IT TOOK A LOT OF WORK, BUT THIS LATEST LINUX PATCH ENABLES SUPPORT FOR MACHINES WITH 4,096 CPUs, UP FROM THE OLD LIMIT OF 1,024.

Do you have support for smooth full-screen Flash video yet?

No, but who uses that?
Big Picture: Multicore and Parallelism
Big Picture: Multicore and Parallelism

Why do I need *four* computing cores on my phone?!
Big Picture: Multicore and Parallelism

Why do I need *eight* computing cores on my phone?!
Big Picture: Multicore and Parallelism

Why do I need *sixteen* computing cores on my phone?!
Pitfall: Amdahl’s Law

Execution time after improvement =

affected execution time

amount of improvement

+ execution time unaffected

\[ T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}} \]
Pitfall: Amdahl’s Law

Improving an aspect of a computer and expecting a proportional improvement in overall performance

\[ T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}} \]

Example: multiply accounts for 80s out of 100s
• How much improvement do we need in the multiply performance to get 5× overall improvement?

\[ 20 = \frac{80}{n} + 20 \quad – \quad \text{Can’t be done!} \]
Scaling Example

Workload: sum of 10 scalars, and $10 \times 10$ matrix sum

- Speed up from 10 to 100 processors?

Single processor: Time = $(10 + 100) \times t_{\text{add}}$

10 processors
- Time = $100/10 \times t_{\text{add}} + 10 \times t_{\text{add}} = 20 \times t_{\text{add}}$
- Speedup = $110/20 = 5.5$

100 processors
- Time = $100/100 \times t_{\text{add}} + 10 \times t_{\text{add}} = 11 \times t_{\text{add}}$
- Speedup = $110/11 = 10$

Assumes load can be balanced across processors
Scaling Example

What if matrix size is $100 \times 100$?

Single processor: Time = $(10 + 10000) \times t_{\text{add}}$

10 processors
- Time = $10 \times t_{\text{add}} + \frac{10000}{10} \times t_{\text{add}} = 1010 \times t_{\text{add}}$
- Speedup = $\frac{10010}{1010} = 9.9$

100 processors
- Time = $10 \times t_{\text{add}} + \frac{10000}{100} \times t_{\text{add}} = 110 \times t_{\text{add}}$
- Speedup = $\frac{10010}{110} = 91$

Assuming load balanced
Goals for Today

How to improve System Performance?

• Instruction Level Parallelism (ILP)
• Multicore
  – Increase clock frequency vs multicore
• Beware of Amdahls Law

Next time:

• Concurrency, programming, and synchronization
Problem Statement

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
Instruction-Level Parallelism (ILP)

Pipelining: execute multiple instructions in parallel

Q: How to get more instruction level parallelism?

A: Deeper pipeline
   - E.g. 250MHz 1-stage; 500Mhz 2-stage; 1GHz 4-stage; 4GHz 16-stage

Pipeline depth limited by...
   - max clock speed (less work per stage ⇒ shorter clock cycle)
   - min unit of work
   - dependencies, hazards / forwarding logic
Instruction-Level Parallelism (ILP)

Pipelining: execute multiple instructions in parallel

Q: How to get more instruction level parallelism?

A: Multiple issue pipeline

– Start multiple instructions per clock cycle in duplicate stages

ALU/Br

LW/SW
Static Multiple Issue

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

Compiler groups instructions to be issued together
  • Packages them into “issue slots”

Q: How does HW detect and resolve hazards?
A: It doesn’t.
  ➔ Simple HW, assumes compiler avoids hazards

Example: Static Dual-Issue 32-bit MIPS
  • Instructions come in pairs (64-bit aligned)
    – One ALU/branch instruction (or nop)
    – One load/store instruction (or nop)
MIPS with Static Dual Issue

Two-issue packets

- One ALU/branch instruction
- One load/store instruction
- 64-bit aligned
  - ALU/branch, then load/store
  - Pad an unused instruction with nop

