# Effective Jump-Pointer Prefetching for Linked Data Structures

Amir Roth and Guri Sohi

amir,sohi@cs.wisc.edu

Computer Sciences Department
University of Wisconsin-Madison

# Introduction

Problem: Pointer chasing latency

- Especially long latency

Angle: Overlap pointer loads with one another

- Challenge: overcome explicit serialization

**New technique: Jump Pointer Prefetching**

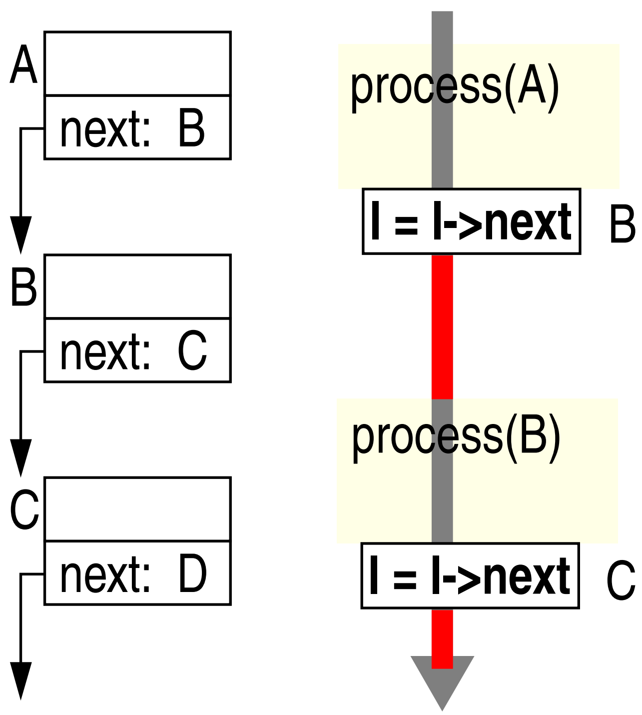
- Creates parallelism
- Hides arbitrary latency
- Choice of implementation: software, hardware, cooperative

# Problem Overview

## Linked Data Structure Traversal:

```
for (l = A; l != NULL; l = l->next)
      process(l);
```

Memory / Execution




What happens:

- Do some w ork with A
- Get addr ess of B from A
- Access B (wait)
- Repeat

**Pointer loads:**

- Serialized
- Hard to address-predict

  → **Hard to overlap w/ each other**

**Jump pointer prefetching:**
**Overlap pointer loads with each other anyway!**

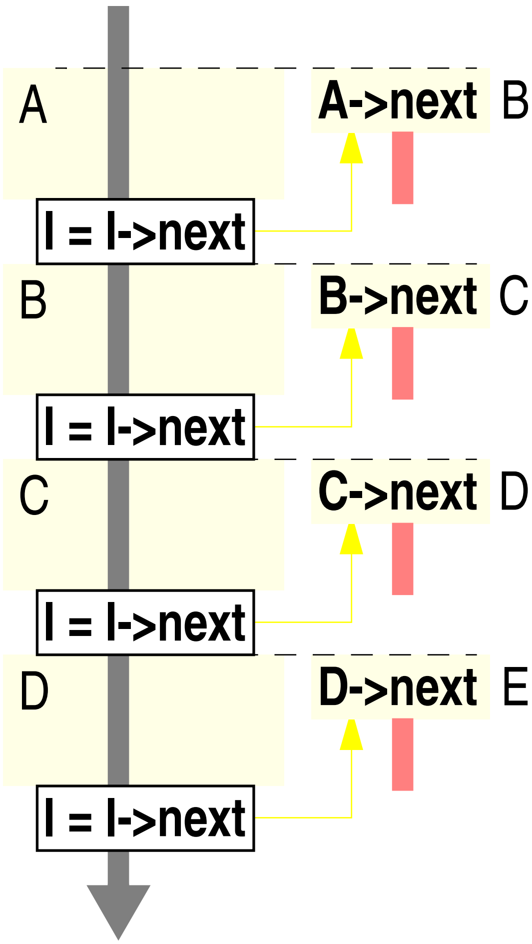
# Talk Outline

- ~~Intr oduction~~
- T wo scenarios for pointer load overlapping:
  - Unnecessary: scheduling is enough
  - Necessary: use jump pointers
- Jump P ointer Prefetching
  - Concepts
  - Implementations: software, hardware, or cooperative
  - Hardware mechanics
- Number s
- Summ ary

