Multicore & Parallel Processing

Kevin Walsh
CS 3410, Spring 2010
Computer Science
Cornell University

P&H Chapter 4.10-11, 7.1-6
Q: How to improve system performance?

→ Increase CPU clock rate?

→ But I/O speeds are limited
   
   Disk, Memory, Networks, etc.

Recall: Amdahl’s Law

Solution: Parallelism
Pipelining: execute multiple instructions in parallel

Q: How to get more instruction level parallelism?

A: Deeper pipeline
  - Less work per stage $\Rightarrow$ shorter clock cycle

A: Multiple issue pipeline
  - Start multiple instructions per clock cycle in duplicate stages
  - Example: 1GHz 4-way multiple-issue
    
    Peak CPI = 0.25 $\Rightarrow$ 16 billion instructions per second
Static Multiple Issue

a.k.a. Very Long Instruction Word (VLIW)

Compiler groups instructions to be issued together
- Packages them into “issue slots”
- Simple HW: Compiler detects and avoids hazards

Example: Static Dual-Issue MIPS
- Instructions come in pairs (64-bit aligned)
  - One ALU/branch instruction (or nop)
  - One load/store instruction (or nop)
## Compiler scheduling for dual-issue MIPS...

TOP:

```
lw   $t0, 0($s1)  # $t0 = A[i]
lw   $t1, 4($s1)  # $t1 = A[i+1]
addu $t0, $t0, $s2  # add $s2
addu $t1, $t1, $s2  # add $s2
sw   $t0, 0($s1)  # store A[i]
sw   $t1, 4($s1)  # store A[i]
addi $s1, $s1, +8  # increment pointer
bne $s1, $s3, TOP  # continue if $s1!=end
```

<table>
<thead>
<tr>
<th>ALU/branch slot</th>
<th>Load/store slot</th>
<th>cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOP:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td>lw  $t0, 0($s1)</td>
<td>1</td>
</tr>
<tr>
<td>addu $s1, $s1, +8</td>
<td>lw  $t1, 4($s1)</td>
<td>2</td>
</tr>
<tr>
<td>addu $t0, $t0, $s2</td>
<td>nop</td>
<td>3</td>
</tr>
<tr>
<td>addu $t1, $t1, $s2</td>
<td>sw  $t0, -8($s1)</td>
<td>4</td>
</tr>
<tr>
<td>bne $s1, $s3, TOP</td>
<td>sw  $t1, -4($s1)</td>
<td>5</td>
</tr>
</tbody>
</table>
Dynamic Multiple Issue

a.k.a. SuperScalar Processor

- CPU examines instruction stream and chooses multiple instructions to issue each cycle
- Compiler can help by reordering instructions....
- ... but CPU is responsible for resolving hazards

Even better: Speculation/Out-of-order Execution

- Guess results of branches, loads, etc.
- Execute instructions as early as possible
- Roll back if guesses were wrong
- Don’t commit results until all previous insts. are retired
Q: Does multiple issue / ILP work?
A: Kind of... but not as much as we’d like

Limiting factors?

• Programs dependencies
• Hard to detect dependencies → be conservative
  – e.g. Pointer Aliasing: A[0] += 1; B[0] *= 2;
• Hard to expose parallelism
  – Can only issue a few instructions ahead of PC
• Structural limits
  – Memory delays and limited bandwidth
• Hard to keep pipelines full
Q: Does multiple issue / ILP cost much?

A: Yes.

→ Dynamic issue and speculation requires power

<table>
<thead>
<tr>
<th>CPU</th>
<th>Year</th>
<th>Clock Rate</th>
<th>Pipeline Stages</th>
<th>Issue width</th>
<th>Out-of-order/Speculation</th>
<th>Cores</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>i486</td>
<td>1989</td>
<td>25MHz</td>
<td>5</td>
<td>1</td>
<td>No</td>
<td>1</td>
<td>5W</td>
</tr>
<tr>
<td>Pentium</td>
<td>1993</td>
<td>66MHz</td>
<td>5</td>
<td>2</td>
<td>No</td>
<td>1</td>
<td>10W</td>
</tr>
<tr>
<td>Pentium Pro</td>
<td>1997</td>
<td>200MHz</td>
<td>10</td>
<td>3</td>
<td>Yes</td>
<td>1</td>
<td>29W</td>
</tr>
<tr>
<td>P4 Willamette</td>
<td>2001</td>
<td>2000MHz</td>
<td>22</td>
<td>3</td>
<td>Yes</td>
<td>1</td>
<td>75W</td>
</tr>
<tr>
<td>UltraSparc III</td>
<td>2003</td>
<td>1950MHz</td>
<td>14</td>
<td>4</td>
<td>No</td>
<td>1</td>
<td>90W</td>
</tr>
<tr>
<td>P4 Prescott</td>
<td>2004</td>
<td>3600MHz</td>
<td>31</td>
<td>3</td>
<td>Yes</td>
<td>1</td>
<td>103W</td>
</tr>
<tr>
<td>Core</td>
<td>2006</td>
<td>2930MHz</td>
<td>14</td>
<td>4</td>
<td>Yes</td>
<td>2</td>
<td>75W</td>
</tr>
<tr>
<td>UltraSparc T1</td>
<td>2005</td>
<td>1200MHz</td>
<td>6</td>
<td>1</td>
<td>No</td>
<td>8</td>
<td>70W</td>
</tr>
</tbody>
</table>

→ Multiple simpler cores may be better?
Curve shows ‘Moore’s Law’: transistor count doubling every two years.
Moore’s law

• A law about transistors
• Smaller means more transistors per die
• And smaller means faster too

But: Power consumption growing too...
Power Limits Performance

- Nuclear Reactor
- Rocket Nozzle
- Sun's Surface

- Hot plate
- Pentium III® processor
- Pentium II® processor
- Pentium Pro® processor
- Pentium® processor
- i386
- i486

- Watts/cm²

- 1000
- 100
- 10
- 1

- 1.5μ, 1μ, 0.7μ, 0.5μ, 0.35μ, 0.25μ, 0.18μ, 0.13μ, 0.1μ, 0.07μ
Power = capacitance * voltage^2 * frequency

In practice: Power ~ voltage^3

Reducing voltage helps (a lot)
... so does reducing clock speed
Better cooling helps

The **power wall**

- We can’t reduce voltage further
- We can’t remove more heat
Why Multicore?

<table>
<thead>
<tr>
<th></th>
<th>Performance</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Core</td>
<td>1.2x</td>
<td>1.7x</td>
</tr>
<tr>
<td>Overclocked</td>
<td>1.0x</td>
<td>1.0x</td>
</tr>
<tr>
<td>Dual-Core</td>
<td>1.02x</td>
<td>1.6x</td>
</tr>
<tr>
<td>Underclocked</td>
<td>-20%</td>
<td>-20%</td>
</tr>
</tbody>
</table>

Single-Core
Overclocked +20%
Single-Core
Dual-Core
Underclocked -20%
AMD Barcelona: 4 processor cores
Q: So lets just all use multicore from now on!
A: Software must be written as parallel program

Multicore difficulties

- Partitioning work
- Coordination & synchronization
- Communications overhead
- Balancing load over cores
- How do you write parallel programs?
  - ... without knowing exact underlying architecture?
Partition work so all cores have something to do
Load Balancing

Need to partition so all cores are actually working
If tasks have a **serial part** and a **parallel part**...

**Example:**

- step 1: divide input data into \( n \) pieces
- step 2: do work on each piece
- step 3: combine all results

**Recall:** **Amdahl’s Law**

As number of cores increases ...

- time to execute parallel part?
- time to execute serial part?