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction type</th>
<th>Pipeline Stages</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>ALU/branch</td>
<td>IF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MEM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WB</td>
</tr>
<tr>
<td>n + 4</td>
<td>Load/store</td>
<td>IF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MEM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WB</td>
</tr>
<tr>
<td>n + 8</td>
<td>ALU/branch</td>
<td>IF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MEM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WB</td>
</tr>
<tr>
<td>n + 12</td>
<td>Load/store</td>
<td>IF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MEM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WB</td>
</tr>
<tr>
<td>n + 16</td>
<td>ALU/branch</td>
<td>IF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MEM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WB</td>
</tr>
<tr>
<td>n + 20</td>
<td>Load/store</td>
<td>IF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MEM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WB</td>
</tr>
</tbody>
</table>
Scheduling Example

Schedule this for dual-issue MIPS

\[
\text{Loop: } \text{lw} \ $t0, 0($s1) \quad \# \ $t0=\text{array element} \\
\text{addu} \ $t0, $t0, $s2 \quad \# \ \text{add scalar in } $s2 \\
\text{sw} \ $t0, 0($s1) \quad \# \ \text{store result} \\
\text{addi} \ $s1, $s1, -4 \quad \# \ \text{decrement pointer} \\
\text{bne} \ $s1, $zero, \text{Loop} \quad \# \ \text{branch } $s1! = 0
\]

<table>
<thead>
<tr>
<th>ALU/branch</th>
<th>Load/store</th>
<th>cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop:</td>
<td>\text{nop}</td>
<td>1</td>
</tr>
<tr>
<td>addi $s1, $s1, -4</td>
<td>\text{nop}</td>
<td>2</td>
</tr>
<tr>
<td>addu $t0, $t0, $s2</td>
<td>\text{nop}</td>
<td>3</td>
</tr>
<tr>
<td>bne $s1, $zero, Loop</td>
<td>\text{sw} $t0, 4($s1)</td>
<td>4</td>
</tr>
</tbody>
</table>

\[
\frac{5 \text{ instructions}}{4 \text{ cycles}} = \text{IPC} = 1.25 \quad \frac{4 \text{ cycles}}{5 \text{ instructions}} = \text{CPI} = 0.8
\]
Scheduling Example

Compiler scheduling for dual-issue MIPS...

Loop:  
- `lw $t0, 0($s1)`  
- `lw $t1, 4($s1)`  
- `addu $t0, $t0, $s2`  
- `addu $t1, $t1, $s2`  
- `sw $t0, -8($s1)`  
- `sw $t1, -4($s1)`  
- `addi $s1, $s1, +8`  
- `bne $s1, $s3, Loop`  

ALU/branch slot
- `nop`  
- `addu $t0, $t0, $s2`  
- `addu $t1, $t1, $s2`  
- `addi $s1, $s1, +8`  
- `bne $s1, $s3, Loop`

Load/store slot
- `lw $t0, 0($s1)`  
- `lw $t1, 4($s1)`  
- `nop`  
- `sw $t0, 0($s1)`  
- `sw $t1, 4($s1)`  
- `nop`  
- `lw $t0, 0($s1)`  
- `lw $t1, 4($s1)`  

Delay slot
- `nop

Cycle
- `1`  
- `2`  
- `3`  
- `4`  
- `5`  
- `6`

8 cycles

6 cycles

\[\frac{6 \text{ cycles}}{8 \text{ instructions}} = \text{CPI} = 0.75\]
Scheduling Example

Compiler scheduling for dual-issue MIPS...

