Delivering Instruction Bandwidth
Using a Trace Cache

Sanjay J. Patel

High Performance Systems Group
Advanced Computer Architecture Laboratory
University of Michigan

March 1999
Research Objectives

My Objective:
Develop an instruction fetch mechanism for a 16-wide processor.

Group Objective:
Develop a 16-wide processor achieving a performance of 10 IPC.
Outline

- Processor Performance and the Partial Fetch Problem
- The Trace Cache
- Performance
- Summary
A Simple Processor

**Diagram:**
- **FETCH** → **EXECUTE** → **COMMIT**
...but for High Performance
Fetching for 10 IPC

- No empty fetches (caused by cache misses)
- No fetches thrown away (caused by mispredictions)
- No partial fetches (caused by branches)
## Average block lengths

<table>
<thead>
<tr>
<th></th>
<th>Inst/branch</th>
<th>Inst/taken branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>compress</td>
<td>9.15</td>
<td>13.07</td>
</tr>
<tr>
<td>gcc</td>
<td>5.34</td>
<td>8.46</td>
</tr>
<tr>
<td>go</td>
<td>6.21</td>
<td>9.11</td>
</tr>
<tr>
<td>ijpeg</td>
<td>15.59</td>
<td>19.14</td>
</tr>
<tr>
<td>li</td>
<td>5.54</td>
<td>8.41</td>
</tr>
<tr>
<td>m88ksim</td>
<td>4.24</td>
<td>5.78</td>
</tr>
<tr>
<td>perl</td>
<td>6.06</td>
<td>9.13</td>
</tr>
<tr>
<td>vortex</td>
<td>6.57</td>
<td>11.05</td>
</tr>
<tr>
<td>Average</td>
<td>7.34</td>
<td>10.52</td>
</tr>
</tbody>
</table>
The Trace Cache
Caching the dynamic instruction stream

Dynamic Instruction Stream

| A | B | C | ... | A | B | C |

Time
<table>
<thead>
<tr>
<th>Block A</th>
<th>Block C</th>
<th>Block B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Instruction Cache**

<table>
<thead>
<tr>
<th>Block A</th>
<th>Block B</th>
<th>Block C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Trace Cache**
The Microarchitecture

- Fill Unit
- Trace Cache
  - Fetch Address
  - Multiple Branch Predictor
  - Selection Logic
  - Path
  - Next Fetch Address
- Instruction Cache
  - Align/Merge
  - Decoder
- Level 2 Instruction Cache
- Level 2 Data Cache
- Register Rename
- Execution Core
Trace Cache Fetch Mechanism

- Fill Unit
- Trace Cache
- Multiple Branch Predictor
- Instruction Cache
- Align/Merge
- Decoder
- Selection Logic
- Path
- Next Fetch Address

From Execution Core

To Register Rename

From L2 ICache
Other approaches

• Hardware techniques
  – Branch Address Cache - Yeh et al., 1993
  – Collapsing Buffer - Conte et al., 1993
  – Multiple-Block Ahead - Seznec et al., 1996

• Compiler techniques
  – Trace Scheduling - Fisher, 1981
  – Superblock, Hyperblock - Hwu et al., 1989
  – Branch Alignment - Pettis, 1990, Calder, 1994
  – BS-ISA, block enlargement - Hao et al., 1997
Trace Cache history

Large Execution Atomic Units : Melvin, Patt 1989


Trace Cache : Rotenberg et al., Patel et al. 1996-7
Performance
Experimental infrastructure

- **Simulator**: Built upon SimpleScalar3.0
  - Simulates Alpha AXP code
  - Models out-of-order execution (HPS)

- **Benchmarks**: 8 SPECint95 + 5 UNIX apps
  - Aggressive compiler optimizations
  - Dependency graphs

- **Analysis Tools**: Trace segments
  - Branches
Baseline configurations
## Baseline specifics

<table>
<thead>
<tr>
<th></th>
<th>Trace Cache</th>
<th>Sequential</th>
<th>Single</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCache size</td>
<td>128KB</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>ICache size</td>
<td>4KB</td>
<td>128KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Blocks per fetch</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Branch predictor</td>
<td>Multiple</td>
<td>Multiple</td>
<td>Hybrid</td>
</tr>
<tr>
<td>Execution units</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>DCache size</td>
<td>64KB</td>
<td>64KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Inst window</td>
<td>512 insts</td>
<td>512 insts</td>
<td>512 insts</td>
</tr>
<tr>
<td>Mem dependence</td>
<td>Perfect</td>
<td>Perfect</td>
<td>Perfect</td>
</tr>
</tbody>
</table>
How well do the compiler optimizations do?