# Scenario I: Pointer Load Overlapping is Unnecessary

## Deciding Factor: Pointer load latency vs. Iteration Work

### Work > Latency



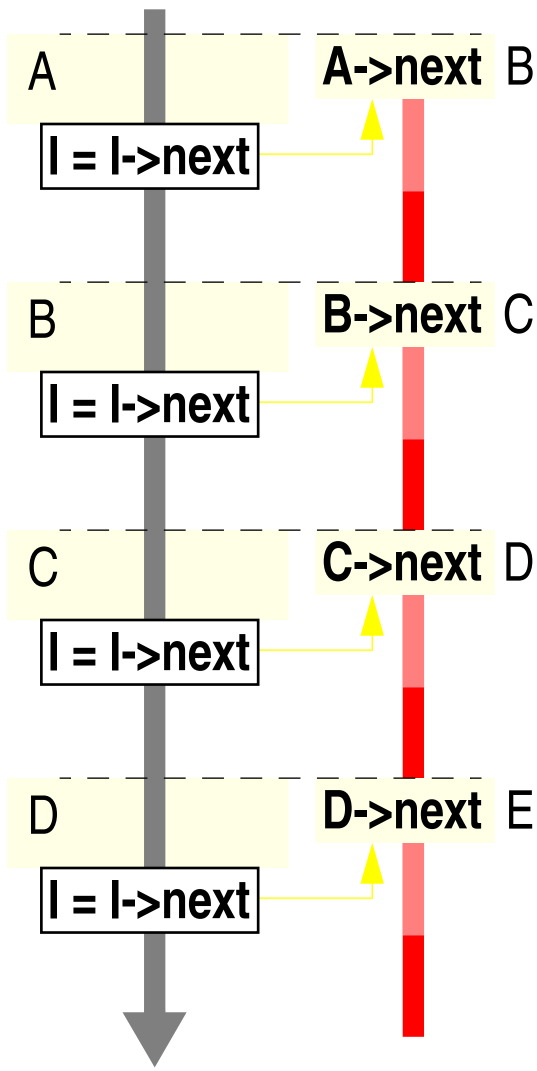

Schedule load early

- Get addr ess of B from A
- \+ Access B, work with A in parallel
- Repeat

**Dependent prefetching:**

- \+ Scheduling, not address prediction
- Compiler
- OOO issue (up to windo w size)
- Another mechanism
- \- Limited latency hiding (1 iteration)

# Scenario II: Pointer Load Overlapping is Necessary




**Latency > Work**

Pointer load latency = 2 iterations

- Scheduling (1 iteration) not enough
- Must o verlap pointer loads with one another

Functionality of a solution:

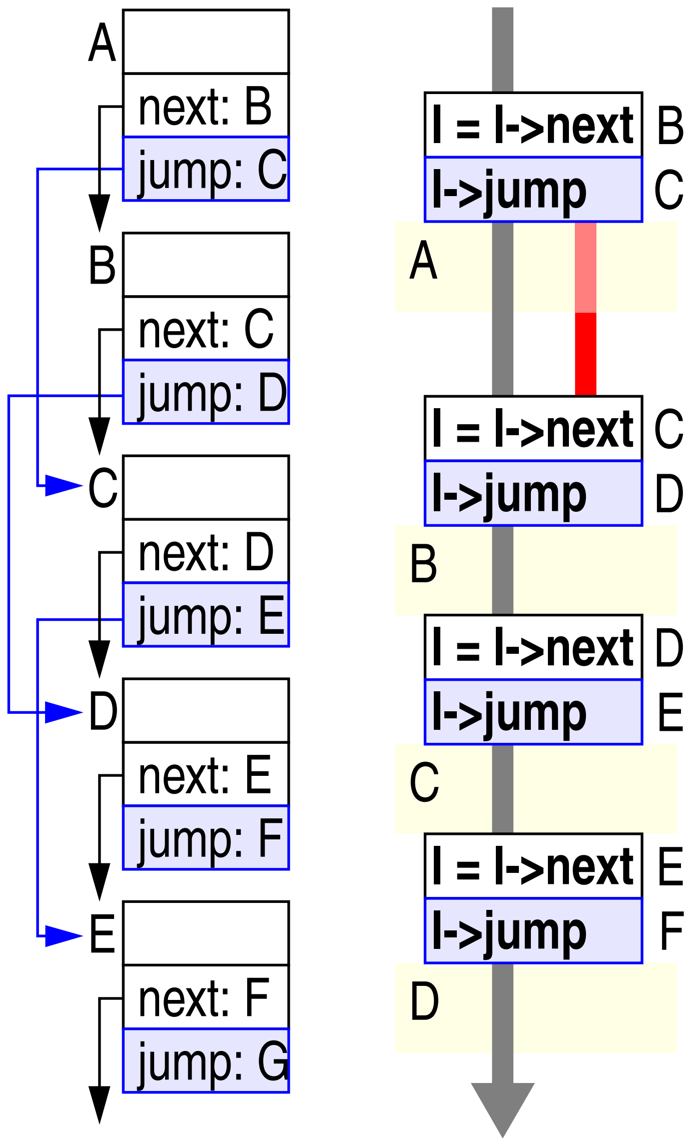
- Addr ess of node 2 hops ahead

+ "Create" access parallelism

- Remember: no address prediction

**Use Address Lookup Mechanism!**

## Jump Pointer Prefetching




### Jump Pointers:

- Implement address lookup
- Added to every node
- Situated at home, point to target
- Interval = target - home (2 here)

What happens now:

- From A, get addresses of B,C

\+ In parallel: access B, C, work on A

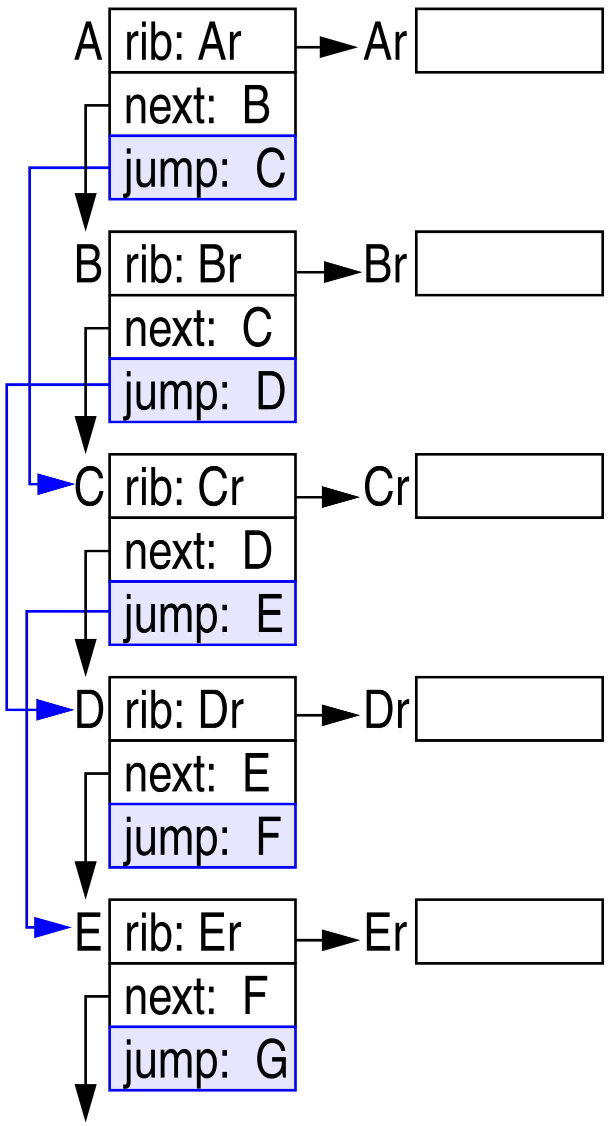
\+ Always access 2 iterations ahead

### Jump pointer prefetches:

\+ Tune interval to hide latency

\- Overheads: storage, instructions

# A More Realistic Example



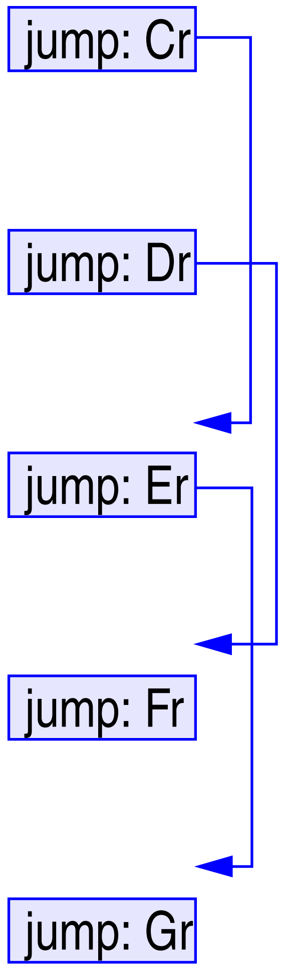

So far: simple structures

- List, tr ee, etc. (one-level)

More complex: “backbone+ribs”

- List of r ecord pointers
- Jump pointer s for “backbone”
- Ho w to tolerate “rib” latency?

One possibility: do nothing

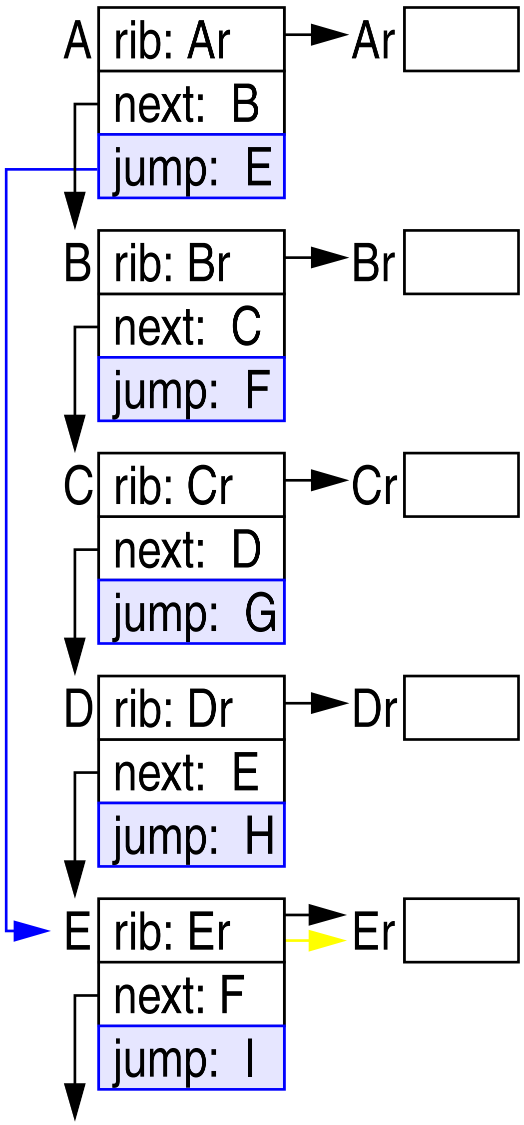



**Full jumping:** jump pointers

+ Full latency tolerance
- More overhead

• W e can do better

# Combining Dependent and Jump Pointer Prefetches




Another possiblity:

- Launch dependent prefetches from completed jump pointer prefetches
- Gotcha: the tw o prefetches are serial

