EECS 571 “Principles of Real-Time Embedded Systems”
Lecture Note #12:

Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems

ACM SOSP’01

Energy Is a Critical Resource!

- Mobile, handheld devices
- Satellite, space and military systems with limited power budgets
- Physical and thermal limitations on systems
- Task execution uses significant energy, often subject to stringent timing constraints

How to Reduce Power Consumption?

- Hardware:
  - Limit parallelism and speculative execution
  - Improve circuit technology
- Software:
  - Perform fewer computations
  - Improve algorithms and mechanisms

Issue of Operating Voltage

- CMOS is today's predominant device technology
- $E \propto V^2$
- Maximum gate delays inversely related to voltage

$\Rightarrow$ Can reduce energy per unit computation by reducing frequency and voltage

Dynamic Voltage Scaling (DVS)

- [Weiser+94]
  busy system $\Rightarrow$ increase frequency
  idle system $\Rightarrow$ reduce frequency
- Many algorithms for non-RT applications focusing on avg throughput
- Software adjustable PLL, voltage regulator
  - often available, but intended for other things
  - XScale, SpeedStep, PowerNow!, Crusoe

Why not avg throughput-based DVS for rt apps?

- An embedded camcorder contains a task that (i) requires 3 ms of comp time when CPU runs at max fequency, and (ii) must react to a change in sensor reading in 5 ms.

- What happens if the CPU runs at 50% of its max frequency?

Real-Time Systems

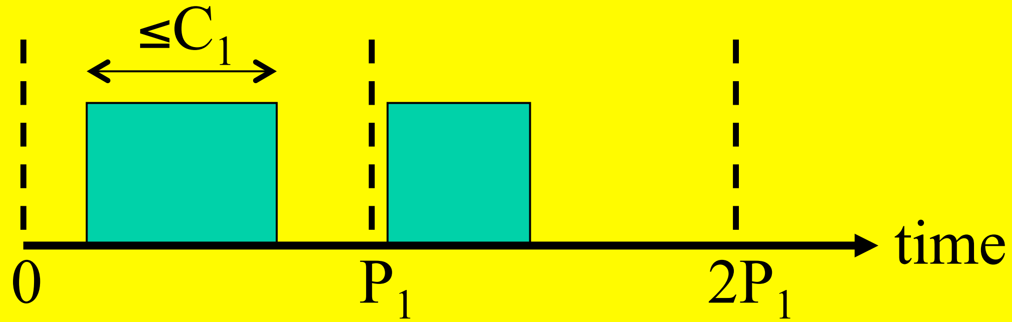
- Require strict timeliness guarantees
- Widespread use in embedded systems
- Well-studied scheduling theories
- Problem: reducing frequency may violate timeliness guarantees

Our Approach

- Development of real-time DVS (RT-DVS)
  - DVS algorithms that maintain RT guarantees
  - Simple enough for online scheduling
  - Work closely with existing RT sched algs.
- In-depth simulation
- Implementation in a real, working system

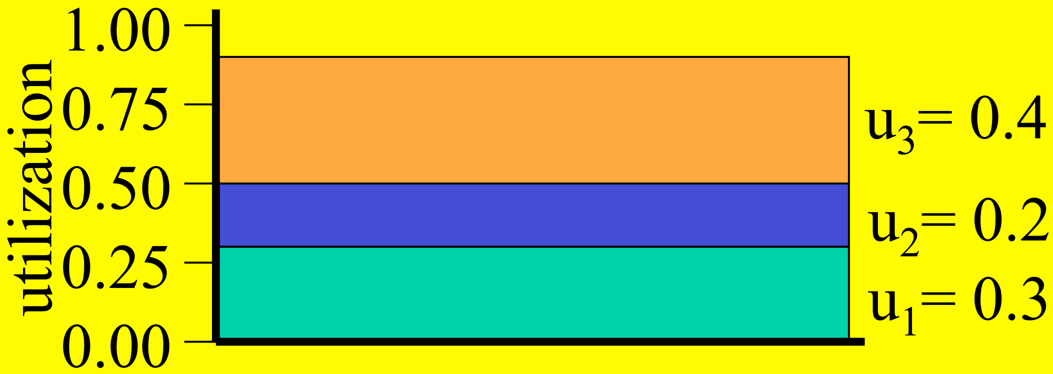
Real-Time Task Model

- Tasks invoked periodically: T_i has period P_i
- C_i is T_i's worst-case execution time (WCET)
- Relative deadline = task period
  - must complete by next invocation



