Review

- Sequential software is slow software
  - SIMD and MIMD only path to higher performance
- Multithreading increases utilization, Multicore
  more processors (MIMD)
- Multiprocessor/Multicore uses Shared Memory
  - Cache coherency implements shared memory even with multiple copies in multiple caches
  - False sharing a concern; watch block size!
- OpenMP as simple parallel extension to C
  - Threads, Parallel for, private, critical sections, ...
  - C: small so easy to learn, but not very high level and its easy to get into trouble

Review: Strong vs Weak Scaling

- Strong scaling: problem size fixed
- Weak scaling: problem size proportional to increase in number of processors
  - Speedup on multiprocessor while keeping problem size fixed is harder than speedup by increasing the size of the problem
  - But a natural use of a lot more performance is to solve a lot bigger problem

Agenda

- Wrap up TLP
- Administrivia
- Switching Networks, Transistors
- Gates and Truth Tables for Circuits
- Boolean Algebra
- Summary

Experiment

- Try compile and run at NUM_THREADS = 64
- Try compile and run at NUM_THREADS = 64 with -O2
- Try compile and run at NUM_THREADS = 32, 16, 8, ... with -O2
32 Core: Speed-up vs. Scale-up

Speed-up: (strong scaling)  
Scale-up: Fl. Pt. Ops = 2 x Size³

<table>
<thead>
<tr>
<th>Threads</th>
<th>Time (secs)</th>
<th>Speedup</th>
<th>Time (secs)</th>
<th>Size (Dim)</th>
<th>Fl. Ops x 10⁹</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13.75</td>
<td>1.00</td>
<td>13.75</td>
<td>1000</td>
<td>2.00</td>
</tr>
<tr>
<td>2</td>
<td>13.52</td>
<td>1.00</td>
<td>13.52</td>
<td>1240</td>
<td>3.81</td>
</tr>
<tr>
<td>4</td>
<td>13.79</td>
<td>1.00</td>
<td>13.79</td>
<td>1430</td>
<td>5.85</td>
</tr>
<tr>
<td>8</td>
<td>12.55</td>
<td>1.00</td>
<td>12.55</td>
<td>1600</td>
<td>8.19</td>
</tr>
<tr>
<td>16</td>
<td>13.61</td>
<td>1.00</td>
<td>13.61</td>
<td>2000</td>
<td>16.00</td>
</tr>
<tr>
<td>32</td>
<td>13.92</td>
<td>1.00</td>
<td>13.92</td>
<td>2500</td>
<td>31.25</td>
</tr>
<tr>
<td>64</td>
<td>13.83</td>
<td>1.00</td>
<td>13.83</td>
<td>2600</td>
<td>35.15</td>
</tr>
</tbody>
</table>

Memory Capacity = f(Size²), Compute = f(Size³)

3/10/12 Spring 2012 - Lecture #16

Strong vs. Weak Scaling

Top 500 Supercomputer List

- Linpack: solve system of linear equations
  - Most time in \( Z = A \times X + Y \), where \( X, Y, Z \) are vectors
- Matrix size first 100x100, then 1000x1000
- Size too small for supercomputers
  - Let problem size increase as big as possible
- Current Fastest: Fujitsu SPARC64 (RISC)
  - 548,352 processors @ 2 GHz, 8 cores/chip
  - 10.5 PetaFLOPS (10¹⁵ Fl. Pt. Operations/Second)
- Matrix: 10,725,120 x 10,725,120 (!)

Peer Instruction: Why Multicore?
The switch in ~2004 from 1 processor per chip to multiple processors per chip happened because:

I. The “power wall” meant that no longer get speed via higher clock rates and higher power per chip
II. There was no other performance option but replacing 1 inefficient processor with multiple efficient processors
III. OpenMP was a breakthrough in ~2000 that made parallel programming easy

A)(orange) I only
B)(green) II only
C)(pink) I & II only
D)(yellow) I, II, & III
False Sharing in OpenMP
```c
{ int i; double x, pi, sum[NUM_THREADS];
#pragma omp parallel private (i,x)
{ int id = omp_get_thread_num();
  for (i=id; sum[i]=0.0; i+=NUM_THREADS) {
    x = (i+0.5)*step;
    sum[i] += 4.0/(1.0+x*x);
  }
  }  
  // What is problem?
  // Sum[0] is 8 bytes in memory, Sum[1] is adjacent 8 bytes in memory => false sharing if block size ≥ 16 bytes
```

Peer Instruction: No False Sharing
```c
{ int i; double x, pi, sum[10000];
#pragma omp parallel private (i,x)
{ int id = omp_get_thread_num(), fix = ___________; 
  for (i=id; sum[i]=0.0; i+=num_steps; i+=NUM_THREADS) { 
    x = (i+0.5)*step;
    sum[i*fix] += 4.0/(1.0+x*x);
  }
  // What is best value to set fix to prevent false sharing? 
  A)(orange) omp_get_num_threads(); 
  B)(green) Constant for number of blocks in cache 
  C)(pink) Constant for size of cache block in bytes 
  D)(yellow) Constant for size of blocks in doubles
```

Administrivia
- Need Partners for Project 3: Who has a partner? Who doesn’t?
- Part 1 due Sunday March 25 before midnight
- Homework due Sunday March 25 before midnight
- Spring Break: Prefer March 25 or April 1? — Friday March 23?