  → Jump pointer must hide two loads

  → Increase interval to 4 iterations

**Chain jumping:**

+ Same latency hiding as full jumping
+ Less jump pointer overhead

**Trade L2 jump pointers for Dependent Prefetches + L1 Interval**

# Implementation Space

## Software overhead vs. Hardware cost

### Dependent prefetching:

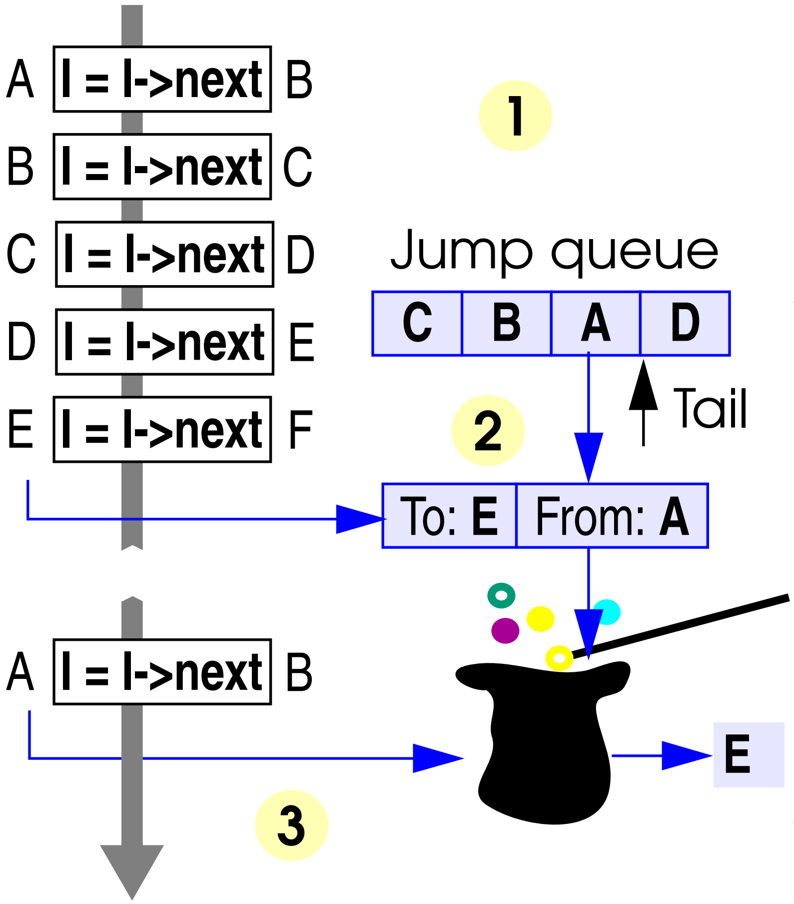
- S/W : Greedy Compiler-Based (Luk&Mowry ASPLOS96)
- H/W : Dependence-Based (Roth,Moshovos&Sohi ASPLOS98)

### Jump pointer prefetching:

- S/W : History-Pointer (L&M ASPLOS96, this paper)
- H/W : (this paper)

| | | Dependent Prefetches | |
|---|---|---|---|
| | | **Software** | **Hardware** |
| **Jump Pointer Prefetches** | **Software** | Software | Cooperative |
| | **Hardware** | | Hardware |

# Hardware Jump Pointer Prefetching: Mechanics




**Step 1:** find “backbone” loads

- learn dependence (ASPLOS98)

**Step 2:** create jump pointers

- Jump queue: stores N recent addresses (N = interval)
- Create jump pointer:
  - Home = tail of queue
  - Target = current node

