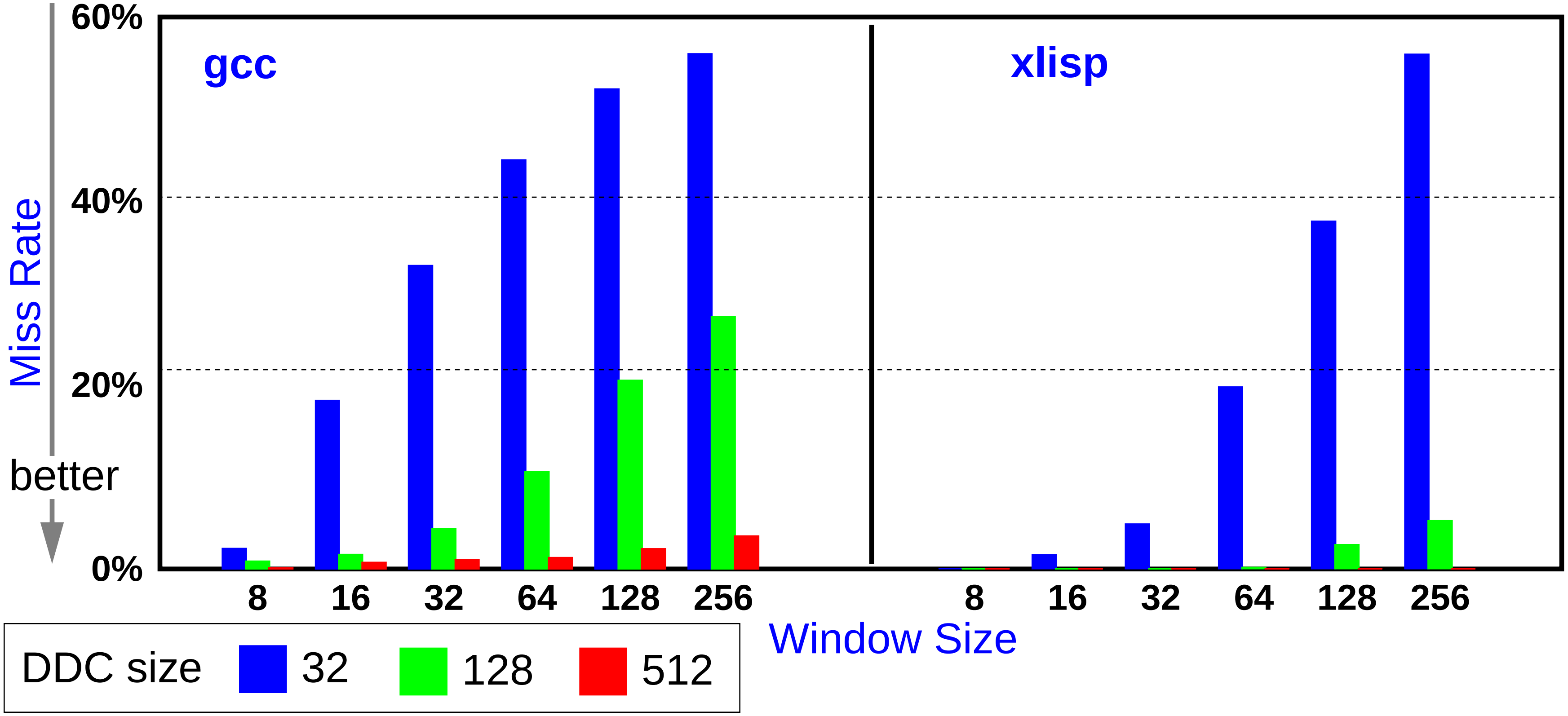
Are Dependences Predictable?

Dependences as (Load PC, Store PC):

(1). Temporal Locality (2). Small working set

Data Dependence Cache (DDC) to demonstrate:

Records the n most recent dependences



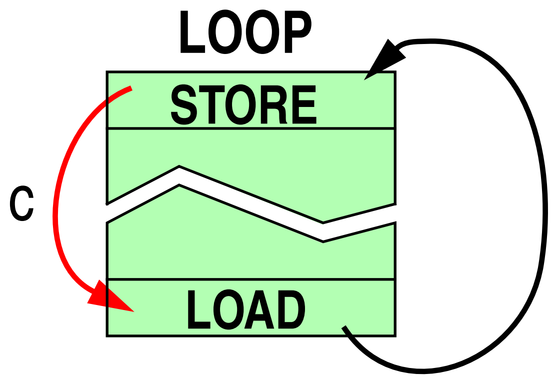
Impact of Window Size

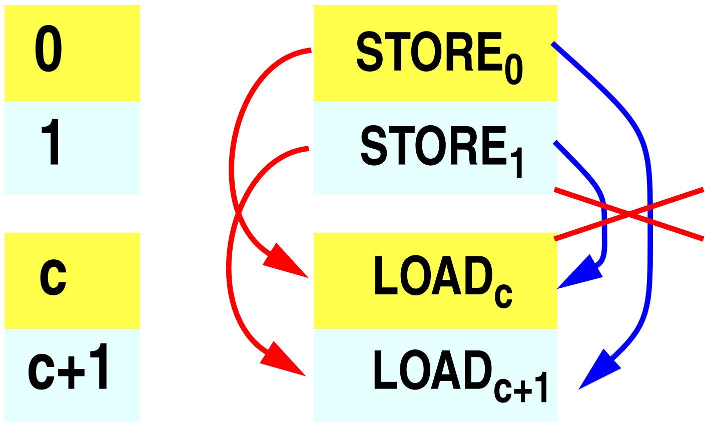
Window	Dependences	Parallelism	Net Penalty w/ Naive
Small	infrequent	not much	insignificant
Wider	more frequent	more	significant

