Computer Science 246
Advanced Computer Architecture
Spring 2008
Harvard University

Instructor: Prof. David Brooks
dbrooks@eecs.harvard.edu
## Analysis Abstraction Levels

<table>
<thead>
<tr>
<th>Abstraction Level</th>
<th>Analysis Capacity</th>
<th>Analysis Accuracy</th>
<th>Analysis Speed</th>
<th>Analysis Resources</th>
<th>Energy Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application (Behavioral)</td>
<td>Most</td>
<td>Worst</td>
<td>Fastest</td>
<td>Least</td>
<td>Most</td>
</tr>
<tr>
<td>Architectural (RTL)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logic (Gate)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transistor (Circuit)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Least</td>
</tr>
</tbody>
</table>

- Application Behavior: Least to Best
- Architectural RTL: Least to Best
- Logic Gate: Least to Best
- Transistor Circuit: Least to Best
Modeling Hierarchy and Tool Flow

- Early analytical performance models
  - Trace/exec-driven, cycle-accurate simulation models
  - Microarch parms/specs
- Circuit-level (hierarchical) netlist model
  - Gate-level model (if synthesized)
- Layout-level physical design model
  - Cap extract, sim
- RTL MODEL (VHDL)
  - RTL sim
- Performance Test Cases
  - edit/debug
- (Architectural) Sim Test Cases
  - edit/debug
- Bitvector test cases
  - edit/tune/debug
- Design rules
  - design rule check, validate

Energy Models

set of workloads

edit/debug
Processor simulators: An overview

- Performance simulator: a tool that emulates the behavior of a real processor
  - Software:
    - Concept phase: C/C++/System C
    - Early design phase: VHDL
  - Hardware:
    - FPGA

- Simulators are used for:
  - Workload characterization
  - Performance / power target projection
  - Design space exploration and trade-off evaluation
  - Testing / debugging/ validation

- Existing simulators
  - Academia simulators: SimpleScalar, RSIM, etc
  - Industrial simulators:
    - Concept phase
    - Product phase
Example: Turandot

- An out-of-order superscalar processor model for the PowerPC architecture
  - Power4-like machine configuration by default
  - Other configurations attainable through compile-time parameters
- Performance model validated against Power4 product model

```
program trace 1
...
program trace N
```

```
Turandot
```

```
trace-driven mode
```

```
program binary
program inputs
```

```
Aria
```

```
trace segment
trace request
```

```
Turandot
```

```
execution-driven mode
```
Simulation Framework

Fetch
	- Decode/Expand
	- Rename/Dispatch
	- Issue
	- Reg Read
	- Exec/Mem
	- WB
	- Retire
stores are committed to cache/memory COMMIT_STORES_DELAY cycles after retire

• If store, remove from storeq
• If load, remove from reorderq, check if there is a load/store conflict. If yes, flush the pipeline (reset.macros)
• Update branch history
• Remove instruction from retireq

• Rename
  • Check if enough rename registers available, if not, stall until available.
  • Rename architectural registers to physical registers.
  • If the instruction is a mispredicted branch instruction, check if all operands are ready. If yes, resolve the branch and start fetching from the right path from next cycle.
  • Note: registers in different class are renamed separately.

• Dispatch
  • Place renamed IOPs into the corresponding issue queue. If a given operation cannot be placed in the issue queue (i.e. the queue is full), stall the stage until available.
Power/Performance abstractions

• Low-level:
  • Hspice
  • PowerMill

• Medium-Level:
  • RTL Models

• Architecture-level:
  • PennState SimplePower
  • Intel Tempest
  • Princeton Wattch
  • IBM PowerTimer
  • Umich/Colorado PowerAnalyzer
Low-level models: Hspice

- Extracted netlists from circuit/layout descriptions
  - Diffusion, gate, and wiring capacitance is modeled
- Analog simulation performed
  - Detailed device models used
  - Large systems of equations are solved
  - Can estimate dynamic and leakage power dissipation within a few percent
  - Slow, only practical for 10-100K transistors
- PowerMill (Synopsys) is similar but about 10x faster
Medium-level models: RTL

- Logic simulation obtains switching events for every signal
- Structural VHDL or verilog with zero or unit-delay timing models
- Capacitance estimates performed
  - Device Capacitance
    - Gate sizing estimates performed, similar to synthesis
  - Wiring Capacitance
    - Wire load estimates performed, similar to placement and routing
- Switching event and capacitance estimates provide dynamic power estimates
Architecture level models

