
Announcements: EE282 End Game

- HW3 is due on 12/1
- PA2 is due on 12/3
- Quiz 2 on 12/8, 8.30 – 11.30am, Skilling Auditorium & Skilling 191
 - All lectures, closed book, 1 page of notes, calculator
 - The duration of the exam may be <3h
 - Read the cover page of the exam carefully
 - Will provide a practice quiz
 - Attend Quiz 2 review session on Friday 12/5

Lecture 17:

Reliability & Energy Efficiency

Department of Electrical Engineering
Stanford University

<http://eeclass.stanford.edu/ee282>

Review: Reliable Systems

- Faults \ error \ failure
 - Failures are observable by end users
 - Not all faults lead to failures
- Important metrics: MTTF & MTTR
 - Improvements on either improves availability
- Basic ideas in fault-tolerance
 - Reliability through redundancy
 - In resources or time
 - Reliability Vs. cost and performance
 - Watch out for single points of failure

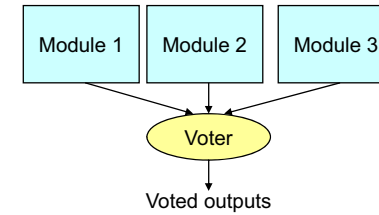
Review: Reliability in Storage, Memory, and Interconnects

- Reliability in storage: RAID
 - A collection of disks that behaves like a single disk with
 - Higher capacity, higher bandwidth, higher reliability
 - Key idea in RAID: error detection code across disks
- Reliability in memory: parity and ECC
 - Extra storage for error correcting or error detecting codes
 - Can do coding within chip or across chips/DIMMs (chipkill)
- Reliability in interconnects
 - Transient faults: error detection & retransmission
 - Permanent faults: path diversity

Reliability in Computing

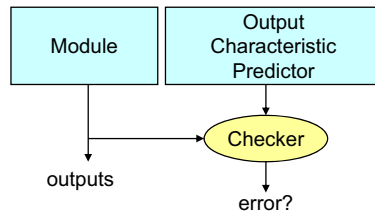
- For systems ranging from ALUs to full computers
- Not as simple (or cost effective) as with storage/memory
- Techniques
 - Triple module redundancy (TMR)
 - Or more than triple...
 - Concurrent error detection (CED)
 - In various hardware & software forms
 - Checkpoint & restore
 - Also called Backward Error Recovery (BER)
- Consider two issues: detection & recovery
 - Not always clearly separated...

Triple Module Redundancy (TMR)



- Advantages
 - Can fully mask 1 failed component
 - Any type of fault
 - Generalizes to N-MR (can tolerate $[N/2-1]$ faulty components)
- Disadvantages
 - High implementation cost
 - Reliability of voter?
 - Synchronization of modules?

Concurrent error detection (CED)



- CED idea
 - Used a 2nd module to detect errors
 - Any type of fault again
 - Once error is detected, use re-execution for correction
 - Works with many transient errors
 - Repeated errors signal a more systematic problem

CED Details

- What can the 2nd module be
 - A replica of the computation module
 - Checker is an equality comparator
 - A parity predictor
 - Checker compares actual to predicted parity
 - Many others
- Common mode errors
 - An error affects both modules the same way
 - Solutions: design diversity
 - Use different implementations of same function
 - N version programming
 - Use different processors (in Boeing and Airbus)

TMR and CED Issues

- At what level do you apply CED?
 - Compare ALU outputs
 - Compare Register updates
 - Compare L1/L2/L3 traffic?
- Bandwidth requirement vs. detection latency tradeoff
 - If you compare all ALU outputs: fast detects, need lots of BW
 - If you compare all L2 cache traffic: low BW, delayed detection
- A solution
 - Hash all chip updates using a CRC code (e.g. 16-bit)
 - Compare CRCs across two modules on every cycle
 - Low probability of masking an error
 - Low bandwidth usage
 - Fast fault detection (but coarse-grain)

Checkpoint & Restore (Backward Error Correction)

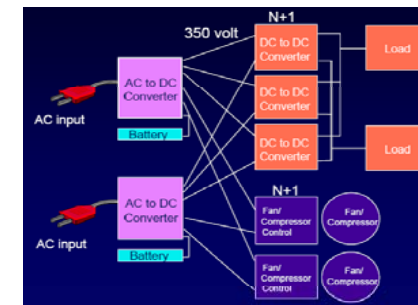
- Basic idea
 - Periodically get a system state checkpoint (registers, memory, ...)
 - Checkpoint must be stored in “protected storage”
 - During execution, try to detect errors
 - On error, roll back to checkpointed state and restart
 - Works for transient faults
 - For permanent errors, first reconfigure and then restore
- Advantages
 - A general concept applicable to many systems and types of errors
- Disadvantage
 - Recovery carries a performance penalty
 - Not ideal when there is long error detection latency
 - If detection time > checkpoint interval
 - Some part of the system state may be unrecoverable
 - E.g., if an error affects an external state not checkpointed

