Data Speculation

Adam Wierman    Daniel Neill


Sodani and Sohi. *Understanding the differences between value prediction and instruction reuse*, 1998.
A Taxonomy of Speculation

What can we speculate on?

Speculative Execution

Control Speculation

Branch Direction

Data Speculation

Branch Target

Data Location

Data Value

Question: What makes speculation possible?
Value Locality

**Question:** Where does value locality occur?

<table>
<thead>
<tr>
<th>Somewhat</th>
<th>Single-cycle Arithmetic (i.e. <code>addq $1 $2</code>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Single-cycle Logical (i.e. <code>bis $1 $2</code>)</td>
</tr>
<tr>
<td>No</td>
<td>Multi-cycle Arithmetic (i.e. <code>mulq $1 $2</code>)</td>
</tr>
<tr>
<td>Yes</td>
<td>Register Move (i.e. <code>cmov $1 $2</code>)</td>
</tr>
<tr>
<td>Yes</td>
<td>Integer Load (i.e. <code>ldq $1 8($2)</code>)</td>
</tr>
<tr>
<td>No</td>
<td>Store with base register update</td>
</tr>
<tr>
<td>Yes</td>
<td>FP Load</td>
</tr>
<tr>
<td>Somewhat</td>
<td>FP Multiply</td>
</tr>
<tr>
<td>Somewhat</td>
<td>FP Add</td>
</tr>
<tr>
<td>Yes</td>
<td>FP Move</td>
</tr>
</tbody>
</table>
Value Locality

Question: Why is speculation **useful**?

```
addq $1 $2 $3
addq $3 $1 $4
addq $3 $2 $5
```

Speculation lets all these **run in parallel** on a superscalar machine
Exploiting Value Locality

“predict the results of instructions based on previously seen results”

Value Prediction (VP)
Instruction Reuse (IR)

“recognize that a computation chain has been previously performed and therefore need not be performed again”
Exploiting Value Locality

Fetch → Decode → Issue → Execute → Commit

Predict Value

if mispredicted

Verify

Value Prediction (VP)

Instruction Reuse (IR)

Fetch → Decode → Issue → Execute → Commit

Check for previous use

Verify arguments are the same

if reused
Value Prediction

(Lipasti & Shen, 1996)
Value prediction

- Speculative prediction of register values
  - Values predicted during fetch and dispatch, forwarded to dependent instructions.
  - Dependent instructions can be issued and executed immediately.
  - Before committing a dependent instruction, we must verify the predictions. If wrong: must restart dependent instruction with correct values.

![Value Prediction Diagram](image)
Overview

Classification Table (CT)

Value Prediction Table (VPT)

Should I predict?

Predicted Value

Prediction
### How to predict values?

#### Classification Table (CT)

<table>
<thead>
<tr>
<th>PC</th>
<th>Pred History</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Value Prediction Table (VPT)

<table>
<thead>
<tr>
<th>PC</th>
<th>Value History</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Value Prediction Table (VPT)
- Cache indexed by instruction address (PC)
- Mapped to one or more 64-bit values
- Values replaced (LRU) when instruction first encountered or when prediction incorrect.
- 32 KB cache: 4K 8-byte entries
Estimating prediction accuracy

Classification Table (CT)
- Cache indexed by instruction address (PC)
- Mapped to 2-bit saturating counter, incremented when correct and decremented when wrong.
  - 0,1 = don’t use prediction
  - 2 = use prediction
  - 3 = use prediction and don’t replace value if wrong
- 1K entries sufficient

Value Prediction Table (VPT)
Verifying predictions

• Predicted instruction executes normally.
• Dependent instruction cannot commit until predicted instruction has finished executing.
• Computed result compared to predicted; if ok then dependent instructions can commit.
• If not, dependent instructions must reissue and execute with computed value. Miss penalty = 1 cycle later than no prediction.
Results

