IA-64 Architecture Innovations

Abbreviated Version of 2/23/99 IA-64 Architecture Disclosure
Agenda

- Review content disclosed at IDF
  - Focus on benefits
  - Focus on “What’s New?”
- Branch Handling: Predication and Prediction
- Speculation
- Register Rotation & Loop Handling
So What’s New?

- **Static prediction**
  - Improves prediction of always or never taken branches
- **Parallel compares**
  - Increased parallelism through logical combination of compares
- **Hoisting uses**
  - Increased scheduling flexibility improves performance
- **Data Speculation**
  - Moves loads above stores, increasing scheduling flexibility and performance
- **Nat bits**
  - Enables deferral of exceptions, supports more aggressive speculation
- **Multiway branch**
  - EXECutes multiple branches in a single cycle
- **Register rotation**
  - Enables wider use of software pipelining performance benefits
- **Predicate rotation**
  - More efficient implementation of software pipelining
Today’s Processors are often 60% Idle
IA-64 Architecture: Explicit Parallelism

Increases Parallel Execution

Compiler

Original Source Code

Compile

Parallel Machine Code

Hardware

multiple functional units

More efficient use of execution resources

IA-64 Compiler Views Wider Scope

Increases Parallel Execution
Branch Handling

Traditional Arch

- Dynamic Prediction
- Static Prediction on a per branch basis
- Predication

IA-64

- & Others

• Branches are breaks in code
• Can indicate a decision
• Common in wide variety of applications
Dynamic Branch Prediction

Traditional Architectures
- Guess either B or C
- Suffer performance penalty when mispredicted
- 5-10% mispredict rate can cost 40+% performance
Static Branch Prediction

Compiler: “It's going to be B”

- Compiler knows which one is almost always taken or never taken
- Removes guesswork for processor, reduces mispredict penalties
- Concentrates hardware resources on other difficult to predict branches
Review

Predication

- Removes Branch, executes B&C in parallel
- Avoids possibility of mispredict
- Predicates (p1 & p2) are bits that turn on/off B & C
- Biggest benefit to code w/ hard to predict branches
  - Large server apps
  - Data sorting
Parallel Compares: Extends Predication

- Reduces critical path, further increasing performance
- Enables reduction from 7 to 4 cycles on queens loop

Unique feature to IA-64
Predication Benefits

- Reduces branches and mispredict penalties
  - 50% fewer branches and 37% faster code*
- Parallel compares further reduce critical paths
- Greatly improves code with hard to predict branches
  - Large server apps - capacity limited
  - Sorting, data mining - large database apps
  - Data compression
- Traditional architectures’ “bolt-on” approach can’t efficiently approximate predication
  - Cmove: 39% more instructions, 30% lower performance*
  - Instructions must all be speculative

* Source: S. Manke, 1995
Speculation Review

Traditional Architectures

```
instr 1
instr 2
br
```

IA-64

```
Id.s
instr 1
instr 2
br
chk.s
use
```

Memory latency stalls CPU

Elevates load, reduces memory latency impact

Speculation overcomes the memory latency bottleneck in today’s & tomorrow’s systems
Hoisting Uses

- The uses of speculative data can also be executed speculatively.
- Provides additional scheduling flexibility to achieve greater parallelism.
Introducing Data Speculation

- Compiler can issue a load prior to a preceding, possibly-conflicting store.

**Traditional Architectures**

```
<table>
<thead>
<tr>
<th>Instruction 1</th>
<th>Instruction 2</th>
<th>...</th>
<th>Store 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load 8</td>
<td>Use</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**IA-64**

```
<table>
<thead>
<tr>
<th>Load 8.a</th>
<th>Instruction 1</th>
<th>Instruction 2</th>
<th>Store 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- Elevates load above "store barrier".
- Memory latency stalls CPU.

New!

Unique feature to IA-64
Exception Handling

**New!**

**Traditional Method 1**
- instr 1
- instr 2
- Load
- ...
- br

**Traditional Method 2**
- instr 1
- instr 2
- Ldnf
- ...
- br

**IA-64**
- ld.s
- instr 1
- instr 2
- br

- chk.s
- use
- Home Block

- NaT Bit

**Only in limited instances**

**Extra baggage to ensure integrity**
- & can’t hoist uses
- (e.g. non-faulting load)

**Widely applicable,**
- NaT bit ensures
- integrity of data
- without penalty
Speculation Benefits

- Reduces impact of memory latency
  - Performance improvement at 79% when combined with predication*
- Greatest improvement to code with many cache accesses
  - Large databases
  - Operating systems
- Scheduling flexibility enables new levels of performance headroom

* August et.al., 1998
Loop Handling

“Do loop 4 times”

IA-64 has special register to accelerate loops & avoid mispredicts: called Loop Counter (LC)

Improves integer code performance

New!

