Using Predicate Path Information in Hardware to Determine True Dependences

Lori Carter and Brad Calder
University of California, San Diego
June 26, 2002
Target – **EPIC Architecture**

- Explicitly **Parallel Instruction Computing**
  - Supports Predicated Execution
- VLIW in nature
- Able to communicate some analysis to the hardware
- Intel Itanium IA64 Architecture is EPIC
Predicated Execution
(If-conversion)

- Both paths execute
- Only path guarded by true commits
If-Conversion Complicates Dependency Analysis

$$b > c$$

$$b = 3$$  $$c = a + b$$

cycle

P1, P2  $$b > c$$  1
(P1) $$b = 3$$  2
(P2) $$c = a + b$$  3

data dependence?
If-Conversion Complicates Dependency Analysis

No data dependence, P1 and P2 are disjoint
Research Goal

Predicate-Sensitive Analysis is Critical!

Our Goal: Provide Dynamic Predicate-Sensitive Analysis for the In-Order EPIC Architecture
Related Work

Compiler based predicate-sensitive analysis mechanisms

Predicate Query System (PQS)

*Analysis Techniques for Predicated Code*
Johnson and Schlansker, Micro ’96

Full Path Predicates

*Predicated Static Single Assignment*
Carter et. al., PACT ’99

Predicate Analysis System (PAS)

*Accurate and Efficient Predicate Analysis with Binary Decision Diagrams*
Sias et. al., Micro 2000
Related Work

Hardware Solutions for Multiple Path Definitions

Select-μop

```
add r5=r2,r4
cmp P2,P3 =r5,0
(P2) mov r5'=r7
(P3) add r6=r5 or r5',3
```

Register Renaming for Dynamic Execution of Predicated Code

Wang et. al., HPCA 2001

Our Work in Disjoint Path Analysis recognizes the need for dynamic predicate analysis within the EPIC structure
Outline

• Introduction to predication and predicate-sensitive analysis

• Related Work

• Motivate need for dynamic predicate-sensitive dependency analysis for EPIC architectures

• Use and implementation of Disjoint Path Analysis

• Methodology and Results

• Conclusions and Future Work
Motivation for Disjoint Path Analysis

• Itanium has a scoreboard

• Predicate Register values help determine when dependences are broken

(P4) ld r7 = [r5]
(P5) mov r8 = r7

• Disjoint Path Analysis
  • Don’t set unnecessary dependences
### Possible Schedules

#### Base Itanium, no disjointness information

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><code>cmp P4, P5 = r8, r5</code></td>
<td>1. Comparison</td>
</tr>
<tr>
<td>1</td>
<td><code>(P4) ld r7 = [r5]</code></td>
<td>2. Load from memory</td>
</tr>
<tr>
<td>2</td>
<td><code>(P5) mov r8 = r7</code></td>
<td>3. Move register</td>
</tr>
</tbody>
</table>

#### Base Itanium, with disjointness information

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><code>cmp P4, P5 = r8, r5</code></td>
<td>1. Comparison</td>
</tr>
<tr>
<td>1</td>
<td><code>(P4) ld r7 = [r5]</code></td>
<td>2. Load from memory</td>
</tr>
<tr>
<td>1</td>
<td><code>(P5) mov r8 = r7</code></td>
<td>3. Move register</td>
</tr>
</tbody>
</table>
Two Predicates Defined in Unconditional CMP are Disjoint

1. \( c=d-2 \)
2. \( \text{cmp P1, P2} \) \( b>c \)
3. (P1) \( b=3 \)
4. (P1) \( \text{cmp P3, P4} \) \( b>a \)
5. (P2) \( c=a+b \)
6. (P3) \( a=b+3 \)
7. (P4) \( a=c+4 \)

Questions:

Is \( b \) in statement 3 a definition of \( b \) in statement 5?
Predicate Definition Inherits Disjointness Properties from Guarding Predicate

1. $c = d - 2$
2. $b > c$
3. $(P1) \ b = 3$
4. $(P1) \ b > a$
5. $(P2) \ c = a + b$
6. $(P3) \ a = b + 3$
7. $(P4) \ a = c + 4$

Questions:

Is $b$ in statement 3 a definition of $b$ in statement 5?