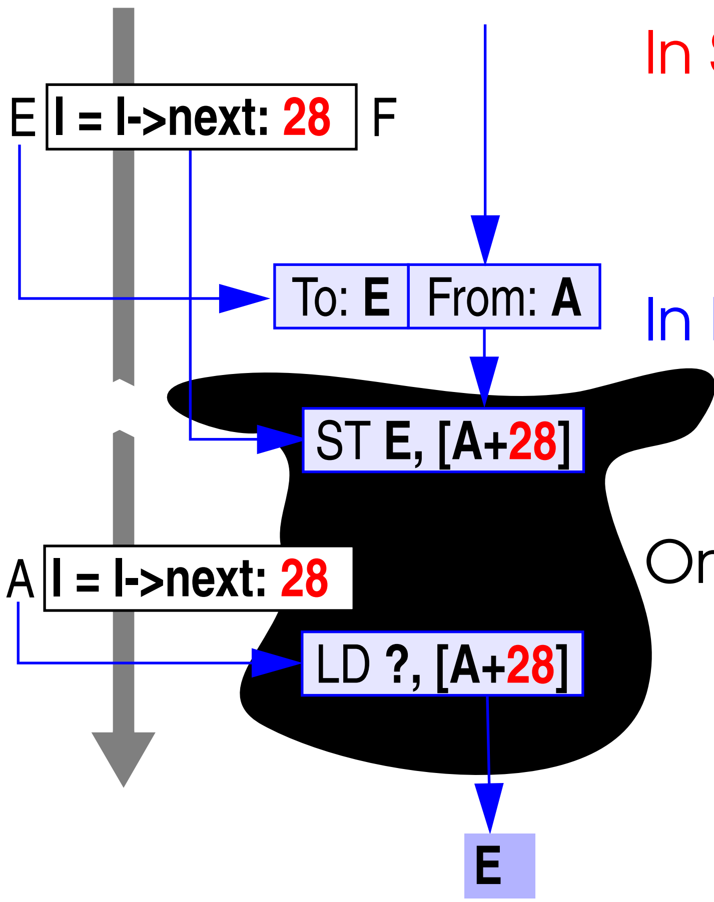
**Step 3:** lookup/prefetch

**Inside the hat:** next slide

**Mechanisms 2 + 3 in Software = Overhead**

## Hardware Jump Pointer Prefetching: Mechanics II

### Inside the Hat: Where do we put jump pointers?




In Software: with the home node

+ Natural lookup from home node
+ Storage often free (malloc padding)

In Hardware: same thing

- Ho w to tell where padding is?

One solution (ours):

- Softw are hints where padding is
- Har dware uses the padding
  - Ex: Padding @ base address + 28