Loop:  

- 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  $t0, -8($s1)  # store A[i]
- sw  $t1, -4($s1)  # store A[i+1]
- addi $s1, $s1, +8  # increment pointer
- bne  $s1, $s3, Loop  # continue if $s1!=end

ALU/branch slot
Loop:  

- nop
- addi $s1, $s1, +8
- addu $t0, $t0, $s2
- addu $t1, $t1, $s2
- bne  $s1, $s3, Loop

Load/store slot  

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>lw $t0, 0($s1)</td>
</tr>
<tr>
<td>2</td>
<td>lw $t1, 4($s1)</td>
</tr>
<tr>
<td>3</td>
<td>nop</td>
</tr>
<tr>
<td>4</td>
<td>sw $t0, -8($s1)</td>
</tr>
<tr>
<td>5</td>
<td>sw $t1, -4($s1)</td>
</tr>
</tbody>
</table>

8 cycles  5 cycles

5 cycles
8 instructions  = CPI = 0.625
Limits of Static Scheduling
Compiler scheduling for dual-issue MIPS...

```
lw  $t0, 0($s1)         # load A
addi $t0, $t0, +1      # increment A
sw  $t0, 0($s1)         # store A
lw  $t0, 0($s2)         # load B
addi $t0, $t0, +1      # increment B
sw  $t0, 0($s2)         # store B
```

<table>
<thead>
<tr>
<th>ALU/branch slot</th>
<th>Load/store slot</th>
<th>cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>nop</td>
<td>lw  $t0, 0($s1)</td>
<td>1</td>
</tr>
<tr>
<td>nop</td>
<td>nop</td>
<td>2</td>
</tr>
<tr>
<td>addi $t0, $t0, +1</td>
<td>nop</td>
<td>3</td>
</tr>
<tr>
<td>nop</td>
<td>sw  $t0, 0($s1)</td>
<td>4</td>
</tr>
<tr>
<td>nop</td>
<td>lw  $t0, 0($s2)</td>
<td>5</td>
</tr>
<tr>
<td>nop</td>
<td>nop</td>
<td>6</td>
</tr>
<tr>
<td>addi $t0, $t0, +1</td>
<td>nop</td>
<td>7</td>
</tr>
<tr>
<td>nop</td>
<td>sw  $t0, 0($s2)</td>
<td>8</td>
</tr>
</tbody>
</table>
Limits of Static Scheduling
Compiler scheduling for dual-issue MIPS...

lw  $t0, 0($s1)  # load A
addi $t0, $t0, +1  # increment A
sw  $t0, 0($s1)  # store A
lw  $t1, 0($s2)  # load B
addi $t1, $t1, +1  # increment B
sw  $t1, 0($s2)  # store B

ALU/branch slot
nop
nop
addi $t0, $t0, +1
nop
nop
nop
addi $t1, $t1, +1
nop

Load/store slot  cycle
lw  $t0, 0($s1)  1
nop  2
nop  3
sw  $t0, 0($s1)  4
lw  $t1, 0($s2)  5
nop  6
nop  7
sw  $t1, 0($s2)  8
Limits of Static Scheduling
Compiler scheduling for dual-issue MIPS...

```
lw $t0, 0($s1)      # load A
addi $t0, $t0, +1   # increment A
sw $t0, 0($s1)      # store A
lw $t1, 0($s2)      # load B
addi $t1, $t1, +1   # increment B
sw $t1, 0($s2)      # store B
```

<table>
<thead>
<tr>
<th>ALU/branch slot</th>
<th>Load/store slot</th>
<th>cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>nop</td>
<td>lw $t0, 0($s1)</td>
<td>1</td>
</tr>
<tr>
<td>nop</td>
<td>lw $t1, 0($s2)</td>
<td>2</td>
</tr>
<tr>
<td>addi $t0, $t0, +1</td>
<td>nop</td>
<td>3</td>
</tr>
<tr>
<td>addi $t1, $t1, +1</td>
<td>sw $t0, 0($s1)</td>
<td>4</td>
</tr>
<tr>
<td>nop</td>
<td>sw $t1, 0($s2)</td>
<td>5</td>
</tr>
</tbody>
</table>

Problem: What if $s1 and $s2 are equal (aliasing)? Won’t work
Dynamic Multiple Issue

a.k.a. SuperScalar Processor (c.f. Intel)

- 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

- Execute instructions as early as possible
- Aggressive register renaming
- Guess results of branches, loads, etc.
- Roll back if guesses were wrong
- Don’t commit results until all previous insts. are retired
Dynamic Multiple Issue
Does Multiple Issue Work?

