2110412
Parallel Computer Architecture:
Multi-core Tech.
4: Advance ILP, Multiprocessor

Krerk Piromsopa, Ph.D.
Department of Computer Engineering
Chulalongkorn University

Exposing and Exploiting ILP

- Basic Scheduling and Loop Unrolling
- Software Pipelining: Symbolic Loop Unrolling
- Predicated Instructions

Loop Unroll

C Code:
For (i=1000; i>0; i=i-1)
x[i] = x[i] + s;

The straightforward MIPS code (without scheduling)
Loop:
  L.D F0, 0(R1) ; F0 = array element
  ADD.D F4, F0, F2 ; add scalar in F2
  S.D F4, 0(R1) ; store result
  DADDUI R1, R1, #-8 ; decrement pointer 8 bytes (per DW)
  BNE R1, R2, Loop ; branch R1!=R2

<table>
<thead>
<tr>
<th>Original Loop</th>
<th>Unroll</th>
<th>ReSchedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 L.D F0,0(R1)</td>
<td>L.D F0,0(R1)</td>
<td>L.D F0,0(R1)</td>
</tr>
<tr>
<td>2 stall</td>
<td>stall</td>
<td>stall</td>
</tr>
<tr>
<td>3 ADD.D F4,F0,F2</td>
<td>ADD.D F1,F0,F2</td>
<td>ADD.D F1,F0,F2</td>
</tr>
<tr>
<td>4 stall</td>
<td>stall</td>
<td>stall</td>
</tr>
<tr>
<td>5 stall</td>
<td>stall</td>
<td>DADDUI R1,R1,#-16</td>
</tr>
<tr>
<td>6 S.D F4,0(R1)</td>
<td>S.D F3,0(R1)</td>
<td>S.D F3,0(R1)</td>
</tr>
<tr>
<td>7 DADDUI R1,R1,#-8</td>
<td>L.D F3,-8(R1)</td>
<td>BNE R1,R2, Loop</td>
</tr>
<tr>
<td>8 stall</td>
<td>stall</td>
<td>stall</td>
</tr>
<tr>
<td>9 BNE R1,R2, Loop</td>
<td>ADD.D F4,F3,F2</td>
<td></td>
</tr>
<tr>
<td>10 Stall</td>
<td>stall</td>
<td>stall</td>
</tr>
<tr>
<td>11 stall</td>
<td>stall</td>
<td>stall</td>
</tr>
<tr>
<td>12 S.D F4,-8(R1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 DADDUI R1,R1,#-16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 stall</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 BNE R1,R2, Loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 Stall</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Maximum # of times?
POP QUIZ:

Loop:
L.D F0, 0(R1) ; F0=array element
ADD.D F4, F0, F2 ; add scalar in F2
S.D F4, 0(R1) ; store result
DADDUI R1, R1, #-8 ; decrement pointer 8 bytes (per DW)
BNE R1, R2, Loop ; branch R1!=R2

Assume the number of iterations is unknown, but large.

What is the actual maximum number of times the simple loop can be unrolled using the given MIPS code? What is the limiting resource? Show how to increase the number of times the loop may be unrolled by transforming the MIPS code to make less intensive use of the limiting resource. How much does this transformation improve performance?

Software Pipeline

- Increasing the possibility that the unrolled loop can be scheduled without stalls.
- Interleaves instructions from different iterations.
Predicate Instruction

- C Code
  
  ```c
  if (A==0) { S=T; }
  ```

- Assembly Code
  
  ```assembly
  BNEZ R1, L
  ADDU R2, R3, R0
  CMOVZ R2, R3, R1
  ```

IA64

- Intel & HP
- Explicit Parallel instruction Computing (EPIC)
  
  - Instruction level parallelism
  - VLIW
  - Branch predication
  - Speculative loading
Execution Units

- **I-Unit**
  - Integer arithmetic
  - Shift and add
  - Logical
  - Compare
  - Integer multimedia ops
- **M-Unit**
  - Load and store
  - Between register and memory
  - Some integer ALU
- **B-Unit**
  - Branch instructions
- **F-Unit**
  - Floating point instructions

Comparison

<table>
<thead>
<tr>
<th>Supervisor</th>
<th>IA-64</th>
</tr>
</thead>
<tbody>
<tr>
<td>RISC-like instructions, one per word</td>
<td>RISC-like instructions bundled into groups of three</td>
</tr>
<tr>
<td>Multiple parallel execution units</td>
<td>Multiple parallel execution units</td>
</tr>
<tr>
<td>Reduces and optimizes instruction stream at run time</td>
<td>Reduces and optimizes instruction stream at compile time</td>
</tr>
<tr>
<td>Branch prediction with speculative execution of one path</td>
<td>Speculative execution along both paths of a branch</td>
</tr>
<tr>
<td>Loads data from memory only when needed, and tries to find the data in the cache first</td>
<td>Speculatively loads data before it's needed, and still tries to find data in the cache first</td>
</tr>
</tbody>
</table>

Instruction Format

- **128-bit bundle**
  - Instruction slot 2
  - Instruction slot 1
  - Instruction slot 0
  - Byte

- **41 41 41 5**

- **4 31 6**

- **4 10 7 7 7 6**

**Notes:**
- PS = Predicate register
- GR = General or floating point register
**Thread-Level Parallelism**

- No Thread
- Temporal
- Coarse
- Fine
- Simultaneous

**Chip Multiprocessor**

- Each processor is assigned thread

**Taxonomy of Parallel Processor**

**Multiprogramming and Multi Processing**

- Process 1
- Process 2
- Process 3

- Blocking
- Running
**SMP**

- Symmetric Shared-Memory Architectures (SMPs)
- Uniform Memory Access (UMA)
- Processors share memory

**Advantage/Disadvantage**

- **Advantage**
  - Simplicity, Flexibility, Reliability
- **Disadvantage**
  - Performance Limitation
  - Cache Coherency

**Cluster**

- A group of interconnected computers (nodes) working together as a unified resource (one machine).
- **Scalability, Availability, and Performance**
Cluster Architecture

Non-Uniform memory access

Access Times may vary

Comparison:

- SMP has limit to number of processors (16 - 64 processors)
- Each cluster has own memory
  - (Application do not see global memory)
- NUMA - Maintain large system wide memory with scalability (1000+)
Exercise

From the following code fragment, assume that all data references are shown, assume that all values are defined before use, and that only b and c are used again after this segment. You may ignore any possible exceptions. The individual statements are numbered to provide an easy reference.

1. if (a>c) {
2.   d = d + 5;
3.   a = b + d + e;
} else {
4.   e = e + 2;
5.   f = f + 2;
6.   c = c + f;
7.   b = a + f;

List the control dependences