Other solutions/storage possible

## Experiments

**Benchmarks:** Olden (pointer-intensive)

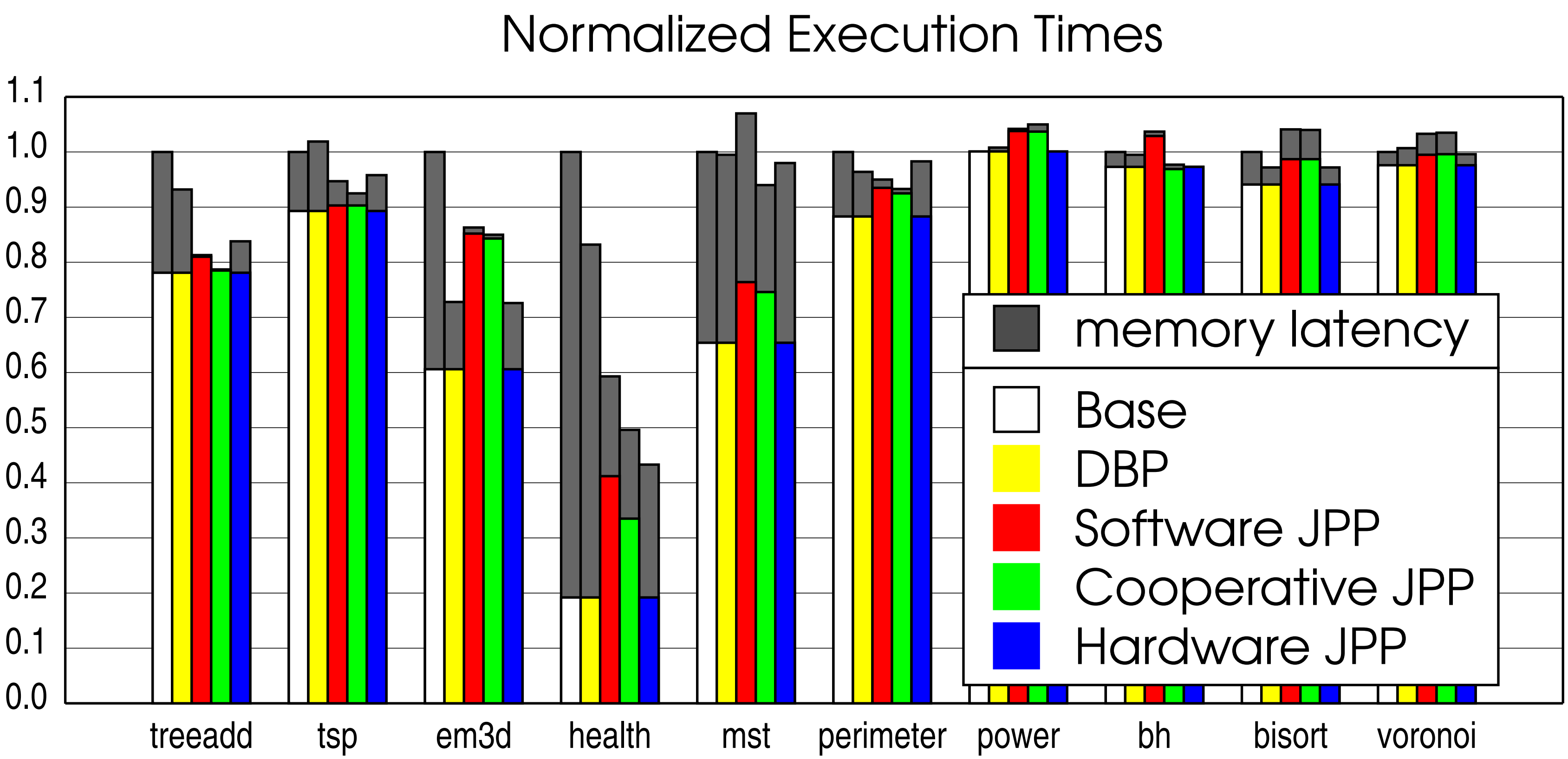
- Softw are jump pointer components inserted manually

**Simulations:** SimpleScalar

- 4-wide super scalar, OOO-issue, 64 instructions in-flight
- 5 stage pipeline
- 64 KB, 32B line, dual-por ted L1 D-Cache, 1 cycle access
- 512KB, 64B line, L2 U-Cache, 10 c ycle access
- 70 c ycle memory latency
- 8 outstanding misses
- 64bit buses (contention modeled)