Is $c$ in statement 5 a definition of $c$ in statement 7?
Outline

• Introduction to predication and predicate-sensitive analysis
• Related Work
• Motivate need for dynamic predicate-sensitive dependency analysis for EPIC architectures
  • Use and implementation of Disjoint Path Analysis
• Methodology and Results
• Conclusions and Future Work
Basic Structures Required to Create Predicate-Sensitive Analysis in Hardware

• Structure to keep track of multiple possible definitions that reach a use and which predicate guards each

Register Alias Table [Wang, HPCA 2001]

add r5 = r2, r4
cmp P2, P3 = r5, 0
(P2) mov r5 = r7
(P3) add r6 = r5, 3
(P3) cmp P4, P5 = r6, 0
(P4) mov r5 = 3
(P4) mov r6 = -1
(P4) mov r6 = 0
(P5) mov r6 = 0
mul r9 = r5, r6
cmp P5, P6 = r4, r5
Basic Structures Required to Create Predicate-Sensitive Analysis in Hardware

- Structure to keep track of multiple possible definitions that reach a use and which predicate guards each

Register Alias Table

- Structure to maintain disjointness information

Path Information Table

```
add r5=r2,r4
cmp P2,P3 =r5,0
(P2) mov r5=r7
(P3) add r6=r5,3
(P3) cmp P4,P5 = r6,0
(P4) mov r5=3
(P4) mov r6=-1
(P5) mov r6=0
    mult r9=r5,r6
    cmp P5,P6=r4,r5
```
Basic Structures Required to Create Predicate-Sensitive Analysis in Hardware

- Structure to keep track of multiple possible definitions that reach a use and which predicate guards each

  **Register Alias Table (RAT)**

- Structure to maintain disjointness information

  **Path Information Table (PIT)**

- Structure to recall what the current definition of a predicate is

  **Last Definition Table (LDT)**

```
add r5=r2,r4
cmp P2,P3 =r5,0
(P2) mov r5=r7
(P3) add r6=r5,3
(P3) cmp P4,P5 = r6,0
(P4) mov r5=3
(P4) mov r6=-1
(P5) mov r6=0
  mult r9=r5,r6
  cmp P5,P6=r4,r5
(P6) mov r8=r6
```
Finding Dependences Using the RAT, PIT and LDT

Register Alias Table

<table>
<thead>
<tr>
<th>slot 0</th>
<th>slot 1</th>
<th>slot 2</th>
<th>slot 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>def inst</td>
<td>def inst</td>
<td>def inst</td>
<td>def inst</td>
</tr>
</tbody>
</table>

Path Information Table

<table>
<thead>
<tr>
<th>Logical register</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>...</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>...</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Last Definition Table

<table>
<thead>
<tr>
<th>Predicate Register</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pit entry 0</td>
</tr>
<tr>
<td>2</td>
<td>Pit entry 1</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td></td>
</tr>
</tbody>
</table>

Instructions:
1. add r5=r2,r4
2. cmp P2,P3 =r5,0
3. (P2) mov r5=r7
4. (P3) add r6=r5,3
Finding Dependences Using the RAT, PIT and LDT

Register Alias Table
1. add r5=r2,r4
2. cmp P2,P3 =r5,0
3. (P2) mov r5=r7
4. (P3) add r6=r5,3

Path Information Table

Predicate Register

Last Definition Table

University of California
San Diego
Finding Dependences Using the RAT, PIT and LDT

Register Alias Table

1. add r5=r2,r4
2. cmp P2,P3 =r5,0
3. (P2) mov r5=r7
4. (P3) add r6=r5,3

Path Information Table

Predicate Register

1. Pit entry 0
2. Pit entry 1
3. 4
4. 5
5. 6
6. 7
7. 8
8. 9
9. 63
Inserting Register Definitions into the RAT

<table>
<thead>
<tr>
<th>logical register</th>
<th>slot 0</th>
<th>slot 1</th>
<th>slot 2</th>
<th>slot 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>def inst PIT</td>
<td>def inst PIT</td>
<td>def inst PIT</td>
<td>def inst PIT</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>…</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[1]</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>…</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Register Alias Table