- **Two major classes:**
  - Cycle/Event-Based: Arch. Level power models interfaced with cycle-driven performance simulation
  - Instruction-Based: Measurement/Characterization based on instruction usage and interactions

- **Components of Arch. Level power model**
  - Could be based on ckt schematic measurements/extrapolation
  - Capacitance models
  - Both may need to consider:
    - Circuit design styles
    - Clock gating styles & Unit usage statistics
    - Signal transition statistics
Architecture level models

Power \sim \frac{1}{2} CV^2 Af

- **Analytical Approach:**
  - Estimate “CV^2f” via analytical models
  - Tools: Wattch, PowerAnalyzer, Tempest (mixed-mode)

- **Empirical Approach**
  - Estimate “CV^2f” via empirical measurements
  - Tools: PowerTimer, AccuPower, Internal Industrial Tools

- Estimate “A” via statistics from architectural-performance simulators
Analytical Modeling Tools: Modeling Capacitance

- Requires modeling wire length and estimating transistor sizes
- Related to RC Delay analysis for speed along critical path
  - But capacitance estimates require summing up all wire lengths, rather than only an accurate estimate of the longest one.
Register File: Capacitance

\[ C_{\text{wordline}} = C_{\text{diffcapWordlineDriver}} + \text{NumberBitlines} \times C_{\text{gatecapN1}} + \text{Wordlinelength} \times C_{\text{metal}} \]

\[ C_{\text{bitline}} = C_{\text{diffcapPchg}} + \text{NumberWordlines} \times C_{\text{diffcapN1}} + \text{Bitlinelength} \times C_{\text{metal}} \]
Register File Model: Validation

- Validated against a register file schematic used in Intel’s Merced design
- Compared capacitance values with estimates from a layout-level Intel tool
- Interconnect capacitance had largest errors
  - Model currently neglects poly connections
  - Differences in wire lengths -- difficult to tell wire distances of schematic nodes

<table>
<thead>
<tr>
<th>Error Rates</th>
<th>Gate</th>
<th>Diff</th>
<th>InterConn.</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wordline (r)</td>
<td>1.11</td>
<td>0.79</td>
<td>15.06</td>
<td>8.02</td>
</tr>
<tr>
<td>Wordline (w)</td>
<td>-6.37</td>
<td>0.79</td>
<td>-10.68</td>
<td>-7.99</td>
</tr>
<tr>
<td>Bitline (r)</td>
<td>2.82</td>
<td>-10.58</td>
<td>-19.59</td>
<td>-10.91</td>
</tr>
<tr>
<td>Bitline (w)</td>
<td>-10.96</td>
<td>-10.60</td>
<td>7.98</td>
<td>-5.96</td>
</tr>
</tbody>
</table>

(Numbers in Percent)
Architecture level models: Signal Transition Statistics

- Dynamic power is proportional to switching
- How to collect signal transition statistics in architectural-level simulation?
  - Many signals are available, but do we want to use all of them?
  - One solution (register file):
    - Collect statistics on the important ones (bitlines)
    - Infer where possible (wordlines)
    - Assign probabilities for less important ones (decoders)
**Architecture level models: Clock Gating: What, why, when?**

- **Dynamic Power is dissipated on clock transitions**
- **Gating off clock lines when they are unneeded reduces activity factor**
- **But putting extra gate delays into clock lines increases clock skew**
- **End results:**
  - Clock gating complicates design analysis but saves power.
Wattch: An Overview

Overview of Features

- Parameterized models for different CPU units
  - Can vary size or design style as needed
- Abstract signal transition models for speed
  - Can select different conditional clocking and input transition models as needed
- Based on SimpleScalar (has been ported to many simulators)
- Modular: Can add new models for new units studied

Wattch’s Design Goals

- Flexibility
- Planning-stage info
- Speed
- Modularity
- Reasonable accuracy
Unit Modeling

**Modeling Capacitance**
- Models depend on structure, bitwidth, design style, etc.
- E.g., may model capacitance of a register file with bitwidth & number of ports as input parameters

**Modeling Activity Factor**
- Use cycle-level simulator to determine number and type of accesses
  - reads, writes, how many ports
- Abstract model of bitline activity
### One Cycle in Wattch

