RECAP

B649
Parallel Architectures and Programming
RECAP
Recap

- ILP
- Exploiting ILP
- Dynamic scheduling
- Thread-level Parallelism
- Memory Hierarchy
- Other topics through student presentations
- Virtual Machines
ILP: Pipelining
Pipelining: Adding Latches
Pipelining: Adding Forwarding
Pipelining: Adding Branch Delay Slots

(a) From before
DADD R1, R2, R3
if R2 = 0 then
  Delay slot
becomes
if R2 = 0 then
  DADD R1, R2, R3

(b) From target
DSUB R4, R5, R6
if R1 = 0 then
  Delay slot
DADD R1, R2, R3
if R1 = 0 then
  DSUB R4, R5, R6

(c) From fall-through
DADD R1, R2, R3
if R1 = 0 then
  OR R7, R8, R9
  Delay slot
DSUB R4, R5, R6
Extending the Basic Pipeline
Extending the Basic Pipeline
Extending the Basic Pipeline
Exploiting ILP Through Compiler Techniques

- Loop unrolling
- Making use of branch delayed slots
- Static branch prediction
- Loop fusion
- Unroll and jam
- ...

B629: Practical Compiling for Modern Machines
Dynamic Branch Prediction

Address bits

bits to index BPB

Branch Prediction Buffer

0 1
1 0

...
Two-bit Branch Predictor
Use Branch Target Buffers (BTBs) for caching branch targets
Dynamic Scheduling: Tomasulo’s Approach
Tomasulo’s Approach: Observations

- RAW hazards handled by waiting for operands
- WAR and WAW hazards handled by register renaming
  - only WAR and WAW hazards between instructions currently in the pipeline are handled; is this a problem?
  - larger number of hidden names reduces name dependences
- CDB implements forwarding
Tomasulo’s Approach + Speculation

Fields in ROB
1. Instruction type
2. Destination
3. Value
4. Ready
Observations on Speculation

- Speculation enables precise exception handling
  - defer exception handling until instruction ready to commit

- Branches are critical to performance
  - prediction accuracy
  - latency of misprediction detection
  - misprediction recovery time

- Must avoid hazards through memory
  - WAR and WAW already taken care of (how?)
  - for RAW
    - don’t allow load to proceed if an active ROB entry has Destination field matching with A field of load
    - maintain program order for effective address computation (why?)
## Multiple Issue Processor Types

<table>
<thead>
<tr>
<th>Common name</th>
<th>Issue structure</th>
<th>Hazard detection</th>
<th>Scheduling</th>
<th>Distinguishing characteristic</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Superscalar (static)</td>
<td>dynamic</td>
<td>hardware</td>
<td>static</td>
<td>in-order execution</td>
<td>mostly in the embedded space: MIPS and ARM</td>
</tr>
<tr>
<td>Superscalar (dynamic)</td>
<td>dynamic</td>
<td>hardware</td>
<td>dynamic</td>
<td>some out-of-order execution, but no speculation</td>
<td>none at the present</td>
</tr>
<tr>
<td>Superscalar (speculative)</td>
<td>dynamic</td>
<td>hardware</td>
<td>dynamic with speculation</td>
<td>out-of-order execution with speculation</td>
<td>Pentium 4, MIPS R12K, IBM Power5</td>
</tr>
<tr>
<td>VLIW/LIW</td>
<td>static</td>
<td>primarily software</td>
<td>static</td>
<td>all hazards determined and indicated by compiler (often implicitly)</td>
<td>most examples are in the embedded space, such as the TI C6x</td>
</tr>
<tr>
<td>EPIC</td>
<td>primarily static</td>
<td>primarily software</td>
<td>mostly static</td>
<td>all hazards determined and indicated explicitly by the compiler</td>
<td>Itanium</td>
</tr>
</tbody>
</table>
Dyn. Scheduling+Multiple Issue+Speculation

• Design parameters
  ★ two-way issue (two instruction issues per cycle)
  ★ pipelined and separate integer and FP functional units
  ★ dynamic scheduling, but not out-of-order issue
  ★ speculative execution

• Task per issue: assign reservation station and update pipeline control tables (i.e., control signals)

• Two possible techniques
  ★ do the task in half a clock cycle
  ★ build wider logic to issue any pair of instructions together

• Modern processors use both (4 or more way superscalar)
Shared-Memory Multiprocessors

- Processor
  - One or more levels of cache
  - Main memory
- Processor
  - One or more levels of cache
- Processor
  - One or more levels of cache
- Processor
  - One or more levels of cache
  - I/O system
Distributed-Memory Multiprocessors
Other Ways to Categorize Parallel Programming

- data vs. task parallel
- client-server vs. p-to-p / master-slave vs. symm.
- course vs. fine grained
- SPMD vs. MPMD
- threads vs. producer-consumer
- tightly vs. loosely coupled
- shared mem vs. msg passing
- recursive vs. iterative
- synch. vs. asynch.
- Parallel Programs
Write Invalidate Cache Coherence Protocol for Write-Back Caches
Distributed Memory + Directories

[Diagram showing a network of processors, caches, memories, and I/O connections, interconnected through a central interconnection network.]
Directory-Based Cache Coherence

Invalid

- CPU read
- Send read miss message
- CPU write
- Data write back
- Read miss
- Data write back
- Send invalid message
- Fetch invalidate
- CPU write hit
- CPU read hit
- Data write back
- Write miss
- CPU write miss
- Modified (read/write)

Shared (read only)

- CPU read hit
- Read miss
- Data value reply
- Write miss
- Uncached

Exclusive (read/write)

- Write miss
- Data value reply
- Fetch/Invalidate
- Data value reply
- Write miss
- Share = Share + {P}

Shared (read only)

- Read miss
- Data value reply
- Share = Share + {P}

B629: Practical Compiling for Modern Machines
Other Topics

• x86 assembly programming
• VLIW / EPIC
• Vector processors
• Embedded systems
• Scientific applications
• GPUs and GPGPUs
• CUDA and OpenCL
• Interconnection networks
• Virtualization
WHAT’S NEXT?
Future

• Continued importance of parallel programming
  ★ challenge: how to program multiprocessors
  ★ role of programming languages and compilers

• Convergence or specialization?
  ★ “standardization” of general purpose architecture
  ★ migration of “special-purpose” CPUs for general use
Landscape of Parallel Computing Research: A View from Berkeley

The recent switch to parallel microprocessors is a milestone in the history of computing. A multidisciplinary group of researchers here in Berkeley has been meeting since Spring 2005 to discuss this change from the conventional wisdom. Our white paper summarizes our learnings from these discussions. This wiki is a meetingplace for us as a research community to explore the future of parallel processing. The video interview with Dave Patterson, Krste Asanovic and Kurt Keutzer, or Dave Patterson's presentation at a recent Distinguished Colloquium here at Berkeley are great introductions to the Berkeley View project. Here are the slides from a related talk by Dave Patterson.

We believe that much can be learned by examining the success of parallelism at the extremes of the computing spectrum, namely embedded computing and high performance computing. This led us to