

Compile-Time Data Speculation

Wen-mei Hwu

Dept. of ECE

University of Illinois
at Urbana-Champaign

Types of Data Speculation

- Compile-time Data Speculation
 - executing an instruction before knowing input is up to date
- Dependence Speculation
 - moving a load and its use above a potentially conflicting store
 - must correct the load and the uses thus moved in case of conflict

Types of Data Speculation

- Value Speculation
 - generate and use the result for an instruction before it is ready for execution
 - use past value
 - prediction using trends

Load/Store Dependence Conditions

STORE (addr1) <- R1

LOAD R2 <- (addr2)

- Always access the same location
 - redundant load elimination
- Never access the same location
 - load/store reordering

Load/Store Dependence Conditions

- May access the same location
 - They sometimes access the same location
 - No conclusion can be reached
 - How often do they access the same location?

Preload Register Update

- Deals with any dependence condition
- Allows load but not use to move above the store
- Tolerates first level memory latency

Memory Conflict Buffer

- Deals with very frequent or very infrequent dependence conditions
- Allows uses of the load value to move above the store
- Eliminates effect of memory dependence to exploit parallelism

Data Spec. Motivations

- Lower entrance level requirement of static disambiguation
- Achieve performance earlier and more consistently for
 - new languages, e.g., C -> C++
 - new applications and libraries
 - e.g., object oriented and dynamically linked
- Allow performance to increase smoothly as static disambiguation matures

Data Spec. Applications

- Code optimization
 - invariant load and/or use code motion in the presence of ambiguous stores
 - redundant load and/or use elimination in the presence of ambiguous stores
- Code scheduling
 - load-store and/or use-store reordering in the presence of ambiguous stores

Runtime Memory Disambiguation

- See figure

Key Issues

- Number of explicit comparisons
- Amount of conflict correction code
- Varying access sizes
- Potential payoff in performance vs. overhead and resource consumption

Access Size Problem

- See figure

Preload Register Update

- Register State Bits (See figure)
 - P: preload register, coherence mechanism on
 - F: freeze state for I/O ports
 - R: ready bit for interlocking mechanism
 - T: trap bit for exception detection
 - Preload Type: size and data alignment

Example Pipeline Timing and Code Example

- See figures

PRU Subset Design

- Hardware Changes
 - Additional address register GRP field required
 - Number of active preload registers is reduced
 - Number of concurrent comparisons reduced

PRU Subset Operation

- Save preload address in an empty address register
- Set V bit and GRP entry
- Use of preload destination register will turn off coherence

Address Register Replacement

- An old entry is overwritten
- Set F bit of GR associated with the replaced entry
- Replaced register is reloaded with address field at time of use

Compiler Terms

- Basic block where the preload originated (HOME)
- Operation that uses the preload destination register (USE)
- Closest ambiguous store before (CASB):

Compiler Terms

- The first ambiguous store before the memory operation
- Closest ambiguous store after (CASA):
 - The first ambiguous store after the memory operation

Dependence Graph

- Remove all previous store dependence to potential preloads
- Insert dependence arc from CASB to USE

Dependence Graph

- Insert dependence arc from first USE to CASA
 - First USE must stay in HOME
 - create a USE if not available

MCB Operations

- Preload inserts addr. in addr. register file
- Store matches addr. against addr. registers
 - If same, set asso. Except. bit
 - Check branches to correction code if conflict bit is set
 - check Rd, label

MCB Hashing Algorithm

- Do not hash 3 LSBs
- Could use bits 3 to $(n+2)$ to select MCB line
 - Concerned about strided accesses might cause collisions

MCB Hashing Algorithm

- Employ a permutation-based hashing scheme
 - In concept, similar to matrix multiplication
 - In application, simple XOR of incoming address bits

Types of MCB Conflicts

- Hashing MCB design introduced false conflicts
- Three types of conflict which can occur
 - True conflicts
 - False load-store conflicts
 - False load-load conflicts

Need for Silent Instructions

- Use of preload value may cause exception
- Must mark these instructions as silent
- Precise detection of exception using sentinels

Compiler Algorithm

- Build dependence graph
- Insert checks for all loads
- Remove dependences from ambiguous store/load pairs
- Schedule, either mark loads to preloads or remove unnecessary checks

Compiler Algorithm

- Insert required correction code
- Register renaming to ensure correction code source operands are preserved

Compiler Algorithm Example

- See Fig.

Register Renaming

- See fig.

Correction Code

- Increased register pressure
 - Insert moves and save-restore if necessary
 - More intelligent register allocation

Correction Code

- Code expansion and I-cache effects
 - Error handling routine invocation is expected to be small.
- Error handling routine invocation
 - control speculation with static disambiguation

Register Pressure Opt.

- Only preload, store, and check affect MCB
- Data and addr registers are distinct
 - If a preload destination register is recycled to hold the result of a non-preload instruction, the contents of the corresponding address registers remain active.

Example Live Range

- See fig.

Example Interference Graph

- See fig.

