

Portland State University  
ECE 587/687

# Trace Cache

# Trace Cache

- High performance OoO processors require
  - ◆ Increased instruction fetch bandwidth
  - ◆ Low instruction fetch latency
- Paper proposes supplementing a conventional instruction cache with a trace cache
- Main idea: Non-contiguous instructions appear contiguous in the trace cache

# Issues with Instruction Fetch

- Performance issues emerging as processor issue rates increase:
  - ◆ Branch throughput
    - Predicting multiple branches, including taken branches every cycle
  - ◆ Non-contiguous instructions and alignment
    - Remember Instruction Cache Fragmentation (Superscalar processor lecture)
  - ◆ Fetch unit latency
    - Throughput and complexity may increase latency
- Basic block stats: Paper Table 1

# Possible Solutions

- One possible solution: allow fetch from multiple non-contiguous basic blocks
  - ◆ Multiple addresses have to be generated before fetch begins:
    - Implies a level of indirection
    - Additional fetch stage latency
  - ◆ Multi-ported or interleaved ICache is required
  - ◆ Instruction merging and alignment increase fetch latency
- Trace cache avoids these problems using redundant instruction storage

# Trace Cache: Concept

- Basic idea:
  - ◆ Conventional instruction cache holds instructions in static program order
  - ◆ Trace cache holds instructions in dynamic program order
- High-level view: Paper Figure 2
- Why would it work?
  - Temporal locality
  - Branches' predictable behavior
- Discuss: Storage overhead

# Core Fetch Unit

- The core fetch unit
  - ◆ Interleaved sequential: 2 consecutive blocks can be accessed in the same cycle
  - ◆ Fetch up to the limit or to the next taken branch
  - ◆ Limit of 16 instructions and 3 branches
  - ◆ 16-way interleaved BTB
  - ◆ Multiple branch GAg predictor:
    - 14-bit global history register
    - Rearranged PHT organization: 8 state machines per pattern
- Paper: Figure 3
- Control not on critical path, datapath delay minimal

# Trace Cache Organization

- Figure 4 shows the core fetch unit and the trace cache
- Trace cache organization:
  - ◆ Up to 16 instructions wide and 3 branches
  - ◆ Contains a fill buffer, instruction traces and control information

# Trace Cache Control

- Control state:
  - ◆ Valid bit
  - ◆ Tag to identify trace starting address
  - ◆ Branch flags to indicate taken/not-taken direction (except last branch)
  - ◆ Branch mask: #branches in a trace, whether it ends in a branch
  - ◆ Trace fall through address
  - ◆ Trace target address
  - ◆ End of trace marker
- Fill buffer has to stop the trace at indirect branches

# Discussion

- Trace cache design space
  - ◆ Address associativity
  - ◆ Path associativity OR partial trace matching
  - ◆ Indexing method
    - Index with address, match branch prediction with tag
    - Index with both address and branch prediction bits
  - ◆ Fill issues
    - Number of fill buffers
    - Speculative traces
  - ◆ Trace selection: Some committed traces never reused
  - ◆ Victim trace cache
  - ◆ Redundancy

# Simulation and Results

- Simulation processor model
  - ◆ Very large instruction window (2048)
  - ◆ Unlimited renaming
  - ◆ Unlimited functional units
  - ◆ Perfect data cache
  - ◆ Perfect memory disambiguation
- Results shown for trace cache and other techniques of 1, 2 and 3 cycles latency
- Discuss results: Figures 8-13

# Reading Assignment

- J. E. Smith and A. R. Pleszkun, “Implementation of Precise Interrupts in Pipelined Processors,” IEEE Trans. On Computers, 1988 (Review)
- G.S. Sohi and S. Vajapeyam, “Instruction Issue Logic for High-Performance Interruptable Pipelined Processors,” ISCA 1987 (Read)
- Project proposals due on Monday
- HW2 due on Wednesday