Creating Checkpoints of System State

- Naïve approach: make a complete copy of system state
 - Takes too much time and too much space
- Incremental checkpoints
 - Keep track (log) of state changes since previous checkpoint
 - Checkpoint entries: address, old value, new value
 - On fault, undo changes to establish checkpoint
 - Reduces space & time requirements to restore
- Checkpoint challenges (particularly in parallel systems)
 - Checkpoint frequency vs detection latency
 - Number of active checkpoints
 - Synchronization (may need to stop the world)
 - Dirty data in caches

Reliability for the Full System

- IO, cooling, power supply...
 - N+1 redundancy (no single point of failure)
 - Service processor to monitor control system health
- Power subsystem example



Energy Efficient Systems

Why is Energy Efficiency Important

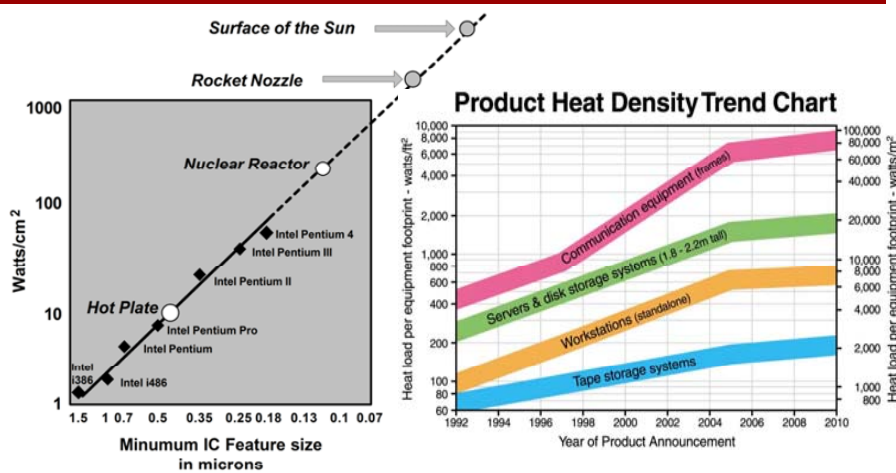
- Always important in mobile devices because of battery life
 - Battery technology not improving fast enough
 - Power consumption constrains circuit designers, architects, mechanical designers, application developers and service providers
 - Innovations: lower-power chips, displays, UIs, storage, wireless protocols...



- What about tethered computers (personal and enterprise)?

Power Density Trends

Power Density Trends



Source: Borkar (Intel), 1999

Source: Uptime Institute

- Reasons for power increase trajectory
 - Smaller feature sizes and increased compaction = greater power density
 - Faster devices (clock frequency)
 - More complicated components
 - OOO processors, complicated speculation/prediction...

Where Do the Trends Stop?

- Electricity costs
 - Google building data centers next to dams!
 - Gets worse with larger data centers
 - 1000 rack data center ~ 10 million dollars a year
 - Cooling system (fans, AC, etc) uses power too
 - 0.5W-1W of cooling per W of computer power used in data centers
- Power delivery, packaging, and cooling cost
 - Increased cost of thermal packing: \$1/W for CPUs > 35W [Tiwari, DAC98]
- Reliability/thermal issues

Where Do the Trends Stop?

- Environmental concerns
 - Compute equipment energy use: 22M GJ + 3.9M tons CO²
 - EnergyStar (US), TopRunner (Japan), FOE rating (Switzerland),...
 - "...goal to increase computer energy efficiency by 85% by 2005." Japan's "TopRunner" energy program, 2002
- Increasing power density limits compaction



Power Density and Compaction (Example from Google's 2003 Paper)

- Computer with mid-range 1.4-GHz Pentium III CPUs draws ~90 W (DC)
 - 55 W for the two CPUs, 10 W for a disk drive, and 25 W to power DRAM and the motherboard
- Assuming an ATX power supply with 75% (high!) efficiency => 120W (AC)
 - A rack is ~10KW
- At 25 ft² per rack=> 400 W/ft².
 - Unfortunately, the typical power density of data centers is 70 to 150 W/ft²
- Implications
 - May not be able to fully utilize racks
 - Incentive to use lower-power components
 - But watch out for application performance (watts/unit of performance)
 - Watch out for cost of lower power components

Metrics

- Energy (Joules) = Power (Watts) * Time (sec)
 - Power is limited by infrastructure
 - Power supplies, data center design, ...
 - Energy: what the utility company charges for or the battery can store
- Power density = Power/area
- Power consumption metrics: Energy-delay, energy-delay-squared
 - Different ways to tradeoff performance for power savings

