Dual Path Instruction Processing

Juan L. Aragón¹, José González¹,* , Antonio González²,* and James E. Smith³

¹ Dept. Ing. y Tecnología de Computadores
   Universidad de Murcia

² Dept. d’Arquitectura de Computadores
   Universitat Politécnica de Catalunya

³ Dept. Electrical and Computing Eng.
   University of Wisconsin-Madison

* Currently at Intel Barcelona Research Center

e-mail: jlaragon@ditec.um.es
Two ways of reducing performance degradation due to branch mispredictions

- Improving prediction accuracy
- Reducing branch misprediction penalty

Branch misprediction penalty

- Deeper pipelines cause higher misprediction penalties
  - Pentium 4 (20 stages); Power 4 (14 stages)
  - Example: IPC slowdown of 22%, using 32 KB gshare comparing a pipeline of 20 stages over 10 stages (go)
Motivation

Causes of performance degradation after a branch misprediction

- Pipeline must be squashed
- Many cycles until new instructions can be issued
  - Front-end length
- Instruction window is not full during many cycles
  - ILP cannot be fully exploited
- Correct instructions cannot be scheduled ahead a mispredicted branch
Outline

- Misprediction Penalty Analysis
- Proposal
- Dual Path Instruction Processing (DPIP)
- Experimental Results
- Sensitivity Analysis
- Conclusions
Three Components

- **Pipeline-fill penalty**
  - Delay between the misprediction and the first correct instruction enters the window
  - Depends on
    - Pipeline length, Recovery actions

- **Window-fill penalty**
  - Window empty many cycles after misprediction
  - ILP cannot be fully exploited

- **Serialization penalty**
  - Correct instructions cannot be scheduled ahead of the mispredicted branch
Misprediction Penalty Analysis

Analysis of each component

<table>
<thead>
<tr>
<th></th>
<th>Overall loss</th>
<th>Pipeline-fill penalty</th>
<th>Window-fill penalty</th>
<th>Serialization penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipeline 6</td>
<td>25%</td>
<td>25%</td>
<td>10%</td>
<td>65%</td>
</tr>
<tr>
<td>pipeline 10</td>
<td>33%</td>
<td>44%</td>
<td>7%</td>
<td>49%</td>
</tr>
<tr>
<td>pipeline 14</td>
<td>39%</td>
<td><strong>54%</strong></td>
<td>6%</td>
<td>40%</td>
</tr>
</tbody>
</table>
Outline

- Misprediction Penalty Analysis
- Proposal
- Dual Path Instruction Processing (DPIP)
- Experimental Results
- Sensitivity Analysis
- Conclusions
Proposal

- Reduce *Pipeline-fill* and *Window-fill* penalties

- **Dual Path Instruction Processing (DPIP)**
  - Fetches, decodes and renames both paths
    - Reduce *Pipeline-fill* penalty
    - Hide front-end stages
  - Alternative path instructions are pre-scheduled in an estimated execution order
    - Reduce *Window-fill* penalty
    - Similar effect as filling the window completely

- **Confidence estimation**
  - Used to filter branches that must be forked
Related work

- **Multiple path execution (MPE)**
  - Fetch, decode and *execute* instructions from multiple paths
    - Selective Dual Path Execution (Heil & Smith, Tech.Report’97)
    - PolyPath (Klauser et al, ISCA’98)
    - Threaded Multiple Path Execution (Wallace et al, ISCA’98)
  - Too expensive (drawbacks)
    - Aggressive fetch engines (allowing up to 8 different paths!!!)
    - Bigger register files, instruction windows and ROBs
    - Complexity of selective flush
    - Resource contention: more functional units, memory ports,...
    - Energy consumption: resources used by useless instructions

*DPIP does not execute instructions*

*balance between complexity, cost, and performance*
Outline

- Misprediction Penalty Analysis
- Proposal
- Dual Path Instruction Processing (*DPIP*)
- Experimental Results
- Sensitivity Analysis
- Conclusions
Dual Path Instruction Processing

**DPIP block diagram**

DPIP can only manage two paths at the same time.
Pre-scheduling alternative path instructions

- **RMT_1**
  - Free list_1

- **Decode Unit**
  - *predicted path*

- **Instruction Window**
  - *issue logic*

- **RMT_2**
  - Free list_2

- **Pre-schedule Buffer**
  - *data-flow order*

- *alternative path*

References:
- Canal & Gonzalez, ICS 2000
- Michaud & Seznec, HPCA 2001
Pre-scheduling Example

\[
\text{schedule\_line} = \max( \{\text{reg\_availability}(\text{input reg1}), \\
\quad \text{reg\_availability}(\text{input reg2})\} )
\]

\[
\text{reg\_availability}(\text{output register}) = \text{schedule\_line} + \text{execution\_latency}
\]

Alternative path instructions:
A \quad r2 \leftarrow r1 + r0
B \quad \text{store} r3, 0(r2)
C \quad \text{load} r2, 0(r6)
D \quad r4 \leftarrow r2 + r0
E \quad r4 \leftarrow r3 + r3
Results

- OoO superscalar simulator *(sim-outorder)*

- Configuration
  - Fetch/decode/issue/commit up to 8 inst/cycle
  - L1 cache: 64 KB I-cache, 64 KB D-cache (2 way)
  - L2 cache: 512 KB 4-way
  - 8 Int ALU’s, 2 Int Mult
  - 8 FP ALU’s, 2 FP mult
  - 64-entry Instruction Window
  - 128-entry Reorder Buffer
  - 14-stage pipeline (IBM Power 4 - like)

- Evaluated programs
  - SpecInt95 and SpecInt2000
**DPIP performance**

- **8%** improvement for **DPIP** (with pre-scheduling)
- **10%** improvement for **DPIP + branch prediction reversal**
- **17%** for oracle estimation (still work to be done)
How much pre-scheduling influences *DPIP* performance?

- 16% of improvement provided by pre-scheduling (31% for *go*)
- Pre-scheduling provides additional benefits.
Outline

- Misprediction Penalty Analysis
- Proposal
- Dual Path Instruction Processing (DPIP)
- Experimental Results
- Sensitivity Analysis
- Conclusions
• 5% average speedup (up to 8% for bzip2)
• 15% for oracle estimation
Improvements remain constant as the window grows
- 10 stages: 4% average speedup
- 20 stages: 12% average speedup
Outline

- Misprediction Penalty Analysis
- Proposal
- Dual Path Instruction Processing (DPIP)
- Experimental Results
- Sensitivity Analysis
- Conclusions
Conclusions

- Categorized branch misprediction penalty
  - Pipeline-fill penalty
  - Window-fill penalty
  - Serialization penalty

- Dual Path Instruction Processing reduces penalties of mispredicted branches
  - Fetches, decodes, renames and pre-schedules alternative path instructions
  - Similar effect as filling the window completely

- Balance between complexity/cost/performance
  - Simpler than Multiple Path Execution schemes

- 12% speedup for 20-stages OoO processors
  - 25% for oracle estimation

*contribution to the overall performance loss*