Can do better than Naive Speculation

Decide What to Speculate and When

Multiple Instances of the Same Dependence





1 **Identification: (Load PC, Store PC) not enough**

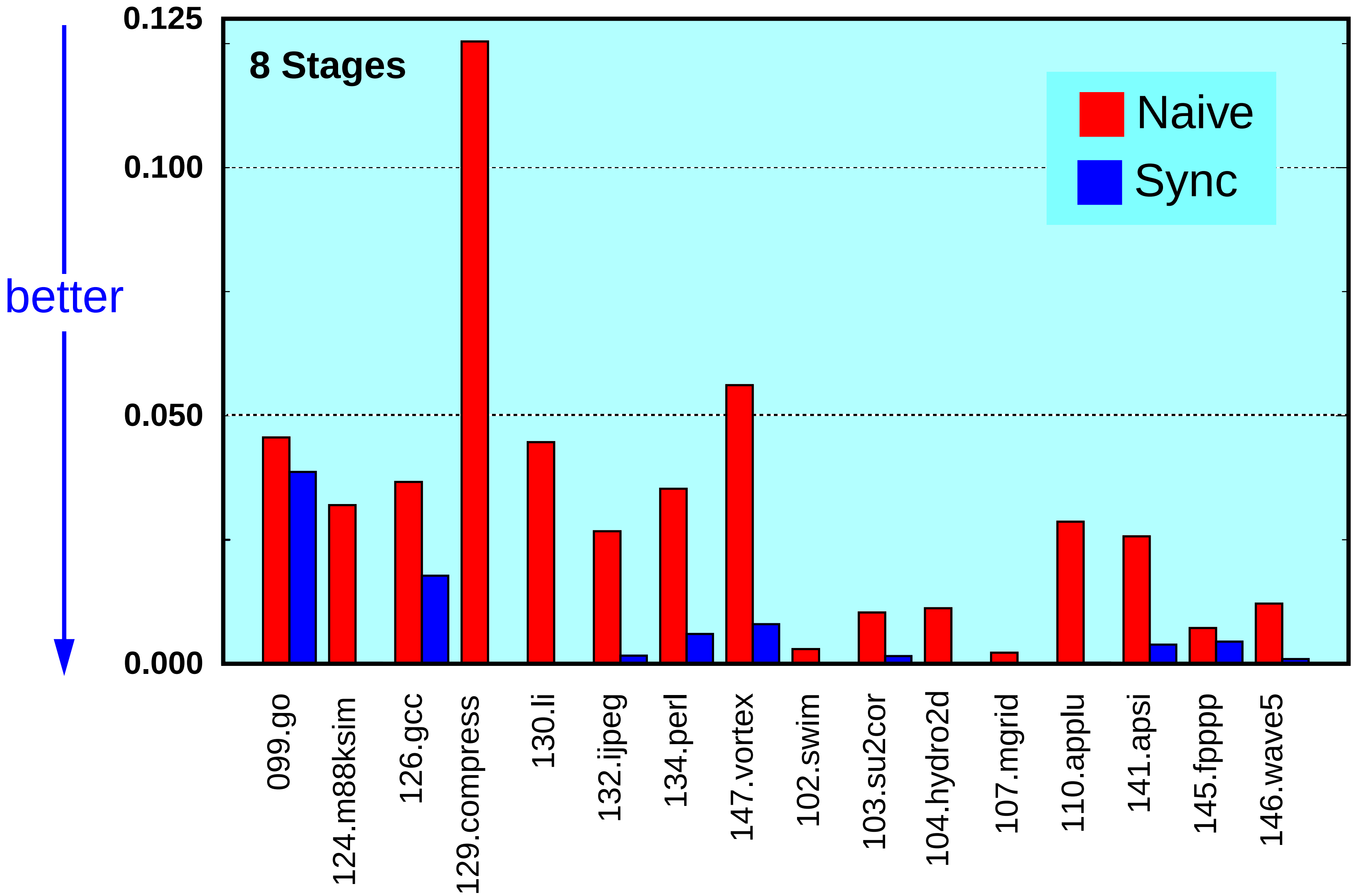
In addition:
- (1). Data Address, or
- (2). Dependence Distance

Analogous to static linear recurrence analysis

2 **May Need:**

Multiple synchronization entries per dependence

Mis-speculation Rates


better
8 Stages
Naive
Sync
099.go
124.m88ksim
126.gcc
129.compress
130.li
132.ijpeg
134.perl
147.vortex
102.swim
103.su2cor
104.hydro2d
107.mgrid
110.applu
141.apsi
145.fpppp
146.wave5