Sources of Power Consumption

$$P = C * V_{DD}^2 * F_{0 \rightarrow 1} + T_{sc} * V_{DD} * I_{peak} * F_{0 \rightarrow 1} + V_{DD} * I_{leakage}$$

- Dynamic or active power consumption
 - Charging and discharging capacitors
 - Depends on switching activity
- Short circuit currents
 - Short circuit path between supply rails during switching
 - Depends on the size of the transistors
- Leakage current or static power consumption
 - Leaking diodes and transistors
 - Gets worse with smaller devices and lower V_{DD}
 - Quickly becoming a major factor
 - Gets worse with higher temperatures

Reducing Power at All Levels

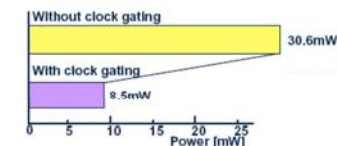
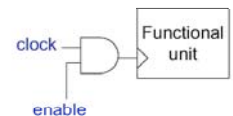
- Process technology
 - Examples: reduce capacitances, provide low/high V_t transistors
- Circuits design (gates and blocks)
 - Example: reduce switching probability, use low/high V_t transistors, use multiple supply voltages
- Architecture
 - Examples: simpler architectures, use of parallelism
- Software
 - Examples: manage voltage/frequency scaling, better algorithms ...
- Some common principles
 - Reduce redundant work/components
 - Turn off reduced components
 - Pick the implementation that best matches constraints

Circuit-level Low Power Techniques Overview

	Constant Throughput/Latency		Variable Throughput/Latency	
Power	Design Time	Sleep Mode	Run Time	
Active	Logic design Low V _{DD} Multi-V _{DD} Gate sizing Pipeline	Clock gating	DVS, DFS	
Leakage	Stack effects Multi-V _{th}	Sleep transistors Multi-V _{DD} , V _{th} Input control	Variable V _{th}	

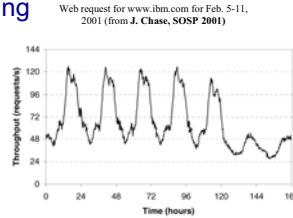
Clock Gating

- Reduces dynamic power on clock network
 - One of the biggest power contributors
 - Can be done at various granularities
 - Whole core to a set of registers
 - Tradeoff: overhead vs benefit
- Challenge: enable signal generation
 - Control logic complexity
 - Timing issues to avoid glitches
- Example: power gating efficiency for MPEG-4 decoder [ISSCC'02]



Dynamic Voltage and Frequency Scaling

- Idea: consume as much as you need
 - Use lowest frequency that achieves perf. target
 - Use lowest Vdd that allows that clock frequency
- Regulated by software based on system load
 - Heavy load: frequency, voltage, power high
 - Light load: frequency, voltage, power low
 - Trade-off: power savings vs overhead of scaling
- Example: Transmeta's LongRun technology
 - 32 levels of frequency and Vdd
 - 200MHz – 700MHz, 1.1V-1.6V
 - 20 usec to change to next level
- Simulation results: 9-38% CPU energy savings for web workloads

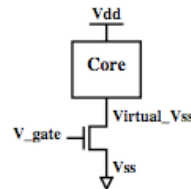


DVFS and Concurrency

- Assume an initial processor with $P_{ref} = C_{ref} * V_{ref}^2 * F_{ref}$
- Two-core system
 - Assume that applications get ideal speedup (2x)
 - We reduce the frequency by a factor of 2 ($F_{tc} = F_{ref}/2$)
 - This allows us to drop voltage by 1.7x ($V_{tc} = V_{ref}/1.7$)
 - Capacitance is higher ($C_{tc} = 2xC_{ref}$)
- Overall results: $P_{tc} = 0.34 * P_{ref}$, same application performance
 - Can get similar results with pipelining or other forms of parallel execution

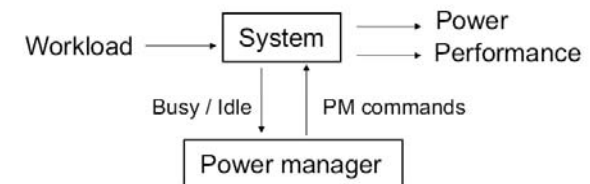
Power Gating

- Idea: turn off power completely to unused components (e.g., cores)
 - Minimizes leakage power on unused component
- Challenges
 - Area power consumption of power gate
 - High
 - Latency of transition (10s msec)
 - Saving SW state, flushing dirty cache lines, turning off clock tree
 - Carefully done to avoid voltage spikes or memory bottlenecks
- Opportunities
 - Use thermal headroom to overclock other cores...
 - Improves single-thread performance



