Trace Cache: Low Latency, High Bandwidth
Instruction Fetching

*Eric Rotenberg, Steve Bennett, Jim Smith*

Eric:
Computer Science Dept.
Univ. of Wisconsin — Madison

Steve:
Intel Corporation

Jim:
Dept. of Elec. and Comp. Engr.
Univ. of Wisconsin — Madison

Superscalar Processors

• Divide into instruction fetch and execution
• Superscalar trends:
  - larger scheduling window
  - wider dispatch/issue
  - deeper speculation
  - more functional units
• Implications for instruction fetch?
Overview

- Instruction fetch issues
- Trace cache
- Alternatives
- Simulation Results
- Conclusions
Instruction Fetch Issues

- Branch Throughput
  - predict more than one per cycle
- Noncontiguous instruction blocks
  - fetch past taken branches
- Fetch unit latency
  - can’t be ignored when solving others
- “Conventional” issues:
  - instruction cache misses
  - branch prediction accuracy
  - NOT the focus of this study
Noncontiguous Instructions

• Fundamental problem:
  - conventional instruction cache stores programs in their static, compiled order
  - the decoder wants to see the instruction stream in its dynamic order

• Two classes of hardware solutions:
  - deal with conventional cache, construct dynamic order on the fly
  - direct approach: cache instructions in dynamic order

• We propose the direct approach
**Trace Cache High Level**

**DYNAMIC INSTRUCTION STREAM**

- Trace: \( \{ A, \text{taken, taken} \} \)
- Later...
- Trace: \( \{ A, \text{taken, taken} \} \)

**TRACE CACHE**

- 1st basic block
- 2nd basic block
- 3rd basic block (still filling)

**TRACE CACHE**

- \( \{ A, t, t \} \)

Fill new trace from instruction cache

Access existing trace using A and predictions (t, t)

To DECODER
Trace Cache Overview

fetch address → Core I-Fetch (I-Cache)

line fill buffer → Trace Cache

hit logic → mux

prediction logic

→ instructions
Core Fetch Unit Detail

BRANCH TARGET BUFFER
16-way Interleaved

BTB LOGIC

MULTIPLE BRANCH PREDICTOR

"01x"

valid instructions
bit vectors

1st: 0000000000111111
2nd: 1111110000000000

Line Size = 16 Instructions

2-Way Interleaved Instruction Cache

INTERCHANGE, SHIFT, MASK

A

taken branch

not taken branch

return

"01x"

A

to decoder

(c) Eric Rotenberg

"Trace Cache: Low Latency, High Bandwidth Instruction Fetching"
Predicting Multiple Branches

- Multiple correlating predictor (Yeh, Marr, Patt):

2^{14} 2-bit counters arranged in (2^{12} x 4) array

3 branch predictions

PATTERN HISTORY TABLE
GLOBAL HISTORY REGISTER

(c) Eric Rotenberg
Trace Cache Contents

• Parameters: n max instructions, m basic blocks

• In addition to tag and instructions, contains:
  - Branch flags: m-1 branch outcomes
  - Branch mask: number of branches
  - Trace fall-thru: next address if last branch is not-taken
  - Trace target: next address if last branch is taken

• Trace Cache may be small, we use:
  - 64 lines
  - direct mapped
  - n = 16, m = 3
  - total size: 712B (tag info) + 4KB (instr.)
Trace Cache Detail

Trace Cache Detail

(c) Eric Rotenberg

“Trace Cache: Low Latency, High Bandwidth Instruction Fetching”
Alternatives

- Branch Address Cache (Yeh, Marr, Patt)
- Collapsing Buffer (Conte, Menezes, Mills, Patel)
- Subgraph Predictor (Dutta, Franklin)
- Two-Block Ahead Predictor (Seznec, Jourdan, Sainrat, Michaud)
Weakness of Alternatives

• General idea behind alternatives:
  
  1. Generate multiple addresses pointing to several, possibly noncontiguous basic blocks.
  
  2. Apply the multiple addresses to an interleaved or multiported instruction cache.
  
  3. Properly order and merge only desired instructions into the predicted dynamic sequence.

  => complex and serial, long latency

• Trace cache: cache long instruction sequences in the order they are likely to be needed

  => moves complexity off critical path, short latency
Branch Address Cache

- 1st branch
  - A
  - B
    - D
      - H
    - E
      - I
    - F
      - J
      - K
  - C
    - T
      - L
    - G
      - M
      - N
      - O
• BTB design can detect intraline branches
• Two passes through BTB allow up to one interline branch
• Collapsing Buffer uses control info from both BTB passes to align instructions into dynamic sequence
Simulation Study

- Superscalar model as in introduction
- Maximum demand from execution engine:
  - Execution engine limited only by true data dependences
  - Oracle memory address disambiguation
  - Data cache always hits
- Instruction window: 2048
- Max dispatch/issue bandwidth: 16
- Workload: SPEC (sparc) and IBS (mips)
- Trace driven (wrong speculations not followed)
Simulation Study

Compare various fetch mechanisms

• Two base fetch models for comparison:
  - can fetch only sequential code
  - SEQ.1 : limited to 1 branch prediction per cycle
  - SEQ.3 : limited to 3 branch predictions per cycle

• Three models for high bandwidth instruction fetching:
  - TC : trace cache
  - BAC : branch address cache
  - CB : collapsing buffer

• Model with 1, 2, and 3 cycle latencies for CB and BAC
IPC for the Various Fetch Mechanisms, **Single-Cycle Latency**

![Graph showing IPC for various benchmarks with different fetch mechanisms.](image-url)

(c) Eric Rotenberg

“Trace Cache: Low Latency, High Bandwidth Instruction Fetching”

Slide 18
IPC for the Various Fetch Mechanisms, Single-Cycle Latency

SPEC92 benchmark

eqntott espresso xlisp gcc sc compress

SEQ.1
SEQ.3
BAC
CB
TC

(c) Eric Rotenberg

“Trace Cache: Low Latency, High Bandwidth Instruction Fetching”
Performance Improvement over SEQ.3, Single-Cycle Latency

% improvement in IPC

-5.0% 0.0% 5.0% 10.0% 15.0% 20.0% 25.0%

benchmark

eqtlott espresso xlisp gcc sc

BAC CB TC

“Trace Cache: Low Latency, High Bandwidth Instruction Fetching”
Performance Improvement over SEQ.3, Non-unit Latency

Benchmark: eqnott, espresso, perl, gcc, gcc, compress, verilog, groff, mpeg, jpeg, nroff

% improvement in IPC

(c) Eric Rotenberg
Comparing Trace Cache to Ideal

IBS benchmark

verilog  groff  gs  mpeg  jpeg  nroff

IPC

TC (4KB)  TC (32KB)  ideal3

“Trace Cache: Low Latency, High Bandwidth Instruction Fetching”
Conclusions

• Fetching past multiple not-taken branches improves performance by >10%

• A small trace cache gets an additional 10% or more by going past multiple taken branches

• Trace cache is consistently better than other proposed methods with similar goals assuming unit latency, much better with realistic latencies

• Trace cache is a natural trend towards reducing superscalar front-end complexity