Path Information Table

Last Definition Table

Predicate Register

1 2 3 4 5 6 …… N

0 1 2 3 4 5 6

1 add r5=r2,r4
Adding Predicate Disjointness Information

1. add r5 = r2, r4
2. cmp P2, P3 = r5, 0

Register Alias Table

Path Information Table

Last Definition Table

Predicate Register

P2 → 0
P3 → 1

1. Pit entry 0
2. Pit entry 1

University of California
San Diego
Inheriting Predicate Disjointness Information

Register Alias Table

<table>
<thead>
<tr>
<th>Logical Register</th>
<th>def inst PIT</th>
<th>slot 0</th>
<th>def inst PIT</th>
<th>slot 1</th>
<th>def inst PIT</th>
<th>slot 2</th>
<th>def inst PIT</th>
<th>slot 3</th>
<th>def inst PIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>…</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[4]</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Path Information Table

 Predicate
Register

<table>
<thead>
<tr>
<th>Predicate</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>…</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Vector copied

Complement bits set

Last Definition Table

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Pit entry 0</th>
<th>Pit entry 1</th>
<th>Pit entry 2</th>
<th>Pit entry 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>…</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1  add r5=r2,r4
2  cmp P2,P3 =r5,0
3  (P2) mov r5=r7
4  (P3) add r6=r5,3
5  (P3) cmp P4,P5 = r6,0
IA64Simplescalar

Benchmark Traces

• Uses traces generated on IA64 machines using ptrace interface
  - binaries created using Electron, SGI and Intel C compilers
• Traces decoded using libraries adapted from GNU opcode library

Simplescalar Adaptations

• Inorder execution
  • Additional dependences
• Supports Predicated Execution
  • Possible multiple definitions
  • Commit only on true predicate
  • Break dependences for false guarding predicate
  • Additional dependences
• Software Pipelining
  • Rotating registers
  • Associated branch instructions
• Advanced Load Address Table to support Speculation
• Bundle and Stop Bit detection
## Parameters for Simulated Architecture

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 I-Cache</td>
<td>16k, 4way set-associative, 32 byte blocks, 2 cycles latency</td>
</tr>
<tr>
<td>L1 D-Cache</td>
<td>16k, 4way set-associative, 32 byte blocks, 2 cycles latency</td>
</tr>
<tr>
<td>Unified L2 Cache</td>
<td>96k, 6way set-associative, 64 byte blocks, 6 cycles latency</td>
</tr>
<tr>
<td>Unified L3 Cache</td>
<td>2Meg, direct mapped, 64 byte blocks, 21 cycle latency</td>
</tr>
<tr>
<td>Functional Units</td>
<td>2 integer ALU / 2 load-store units / 2 FP units / 3 branch units</td>
</tr>
<tr>
<td>Branch Predictor</td>
<td>meta-predictor (bimodal &amp; 2-level g-share) ea table, 4096 entries</td>
</tr>
</tbody>
</table>
Configurations Compared

• Itanium Implementation
• Disjoint Path Analysis with 4-way RAT
• Disjoint Path Analysis with 16-way RAT
• Perfect Predicate Prediction
IPC Gain Produced by Disjoint Path Architecture in If-Converted Regions

% Speedup in if-converted regions

- exchange *(10.2)
- max_subseq (3.3)
- mm (15)
- sqrt (11.2)
- nested (16.9)
- ave

*( ) percent of executed instructions that were if-converted
Disjoint Path Analysis Compared to Perfect Predicate Prediction

Percent of Perfect Predicate Prediction gain achieved

- exchange
- max_subseq
- mm
- sqrt
- nested
- ave

path 4
path 16
Conclusions

- The hardware needs the same predicate-sensitive analysis as the compiler
- IPC was increased up to 6% in if-converted regions for the benchmarks studied
- We averaged almost 50% of the improvement that could be achieved with perfect predicate value knowledge

Future Work:
- Application for an out-of-order processor supporting predication
- Exploring ways to more completely communicate compiler-generated predicate-relationship information to the hardware