Harry Porter’s 8-bit Relay Computer
- Before switches were transistors, built using electronic vacuum tubes
- Before vacuum tubes as switches, built using were electro-mechanical replays
- Porter (not Harry Potter) is a CS Prof at Portland State
- video
  - http://web.cecs.pdx.edu/~harry/Relay/

Agenda
- Wrap up TLP
- Administrivia
- Switching Networks, Transistors
- Gates and Truth Tables for Circuits
- Boolean Algebra
- Summary

Hardware Design
- Next several weeks: we’ll study how a modern processor is built; starting with basic elements as building blocks
- Why study hardware design?
  - Understand capabilities and limitations of hw in general and processors in particular
  - What processors can do fast and what they can’t do fast (avoid slow things if you want your code to run fast!)
  - Background for more in depth hw courses (CS 152)
  - Hard to know what will need for next 30 years
  - There is just so much you can do with standard processors: you may need to design own custom hw for extra performance
  - Even some commercial processors today have customizable hardware!
Synchronous Digital Systems

Hardware of a processor, such as the MIPS, is an example of a Synchronous Digital System

Synchronous:
- All operations coordinated by a central clock
- “Heartbeat” of the system!

Digital:
- Represent All values by 2 discrete values
- Electrical signals are treated as 1’s and 0’s
- 1 and 0 are complements of each other
- High /low voltage for true / false, 1 / 0

Switches: Basic Element of Physical Implementations

- Implementing a simple circuit (arrow shows action if wire changes to “1” or is asserted):
  
  Close switch (if A is “1” or asserted) and turn on light bulb (Z)
  
  Open switch (if A is “0” or unasserted) and turn off light bulb (Z)

Z = A

Switches (cont’d)

- Compose switches into more complex ones (Boolean functions):
  
  AND
  
  OR

Historical Note

- Early computer designers built ad hoc circuits from switches
- Began to notice common patterns in their work: ANDs, ORs, ...
- Master’s thesis (by Claude Shannon) made link between work and 19th Century Mathematician George Boole — Called it "Boolean" in his honor
- Could apply math to give theory to hardware design, minimization, ...

Transistors

- High voltage (V_{out}) represents 1, or true
- Low voltage (0 volts or Ground) represents 0, or false
- Let threshold voltage (V_{th}) decide if a 0 or a 1
- If switches control whether voltages can propagate through a circuit, can build a computer
- Our switches: CMOS transistors

CMOS Transistor Networks

- Modern digital systems designed in CMOS
  — MOS: Metal-Oxide on Semiconductor
  — C for complementary: use pairs of normally-open and normally-closed switches
- CMOS transistors act as voltage-controlled switches
  — Similar, though easier to work with, than relay switches from earlier era (Porter computer)
CMOS Transistors

- Three terminals: source, gate, and drain
- Switch action: if voltage on gate terminal is (some amount) higher/lower than source terminal then conducting path established between drain and source terminals (switch is closed)

n-channel transistor:
open when voltage at gate is low
closed when voltage at gate is low
when voltage(Gate) > voltage (Threshold)

p-channel transistor:
open when voltage at gate is low
closed when voltage at gate is low
when voltage(Gate) > voltage (Threshold)

Note circle symbol to indicate "NOT" or "complement"

CMOS circuit rules

- Don't pass weak values => Use Complementary Pairs
  - N-type transistors pass weak 3s (V<sub>ds</sub>, V<sub>gs</sub>)
  - N-type transistors pass strong 0's (ground)
- Use N-type transistors only to pass 0's (N for negative)
- Converse for P-type transistors: Pass weak 0s, strong 1s
  - Pass weak 0's (V<sub>ds</sub>), strong 1's (V<sub>gs</sub>)
  - Use P-type transistors only to pass 1's (P for positive)
- Use pairs of N-type and P-type to get strong values
  - Never leave a wire undriven
  - Make sure there's always a path to V<sub>dd</sub> or gnd
  - Never create a path from V<sub>dd</sub> to gnd (ground)

MOS Networks

n-channel transistor:
closed when voltage at gate is low
open when:
voltage(Gate) > voltage (Threshold)
when voltage at gate is low
voltage(Gate) > voltage (Threshold)

p-channel transistor:
closed when voltage at gate is low
open when:
voltage(Gate) > voltage (Threshold)
when voltage at gate is low
voltage(Gate) > voltage (Threshold)

Called an invertor or not gate

Two Input Networks

what is the relationship between x, y and z?