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 $\rightarrow$ 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
**Power Efficiency**

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?
Moore's Law

- 4004
- 8008
- 8080
- 8088
- 286
- 386
- 486
- Pentium
- Itanium 2
- P4
- K8
- Dual-core Itanium 2
- K10
- Atom

Curve shows 'Moore's Law': transistor count doubling every two years.
Why Multicore?

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

- Surface of Sun
- Rocket Nozzle
- Nuclear Reactor
- Hot Plate

- Xeon
- 32nm
- 180nm
Power Wall

Power = capacitance * voltage² * frequency

In practice: Power ~ voltage³

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?

Performance

Power

Single-Core

Overclocked +20%

1.2x
1.7x

1.0x

1.0x

1.6x

1.02x

Single-Core

Dual-Core

Underclocked -20%
Inside the Processor

AMD Barcelona Quad-Core: 4 processor cores
Inside the Processor

Intel Nehalem Hex-Core

4-wide pipeline
Hyperthreading

<table>
<thead>
<tr>
<th>Programs:</th>
<th>Multi-Core vs. Multi-Issue</th>
<th>vs. HT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num. Pipelines:</td>
<td>(N) (1)</td>
<td>(N)</td>
</tr>
<tr>
<td>Pipeline Width:</td>
<td>(1) (N)</td>
<td>(N)</td>
</tr>
</tbody>
</table>

Hyperthreads
- HT = MultiIssue + extra PCs and registers – dependency logic
- HT = MultiCore – redundant functional units + hazard avoidance

Hyperthreads (Intel)
- Illusion of multiple cores on a single core
- Easy to keep HT pipelines full + share functional units
Example: All of the above

8 die (aka 8 sockets)
4 core per socket
2 HT per core

Note: a socket is a processor, where each processor may have multiple processing cores, so this is an example of a multiprocessor multicore hyperthreaded system.
Parallel Programming

Q: So let's just all use multicore from now on!
A: Software must be written as a 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?
Work Partitioning

Partition work so all cores have something to do
Load Balancing

Need to partition so all cores are actually working
Amdahl’s Law

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? goes to zero
- time to execute serial part? Remains the same
- *Serial part eventually dominates*
Parallel Programming

Q: So let's 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?

SW

Your career...

HW
**Administrivia**

Lab3 is *due today*, Thursday, April 11\(^{th}\)

Project3 available now, due Monday, April 22\(^{nd}\)
- Design Doc *due next week*, Monday, April 15\(^{th}\)
- Schedule a Design Doc review Mtg now, by *tomorrow* Friday, April 12\(^{th}\)
- *See me after class if looking for new partner*
- *Competition/Games night Friday, April 26\(^{th}\), 5-7pm. Location: B17 Upson*

Homework4 is available now, *due next week*, Wednesday, April 17\(^{th}\)
- Work alone
- Question1 on Virtual Memory is pre-lab question for in-class Lab4
- HW Help Session *Thurs (Apr 11) and Mon (Apr 15), 6-7:30pm in B17 Upson*

Prelim3 is in *two weeks*, Thursday, April 25\(^{th}\)
- Time and Location: 7:30pm in Phillips 101 and Upson B17
- Old prelims are online in CMS
Administrivia

Next four weeks

• Week 11 (Apr 8): Lab3 due and Project3/HW4 handout
• Week 12 (Apr 15): Project3 design doc due and HW4 due
• Week 13 (Apr 22): Project3 due and Prelim3
• Week 14 (Apr 29): Project4 handout

Final Project for class

• Week 15 (May 6): Project4 design doc due
• Week 16 (May 13): Project4 due