New Methods for Exploiting Program Structure and Behavior in Computer Architecture

Guri Sohi

sohi@cs.wisc.edu

Computer Sciences Department

University of Wisconsin-Madison

Contributors: Andreas Moshovos, Avinash Sodani, Amir Roth, Andy Glew, Craig Zilles, Harit Modi, Adam Butts

New Basis for Architecture/Microarchitecture

- Programs have structure (relationships amongst operations)
- Program structure **causes** the **observed** program behavior
- Current microarchitectural mechanisms based upon **observed** program behavior
  - spatial locality, temporal locality, data access patterns
  - **secondary** information
- Can we exploit **primary** information, i.e., **causal relationships** in architecture/microarchitecture?
  - program structure information is primary information

Program Structure Information: An Example

Example: branch prediction

- Early: Branches predicted in isolation
- Major Leap: Branch correlation
- Now: Golden age of branch prediction

Great insight? Different branches related

programs have structure!

Example II: memory hierarchy design

- Early: Program structure not taken into account
- Now: Still not. Why not?
- Major leap: Coming soon

Secondary Information: Not Really Program Structure

Branch correlation is a **secondary** method

Secondary information: instruction inputs/outputs

- Examples: branch outcomes, addresses, values
- Properties: spatial/temporal locality, patterns

Current mechanisms almost exclusively based on secondary information and its properties

Problem I: weak properties may not hold all the time

Problem II: Hard to figure out what's going on sometimes

Primary Information: Real Program Structure

"Programs have structure" is too obvious

Primary information: relationships amongst operations

- Examples: control dependences, data dependences
- Properties:
  - **temporal stability:** program is invariant (strong)
  - **causality:** causes all observed secondary behavior

We have program structure handy! Can we exploit it?

Application: Scheduling OOO Memory Operations

Problem: OOO execution of memory operations can cause misspeculations

Solution1: use prediction to stall offending loads

- no program structure information required
- not very effective

Solution 2: determine store-load dependences ans use to synchronize speculation

- use program structure
- very effective

More (Moshovos, et. al., 1997, Chrysos and Emer 1998)

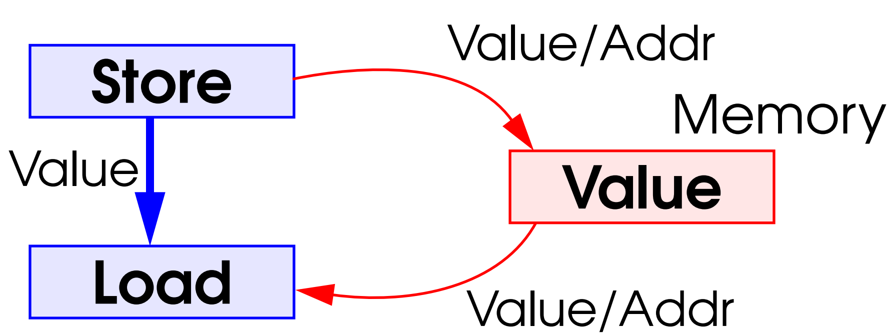
Application: Fast Communication Through Memory

Problem: Accessing memory is inherently slow, ambiguous

Program structure: Memory is a communication device for passing values from stores to loads. Not random: only certain stores to certain loads

Speculative Memory Cloaking

Link stores to loads explicitly, pass value along link

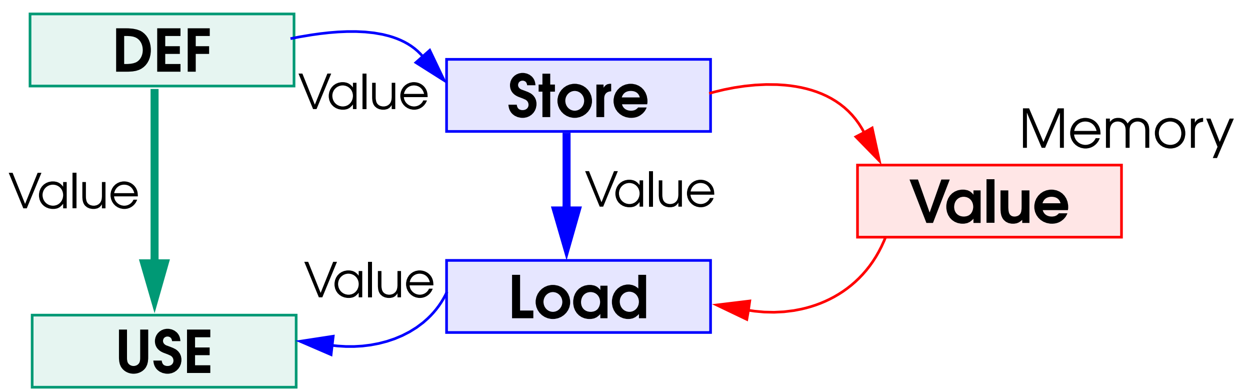


Fast Communication II

Program structure: **Loads and stores are used for passing values from one instruction (DEF) to another (USE).**
Via memory? (maybe not, can do it directly)

Speculative Memory Bypassing

Collapse DEF-store, store-load, load-USE links into a direct DEF-USE link



More on Cloaking & Bypassing: (Moshovos & Sohi, MICRO-30)

Fast communication III: Shared memory MP's

Problem: Optimize CC protocols for sharing patterns

So far: Detect patterns using address attributes

- Track state proportional in size to data (big)
- Little predictive power

Program structure: Sharing pattern property of program, not data

Detect using instruction relationships

- Track state proportional in size to program (small)
- Great predictive power, works much better

More: (Kaxiras, PhD Thesis)

Application: Prefetching Linked Data Structures

Problem: Linked data structures

- Chains of long-latency loads limit parallelism
- Hard to predict addresses for prefetching

Program structure: **(l = list; l; l = l->next)**

Traversal uses few static loads, few relationships

Learn structure and pre-execute speculatively:

- No explicit address prediction, predict loads and execute
- All we need to remember: **l = l->next**
- Compresses chains, removes aritificial issue delays

More: (Roth, Moshovos & Sohi, ASPLOS-8)

Application: Branch Pre-execution

Program structure: Branches more closely related to instructions that feed them than to other branches

Learn dependences, use to pre-compute branches

- Early: avoid mis-speculation
- A little late: reduce penalty

Proof of concept: Virtual Function Calls

- Hard to predict: Multiple targets a problem
- Easy to pre-compute: Linear dependence chains
- Cuts misspeculation by ~80%

More: (Roth, Moshovos & Sohi, unpublished)

Research Issues

For a particular optimization

- What program structure information is required?
- How do we represent this information?
- How do we collect and manage this information?

More broadly

- Where can we apply program structure?
- Is there a larger framework?
- What is general purpose program structure?
- Implementations?

Reseach Issues II: Implementations

Hardware only

- Works well
- Current focus

Software to hardware

- Compiler has all kinds of program information
- How to express it? Instruction-like things awkward
- Where and how much to express?
- Will go here when we have better understanding

Software/hardware hybrids

Summary

Many other applications:

- Instruction fetch
- Memory hierarchy design
- Scheduling
- More

Program structure information makes it all work!

- **Compact:** Handle information size of program not of data
- **Stable:** Always holds
- **Causal:** Pre-computation is the ultimate predictor

Here comes a whole new wave of innovation...