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Student Roulette

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Two Input Networks: Peer Instruction

what is the relationship between x, y and z?

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Called NAND gate (NOT AND)

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Type of Circuits

- **Synchronous Digital Systems** consist of two basic types of circuits:
  - Combinational Logic (CL) circuits
    - Output is a function of the inputs only, not the history of its execution
    - E.g., circuits to add A, B (ALUs)
  - Sequential Logic (SL)
    - Circuits that "remember" or store information
    - aka "State Elements"
    - E.g., memories and registers (Registers)

Truth Tables

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>F(0,0,0,0)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F(0,0,0,1)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>F(0,0,1,0)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>F(0,0,1,1)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>F(0,1,0,0)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>F(0,1,0,1)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>F(0,1,1,0)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>F(0,1,1,1)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>F(1,0,0,0)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F(1,0,0,1)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>F(1,0,1,0)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>F(1,0,1,1)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>F(1,1,0,0)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>F(1,1,0,1)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>F(1,1,1,0)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>F(1,1,1,1)</td>
</tr>
</tbody>
</table>

Truth Table Example #1:
y = F(a,b): 1 iff a ≠ b

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Truth Table Example #2:
2-bit Adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Truth Table Example #3:
32-bit Unsigned Adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Truth Table Example #4:
3-input Majority Circuit

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

This is called *Sum of Products* form; just another way to represent the TT as a logical expression.

More simplified forms (fewer gates and wires)
**Design Hierarchy**

- System
- Control
- Code registers
- Multiplexer
- Comparator
- State registers
- Combinational logic
- Switching networks

**Combinational Logic Symbols**

- Common combinational logic systems have standard symbols called logic gates
  - Buffer, NOT
    - \( A \xrightarrow{\text{NOT}} Z \)
  - AND, NAND
    - \( A \& B \xrightarrow{\text{NAND}} Z \)
  - OR, NOR
    - \( A \oplus B \xrightarrow{\text{NOR}} Z \)

**Boolean Algebra**

- Use plus for OR
  - "logical sum"
- Use product for AND (a*b or implied via ab)
  - "logical product"
- "Hat" to mean complement (NOT)
- Thus
  
  \[
  \begin{align*}
  ab + a + \bar{c} &= a*b + a + \bar{c} \\
  &= (a \text{ AND } b) \text{ OR } a \text{ OR } (\text{NOT } c)
  \end{align*}
  \]

**Booleans Algebra: Circuit & Algebraic Simplification**

- Boolean Algebraic Simplification

**Laws of Boolean Algebra**

<table>
<thead>
<tr>
<th>Law</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \cdot \bar{x} = 0 )</td>
<td>( x + \bar{x} = 1 )</td>
</tr>
<tr>
<td>( x \cdot 0 = 0 )</td>
<td>( x + 1 = 1 )</td>
</tr>
<tr>
<td>( x \cdot 1 = x )</td>
<td>( x + 0 = x )</td>
</tr>
<tr>
<td>( x \cdot x = x )</td>
<td>( x + x = x )</td>
</tr>
<tr>
<td>( x \cdot y = y \cdot x )</td>
<td>( x + y = y + x )</td>
</tr>
<tr>
<td>( (xy)z = x(yz) )</td>
<td>commutativity</td>
</tr>
<tr>
<td>( (x+y)+z = x+(y+z) )</td>
<td>associativity</td>
</tr>
<tr>
<td>( x(y+z) = xy+xz )</td>
<td>( x + y + z = (x+y)(x+z) )</td>
</tr>
<tr>
<td>( xy + x = x )</td>
<td>( (x+y)x = x )</td>
</tr>
<tr>
<td>( xy + y = x + y )</td>
<td>( (x+y)x = xy )</td>
</tr>
<tr>
<td>( \bar{x}y = x + \bar{y} )</td>
<td>( (x+y)x = xy )</td>
</tr>
<tr>
<td>( x + y = x \cdot y )</td>
<td>( x + y = x \cdot y )</td>
</tr>
</tbody>
</table>

**Boolean Algebra Simplification Example**

\[
y = ab + a + c
\]
### Boolean Algebraic Simplification

**Example**

\[
y = ab + a + c \\
= a(b + 1) + c \quad \text{distribution, identity} \\
= a(1) + c \quad \text{law of 1's} \\
= a + c \quad \text{identity}
\]

### Summary

- Real world voltages are analog, but are quantized to represent logic 0 and logic 1
- Transistors are just switches, combined to form gates: AND, OR, NOT, NAND, NOR
- Truth table can be mapped to gates for combinational logic design
- Boolean algebra allows minimization of gates