<table>
<thead>
<tr>
<th>Instructions per taken branch</th>
<th>No Code Layout</th>
<th>With Code Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>compress</td>
<td>13.07</td>
<td>26.07</td>
</tr>
<tr>
<td>gcc</td>
<td>8.46</td>
<td>14.02</td>
</tr>
<tr>
<td>go</td>
<td>9.11</td>
<td>15.05</td>
</tr>
<tr>
<td>ijpeg</td>
<td>19.14</td>
<td>31.65</td>
</tr>
<tr>
<td>li</td>
<td>8.41</td>
<td>10.85</td>
</tr>
<tr>
<td>m88ksim</td>
<td>5.78</td>
<td>14.86</td>
</tr>
<tr>
<td>perl</td>
<td>9.13</td>
<td>10.04</td>
</tr>
<tr>
<td>vortex</td>
<td>11.05</td>
<td>12.45</td>
</tr>
<tr>
<td>Average</td>
<td>10.52</td>
<td>16.87</td>
</tr>
</tbody>
</table>
First measurement

Instructions Per Cycle

Benchmark

Trace Cache
Sequential-Block
Single-Block

comp gcc go jpeg li m88k perl vor ch gs ppg plot ss
Trace Cache Enhancements

• Partial Matching
• Inactive Issue
• Branch Promotion
• Trace Packing
Average fetch misses per 1000 instructions

<table>
<thead>
<tr>
<th></th>
<th>Trace Cache</th>
<th>Sequential-Block</th>
<th>Single-Block</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>33.69</td>
<td>0.19</td>
<td>0.19</td>
</tr>
</tbody>
</table>
Partial Matching

Predicted path: ABC
Fetched segment: ABD

No Partial Matching: miss

Partial Matching: AB
Partial Matching results

- Trace Cache w/Partial Matching
- Sequential-Block
- Single-Block

Benchmarks: comp gcc go jpeg li m88k perl vor ch gs pgp plot ss

Instructions Per Cycle
Inactive Issue

Predicted path: ABC
Fetched segment: ABD

No Partial Matching: miss
Partial Matching: AB

Inactive Issue: AB (active) D(inactive)
Limitations on fetch size: gcc

Ave Fetch Size 10.027

Dynamic Frequency

- PartialMatch
- Icache
- MispredBR
- MaxSize
- Ret, Indir, Trap
- AtomicBlocks
- MaximumBRs
Branch Promotion

Without Branch Promotion

With Branch Promotion

A  B  D  empty

A  B  BD  G

A
B
D
G

A  4 insts
B  3 insts
D  4 insts

A  4 insts
B  3 insts
D  4 insts
G  3 insts
Implementing Branch Promotion

Branch Bias Table

Branch Address

dir n-bit saturating counter

Previous outcome

# Consecutive occurrences

Promote

n
Promoted branch frequency

Percentage of dynamic branches

Promoted Faults
Promoted
Non-Promoted

Benchmarks

comp gcc go jpeg li m88k perl vor ch gs pgp plot ss

Advanced Computer Architecture Laboratory
The University of Michigan
Limitations on trace size: gcc with Promotion

Ave Fetch Size 10.558
Trace Packing

Without Trace Packing

With Trace Packing
Performance w/ enhancements

- Enhanced Trace Cache
- Basic Trace Cache
- Sequential-Block
- Single-Block

Instructions Per Cycle

Benchmarks
- comp
- gcc
- go
- jpeg
- li
- m88k
- perl
- vor
- ch
- gs
- pgp
- plot
- ss

Advanced Computer Architecture Laboratory
The University of Michigan
Average fetch size w/enhancements

Benchmarks

comp gcc go ijpeg li m88k perl vor ch gs pgp plot ss

Ave Fetch Size

Enh. Trace Cache
Basic Trace Cache
Sequential-Block
Single-Block

14% 26% 14% 54% 38% 55% 44% 22% 49% 14% 40% 59%
34% increase in fetch size.

But only a 14% increase in performance.

Why?
Because of...

- Execution bottlenecks
- Longer time to resolve branches
- Duplication in the trace cache
Effect of execution bandwidth

![Graph showing the effect of execution bandwidth. The x-axis represents the average fetch size, ranging from 0 to 16. The y-axis represents the instructions per cycle, ranging from 0 to 10. Three lines depict different execution bandwidth strategies: Ideal, Aggressive, and Conservative. Each line shows an increase in instructions per cycle as the average fetch size increases.](image-url)
Effect on branch resolution time

![Bar charts showing execution time in cycles for gcc and go programs with different fetch window sizes (FW 4, FW 8, FW 16). The bars are divided into Useful Fetches, Branch Misses, Cache Misses, and Other categories.](image)

- **gcc**
  - FW 4: Higher execution time, dominated by Useful Fetches.
  - FW 8: Moderate execution time, balanced among categories.
  - FW 16: Lower execution time, balanced among categories.

- **go**
  - FW 4: Higher execution time, dominated by Useful Fetches.
  - FW 8: Moderate execution time, balanced among categories.
  - FW 16: Lower execution time, balanced among categories.

*Advanced Computer Architecture Laboratory
The University of Michigan*
Branch resolution time vs. fetch size

Average Fetch Size

Ave Missed Br Resolution Time

Ideal

Aggressive
Duplication in the trace cache

ICache: No duplications
1 iteration every 3 cycles

Enhanced Trace Cache: 6 segments
2.5 iterations every 3 cycles

6 instructions

5 instructions

6 instructions
Measure of duplication

Ave Num Copies Per Instruction

- no Trace Packing
- cost-regulated Trace Packing (Redundancy)
- cost-regulated Trace Packing (Replication)
- unregulated Trace Packing

Benchmarks

comp gcc go ijpeg li m88k perl vor ch gs pgp plot ss
Finally...
Major Findings

- Enhanced TCache outperforms Sequential ICache
  - IPC increases by 14%
  - Fetch size increases by 34%
- 60% fewer dynamic cond. branches with Promotion
- Mispredictions more costly as fetch size increases
- Duplication high, but most copies useful