<table>
<thead>
<tr>
<th>Fetch</th>
<th>Dispatch</th>
<th>Issue/Execute</th>
<th>Writeback/Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Power (Units Accessed)</strong></td>
<td><strong>I-cache</strong>&lt;br&gt;<strong>Bpred</strong></td>
<td><strong>Rename Table</strong>&lt;br&gt;<strong>Inst. Window</strong>&lt;br&gt;<strong>Reg. File</strong></td>
<td><strong>Inst. Window</strong>&lt;br&gt;<strong>Reg File</strong>&lt;br&gt;<strong>ALU</strong>&lt;br&gt;<strong>D-Cache</strong>&lt;br&gt;<strong>Load/St Q</strong>&lt;br&gt;<strong>Result Bus</strong>&lt;br&gt;<strong>Reg File</strong>&lt;br&gt;<strong>Bpred</strong></td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td><strong>Cache Hit?</strong>&lt;br&gt;<strong>Bpred Lookup?</strong></td>
<td><strong>Inst. Window Full?</strong></td>
<td><strong>Dependencies Satisfied?</strong>&lt;br&gt;<strong>Resources?</strong>&lt;br&gt;<strong>Commit Bandwidth?</strong></td>
</tr>
</tbody>
</table>

- **On each cycle:**
  - determine which units are accessed
  - model execution time issues
  - model per-unit energy/power based on which units used and how many ports.
Units Modeled by Wattch

- **Array Structures**
  - Caches, Reg Files, Map/Bpred tables

- **Content-Addressable Memories (CAMs)**
  - TLBs, Issue Queue, Reorder Buffer

- **Complex combinational blocks**
  - ALUs, Dependency Check

- **Clocking network**
  - Global Clock Drivers, Local Buffers
PowerTimer

- Circuit Power Data (Macros)
- TechParms
- uArchParms
- Program Executable or Trace

SubUnit Power = f(SF, uArch, Tech)

Compute Sub-Unit Power

AF/SF Data

Architectural Performance Simulator

CPI

Power
PowerTimer: Empirical Unconstrained Power

Pre-silicon, POWER4-like superscalar design
PowerTimer: Energy Models

- Energy models for uArch structures formed by summation of circuit-level macro data

\[
\text{Power} = C1 \cdot SF + \text{HoldPower} \\
\text{Power} = C2 \cdot SF + \text{HoldPower} \\
\text{Power} = Cn \cdot SF + \text{HoldPower}
\]
Empirical Estimates with CPAM

• Estimate power under “Input Hold” and “Input Switching” Modes

• Input Hold: All Macro Inputs (Except Clocks) Held
  • Can also collect data for Clock Gate Signals

• Input Switching: Apply Random Switching Patterns with 50% Switching on Input Pins

Macro Inputs → Macro

• 0% Switching (Hold Power)
• 50% Switching Power
Example: Fixed Point Issue Queue

- Made up of 5 macros
Assumption: Power linearly dependent on Switching Factor
This separates Clock Power and Switching Power

At 0% SF, Power = Clock Power (significant without clock gating)
Key Activity Data

- **SF => Moves along the Switching Power Curve**
  - Estimated on a per-unit basis from RTL Analysis
- **AF => Moves along the Clock Power Curve**
  - Extracted from Microarchitectural Statistics (Turandot)
Microarchitectural Statistics

- Stats are similar to tracking used in Wattch, etc
- Differences:
  - Clock Gating Modes (3 modes)
  - Customized Scaling Based on Circuit Style (4 styles)
- Clock Gating Modes:
  - $P_{\text{constrained}} = P_{\text{unconstrained}}$ (not clock-gateable)
  - $P_{\text{constrained}_1} = AF \cdot (P_{\text{clock}} + P_{\text{logic}})$ (common)
  - $P_{\text{constrained}_2} = AF \cdot P_{\text{clock}} + P_{\text{logic}}$ (rare)
  - $P_{\text{constrained}_3} = P_{\text{clock}} + AF \cdot P_{\text{logic}}$ (very rare)
- Scaling Based on Circuit Styles
  - $AF_1 = \#\text{valid}$ (Latch-and-Mux, No Stall Gating)
  - $AF_2 = \#\text{valid} - \#\text{stalls}$ (Latch-and-Mux, With Stall Gating)
  - $AF_3 = \#\text{writes}$ (Arrays that only gate updates)
  - $AF_4 = \#\text{writes} + \#\text{reads}$ (Arrays, RAM Macros)
Clock Gating Modes: Valid-Bit Gating

- Latch-Based Structures: Execute Pipelines, Issue Queues
Clock Gating Modes

- \( P_{\text{constrained}_1} = AF \times (P_{\text{clock}} + P_{\text{logic}}) \)

- \( P_{\text{constrained}_2} = AF \times P_{\text{clock}} + P_{\text{logic}} \)
Scaling Options: Valid-bit Gating, what about Stalls?