Loop
Multi-way Branch

w/o Speculation

ld8 r6 = (ra)
(p1) br exit1

ld8 r7 = (rb)
(p3) br exit2

ld8 r8 = (rc)
(p5) br exit3

3 branch cycles

Hoisting Loads

ld8.s r6 = (ra)
ld8.s r7 = (rb)
ld8.s r8 = (rc)

chk r6, rec0
(p1) br exit1

chk r7, rec1
(p3) br exit2

chk r8, rec2
(p5) br exit3

1 branch cycle

IA-64

ld8.s r6 = (ra)
ld8.s r7 = (rb)
ld8.s r8 = (rc)

chk r6, rec0
(p2) chk r7, rec1
(p4) chk r8, rec2
{}

(p1) br exit1
(p3) br exit2
(p5) br exit3

New!

Multiway branches: more than 1 branch in a single cycle

Chaining multiway branches allows n-way branching
Software Pipelining

- Overlapping execution of different loop iterations
- More iterations in same amount of time

IA-64 Offers an Innovative Approach
Software Pipelining

- **IA-64 features that make this possible**
  - Full Predication
  - Special branch handling features
  - Register rotation: removes loop copy overhead
  - Predicate rotation: removes prologue & epilogue

- **Traditional architectures use loop unrolling**
  - High overhead: extra code for loop body, prologue, and epilogue

Especially Useful for Integer Code With Small Number of Loop Iterations
Introducing Rotating Registers

<table>
<thead>
<tr>
<th>Traditional Arch</th>
<th>IA-64</th>
</tr>
</thead>
<tbody>
<tr>
<td>load in R36</td>
<td>load in R36</td>
</tr>
<tr>
<td>load in R37</td>
<td></td>
</tr>
<tr>
<td>load in R38</td>
<td></td>
</tr>
<tr>
<td>load in R39</td>
<td></td>
</tr>
<tr>
<td>R35</td>
<td>R35</td>
</tr>
<tr>
<td>R36</td>
<td>R36</td>
</tr>
<tr>
<td>R37</td>
<td>R37</td>
</tr>
<tr>
<td>R38</td>
<td>R38</td>
</tr>
<tr>
<td>R39</td>
<td>R39</td>
</tr>
<tr>
<td>R40</td>
<td>R40</td>
</tr>
</tbody>
</table>

4 instructions

1 instruction

Improves performance without code expansion
Software Pipelining Benefits

- Loop pipelining maximizes performance; minimizes overhead
  - Avoids code expansion of unrolling and code explosion of prologue and epilogue
  - Smaller code means fewer cache misses
  - Greater performance improvements in higher latency conditions
- Reduced overhead allows S/W pipelining of small loops with unknown trip counts
  - Typical of integer scalar codes
Reviewing What’s New:

- Parallel compares
- Tbit
- Nat bits
- Deferral
- Hoisting uses
- Propagation
- Branch instructions
- Static prediction

- Loop branches
- LC register
- EC register
- Multiway branch
- Branch registers
- Register rotation
- Predicate rotation
- RRBs
# Feature Comparison

<table>
<thead>
<tr>
<th>Traditional Architectures</th>
<th>IA-64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic branch prediction</td>
<td>Branch specific Static Prediction and Predication enhance dynamic prediction to reduce mispredict penalties</td>
</tr>
<tr>
<td>Conditional moves limited in applicability and require additional instructions</td>
<td>Predication widely applicable, parallel compares further enhance benefit</td>
</tr>
<tr>
<td>Non faulting loads limited to certain conditions or require additional instructions</td>
<td>Control and data speculation enable greater scheduling freedom of loads</td>
</tr>
<tr>
<td>Software pipelining limited to large loops due to code size explosion</td>
<td>Rotating registers and rotating predicates allow wide application of software pipelining performance benefits without the code growth</td>
</tr>
</tbody>
</table>
Summary

- **Predication removes branches**
  - Eliminates branches & mispredicts, increases ILP
  - Good for large database applications

- **Speculation reduces memory latency**
  - Enhances ILP and scalability
  - Good for variety of server applications
    - (databases, OLTP, etc.)

- **S/W pipelining support enables broad usage**
  - Performance for small integer loops with unknown trip counts as well as monster FP loops