- Realistic configuration, on simulated (current and near-future) PowerPC gave 4.5-6.8% speedups.
  - 3-4x more speedup than devoting extra space to cache.
- Speedups vary between benchmarks (grep: 60%)
- Potential speedups up to 70% for idealized configurations.
  - Can exceed dataflow limit (on idealized machine).
Instruction Reuse

(Sodani & Sohi, 1998)
Instruction Reuse

• Obtain results of instructions from their previous executions.
  – If previous results still valid, don’t execute the instruction again, just commit the results!
• Non-speculative, early verification
  – Previous results read in parallel with fetch.
  – Reuse test in parallel with decode.
  – Only execute if reuse test fails.

Fetch → Decode → Issue → Execute → Commit

Check for previous use -> Verify arguments are the same

if reused
How to reuse instructions?

- **Reuse buffer**
  - Cache indexed by instruction address (PC)
  - Stores result of instruction along with info needed for establishing reusability:
    - Operand register names
    - Pointer chain of dependent instructions
  - Assume 4K entries (each entry takes 4x as much space as VPT: compare to 16K VP)
  - 4-way set-associative.
Reuse Scheme

• Dependent chain of results (each points to previous instruction in chain)
  – Entry is reusable if the entries on which it depends have been reused (can’t reuse out of order).
  – Start of chain: reusable if “valid” bit set; invalidated when operand registers overwritten.
  – Special handling of loads and stores.

• Instruction will not be reused if:
  – Inputs not ready for reuse test (decode stage)
  – Different operand registers
Results

- Attempts to evaluate “realistic” and “comparable” schemes for VP and IR on simulated MIPS architecture.
- Are these really realistic? Assume oracle or || test.
- Net performance: VP better on some benchmarks; IR better on some. All speedups typically 5-10%.
- More interesting question: can the two schemes be combined?
- Claim: 84-97% of redundant instructions reusable.
Comparing VP and IR

“predict the results of instructions based on previously seen results”

Value Prediction (VP)
Instruction Reuse (IR)

“recognize that a computation chain has been previously performed and therefore need not be performed again”
Comparing VP and IR

Value Prediction (VP)

"recognize that a computation chain has been previously performed and therefore need not be performed again"

Instruction Reuse (IR)

IR can’t predict when:
1. Inputs aren’t ready
2. Same result follows from different inputs
3. VP makes a lucky guess

Which captures more redundancy?

Which captures more redundancy?
Comparing VP and IR

Value Prediction (VP)
Instruction Reuse (IR)

“predict the results of instructions based on previously seen results”

Which handles misprediction better?

Which captures more redundancy?

IR is non-speculative, so it never mispredicts
Comparing VP and IR

Value Prediction (VP)
“predict the results of instructions based on previously seen results”

Instruction Reuse (IR)

Which integrates best with branches?

IR
1. Mispredicted branches are detected earlier
2. Instructions from mispredicted branches can be reused.

VP
1. Causes more misprediction

Which captures more redundancy?
Comparing VP and IR

Value Prediction (VP)

Instruction Reuse (IR)

“predict the results of instructions based on previously seen results”

Which is better for resource contention?

IR might not even need to execute the instruction
Comparing VP and IR

Value Prediction (VP)

Instruction Reuse (IR)

“predict the results of instructions based on previously seen results”

Which is better for execution latency?

VP causes some instructions to be executed twice (when values are mispredicted),
IR executes once or not at all.
Value Prediction (VP)
Instruction Reuse (IR)

“predict the results of instructions based on previously seen results”

“recognize that a computation chain has been previously performed and therefore need not be performed again”

Possible class project: Can we get the best of both techniques?
Data Speculation

Adam Wierman    Daniel Neill


Sodani and Sohi. *Understanding the differences between value prediction and instruction reuse*, 1998.
• Value prediction can handle these cases, thus captures more redundancy.

• But IR has several advantages…
  – Skips execute phase when reusing instruction.
  – Early, non-speculative test; never “mispredicts”