- **Option 1: Stalls cannot be gated**

- **Option 2: Stalls can be gated**
Scaling Options: Array Structures

- **Option 1: Reads and Writes Eligible to Gate for Power**
Scaling Options: Array Structures

- Option 2: Only Writes Eligible to Gate for Power
## 12 Clock Gating Modes

<table>
<thead>
<tr>
<th>Gating Mode</th>
<th>Valid</th>
<th>Valid+ Stalls</th>
<th>Writes</th>
<th>Writes+ Reads</th>
<th>Gate Both</th>
<th>Gate Clock</th>
<th>Gate Logic</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Control Logic, Buffers, Small Macros</td>
</tr>
<tr>
<td>1</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Issue Queues, Execute Pipelines</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Caches</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Some Queues</td>
</tr>
<tr>
<td>4</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>CAMs, Selection Logic</td>
</tr>
<tr>
<td>5</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>CAMs, Selection Logic</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>CAMs, Selection Logic</td>
</tr>
<tr>
<td>7</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No Known macros</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No Known macros</td>
</tr>
<tr>
<td>9</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No Known macros</td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No Known macros</td>
</tr>
<tr>
<td>11</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No Known macros</td>
</tr>
<tr>
<td>12</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No Known macros</td>
</tr>
</tbody>
</table>
PowerTimer Observations

- PowerTimer works well for POWER4-like estimates and derivatives
  - Scale base microarchitecture quite well
  - E.g. optimal power-performance pipelining study
  - Lack of run-time, bit-level SF not seen as a problem within IBM (seen as noise)
    - Chip bit-level SFs are quite low (5-15%)
    - Most (60-70%) power is dissipated while maintaining state (arrays, latches, clocks)
    - Much state is not available in early-stage timers
Comparing models: Flexibility

- Flexibility necessary for certain studies
  - Resource tradeoff analysis
  - Modeling different architectures
- Purely analytical tools provides fully-parameterizable power models
  - Within this methodology, circuit design styles could also be studied
- PowerTimer scales power models in a user-defined manner for individual sub-units
  - Constrained to structures and circuit-styles currently in the library
- Perhaps Mixed Mode tools could be very useful
Comparing power models: Accuracy

• **PowerTimer -- Based on validation of individual pieces**
  • Extensive validation of the performance model (AFs)
  • Power estimates from circuits are accurate
  • Circuit designers must vouch for clock gating scenarios
  • Certain assumptions will limit accuracy or require more in-depth analysis

• **Analytical Tools**
  • Inherent Issues
    – Analytical estimates cannot be as accurate as SPICE analysis (“C” estimates, \( CV^2 \) approximation)
  • Practical Issues
    – Without industrial data, must estimate transistor sizing, bits per structure, circuit choices
Comparing power models: Speed

- Performance simulation is slow enough!
- Post-Processing vs. Run-Time Estimates
- Wattch’s per-cycle power estimates: roughly 30% overhead
  - Post-processing (per-program power estimates) would be much faster (minimal overhead)
- PowerTimer allows both no overhead post-processing and run-time analysis for certain studies (di/dt, thermal)
  - Some clock gating modes may require run-time analysis
- Third Option: Bit Vector Dumps
  - Flexible Post-Processing ↔ Huge Output Files
Future Modeling Efforts: Building Block Models

- Building block methodology can provide flexibility with ease of validation (and better leakage)
- Develop/Validate models for common intrinsic blocks (latch, mux, interconnect, etc)
  - Chain building blocks together to model higher level structures (issue queue, register file)
  - Works well for static and dynamic power

Building Blocks

- Local Clock Buffer
- Interconnect
- Latch Bit

Queue Structure

 clk
 data
Future Modeling Efforts: Different Circuit Design Styles

- Tools require the circuit design style assumptions
  - Static vs. Dynamic logic
  - Single vs. Double-ended bitlines in register files/caches
  - Sense Amp designs
  - Transistor and buffer sizings

- Generic solutions are difficult because many styles are popular

- Within individual companies, circuit design styles may be more stable