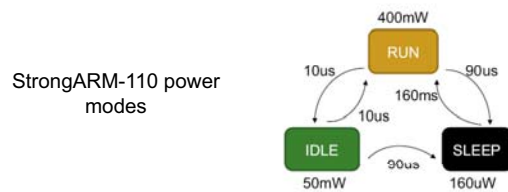
Power Management

- Selectively set the power state of HW components to match software requirements
 - Exploit workload fluctuations
 - Done by the OS, the compiler, and/or the user
 - Trade-off: power saving versus speed of resuming



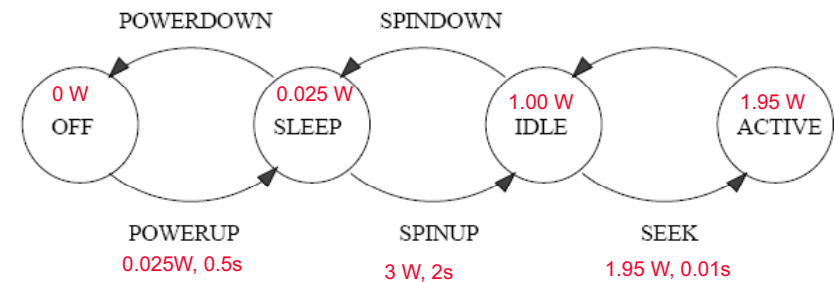
Example: Processor Power States

- Example processor modes
 - Idle mode (short periods of inactivity – few hundred mW)
 - Clock distribution turned off; needs a few cycles to recover
 - Nap mode (medium periods of inactivity – 100mW)
 - Caches turned off
 - Sleep mode (long periods of inactivity – 20 mW)
 - Turn-off power to processor (including PLL)



Example: Disk Drive Power Modes

- Common optimization for mobile computers: stop spinning the disk when it is unused for a certain period of time
- Sample: Toshiba notebook drive



State diagram from Li, 1993.

Advanced Configuration and Power Interface (ACPI)

- A standard for power management of systems
 - Describes power stages for system, devices, cores, ...
 - Vendors may implement subset of states/options
 - Defines interface for SW to query and manage power states
- Global system states:
 - G0: working - system is responsive, user applications run
 - G1: sleeping - appears to be off, OS may still be silently running
 - Multiple S-states (sleep states) defined within G1 with different wake latencies/power consumption/retention of context
 - G2: soft off - neither user nor OS is running
 - G3: hard (mechanical) off

Advanced Configuration and Power Interface (ACPI)

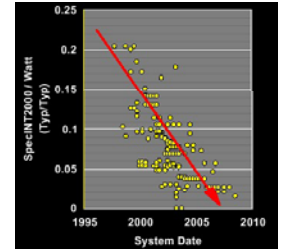
- Device states
 - D0 fully-on operating state
 - D1 and D2 are intermediate states (vary by design)
 - D3 is a powered off state (device is unresponsive)
- Processor states (P and C)
 - P states are DVFS stages
 - C0 is fully on
 - C1 to C3 are various idle modes
 - Clock may be stopped but state is maintained
 - C4 and beyond are various power off states
 - First the caches, then cores, and finally the whole chip

Trade-off power for performance

- Optimize architecture for energy-delay product
 - Energy benefits can be at expense of performance
 - E.g.: when battery is low, video rendering at 20fps instead of 30fps
 - E.g.: data center power failure means lower QoS for workloads
 - Same techniques for turn-off/turn-down can be used
 - Allows for other optimizations
 - E.g.: energy-efficient but performance-inefficient algorithms
 - E.g.: find inflection points in power-performance-efficiency curves

Reducing Power with Simpler Hardware

- Complexity = higher power
 - Diminishing returns from power budget for higher performance
 - Pipeline depth
 - More stages increase register count and clock load
 - ILP
 - Parallel execution units, dependence check logic, VLIW instruction fetch increase power
 - Speculation
 - Speculation hardware burns power (predictors, etc.)
 - Guessing wrong wastes power
 - Caches
 - Large/complicated caches burn power to access
- Example
 - Pentium 4: 65W
 - OOO, 3-way issue, 22 pipeline stages, 12K \$I, 8K \$D, 512K \$L2, 4K entry predictor, 211 mm², 0.13um, 2 GHz
 - ARM 920T: 0.09W
 - in-order, 1-way, 5 pipeline stages, 16KB \$I, 16KB \$D, 4.7mm², 0.13um, 250 MHz



Low Power & Memory Systems

- Caches reduce power
 - Avoid expensive off-chip (or remote) accesses
 - Anything that improves cache performance typically improves power consumption
 - But introduce an issue of leakage management
- Low power caches
 - Sequential tag – data accesses
 - Multi-bank caches
 - Configurable caches
 - Decay or drowsy caches (turn off portions of the cache)
 - Data compression