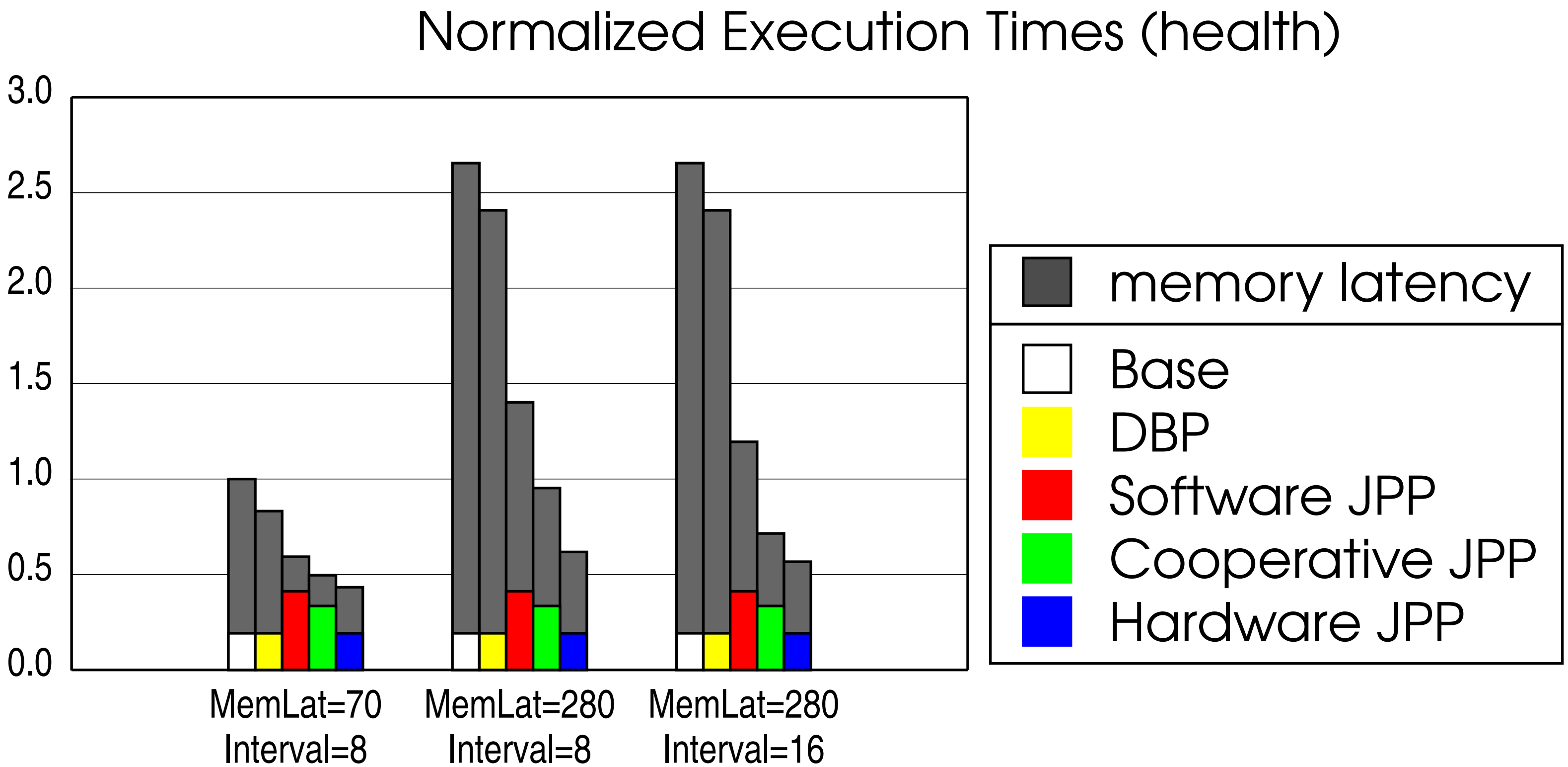
- Dependence-based prefetching: 256 dependences
- Jump-pointer prefetching: 32 4-interval jump queues

# Numbers




| | | | |
|---|---|---|---|
| Memory Latency | -72% | **-83%** | -55% |
| Compute Time Overhead | +10% | +7% | **0%** |
| Execution Time | -15% | -20% | **-22%** |

# Tolerating Longer Latencies




Highlights:

+ MemLat=280 + Hardware JPP: 40% faster than MemLat=70
- Cooper ative JPP: normally -50% execution time
    - MemLat=280, Interval=8: -5%
    + MemLat=280, Interval=16: -30%

# Summary

## Linked Data Structures

- Unpr edictable addresses + Serialized latencies
- Scheduling (DBP) w orks when Latency < Iteration size

## Jump Pointer Prefetching

+ Works even when Latency > Iteration size
+ Creates access parallelism where there was none
+ Tunable for long latencies
+ Synergy with scheduling reduces overhead and cost

## Summary II

Three implementations:

| | Software | Cooperative | Hardware |
|---|---|---|---|
| Software Overhead | high | low | none(+) |
| Hardware Cost | none(+) | low | medium |
| Performance | good | very good | best(+) |

**Pointer chasing problem: Solved?**

# Loose Ends

Memory bandwidth requirements

- Jump pointer stor es always hit
- Jump pointer lookups almost alw ays hit
- Jump pointer pr efetches very accurate

+ Very low (see paper)

Trees and graphs

+ Queue mechanism still works

Highly dynamic data structures

+ Speedup degrades gracefully

Interaction with loop unrolling

+ Can be made transparent