Static Voltage Scaling

- Earliest-Deadline-First (EDF)
- Worst-case utilization: $u_i = C_i / P_i$
- Frequency selection: $\sum u_i \leq f / f_{max}$



$f = 0.9 * f_{max}$

Simulation

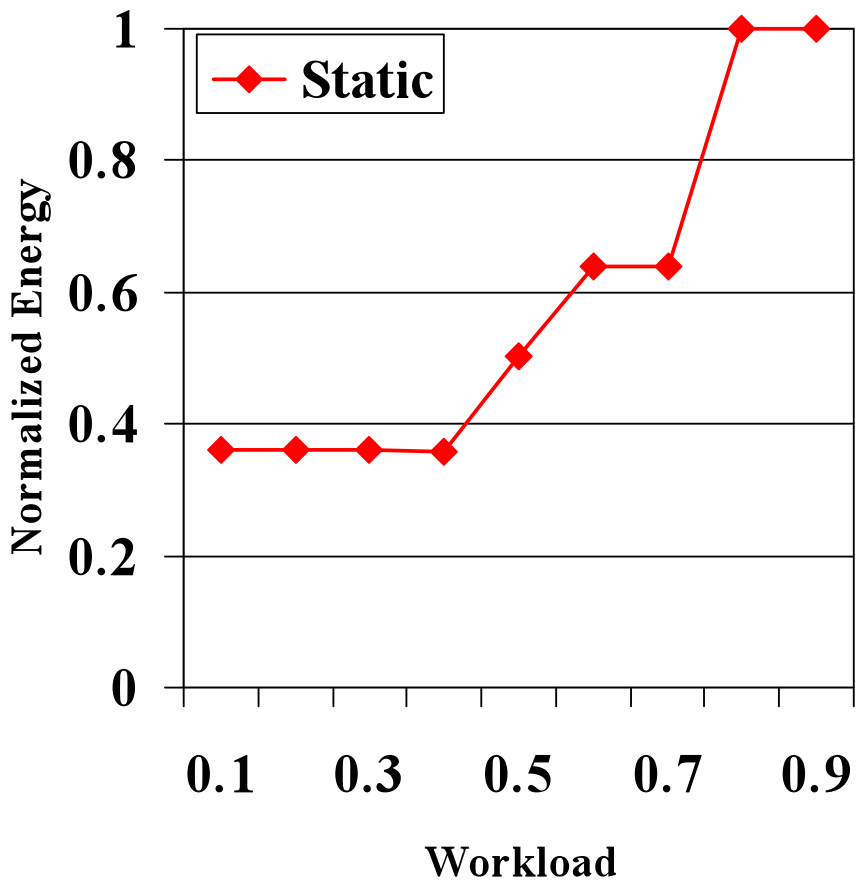
- Parameterized simulation
- Synthetic random real-time task sets
  - uniform distribution of short (1-10), medium (10-100), long (100-1000ms) periods
- Vary execution time distributions
- Vary hardware specifications
- Compare different scheduling algorithms and *theoretical bounds*.

Simulation Setup

- Input: task set, system parameters, sched alg
- Output: energy consumption of each alg
- System parameters:
  - list of freqs and voltages
  - actual fraction of WCET for each task
  - idle level (idle vs. normal op energy consumed)
- Theoretical lower bounds:
  - task execution thruput w/o timing constraints

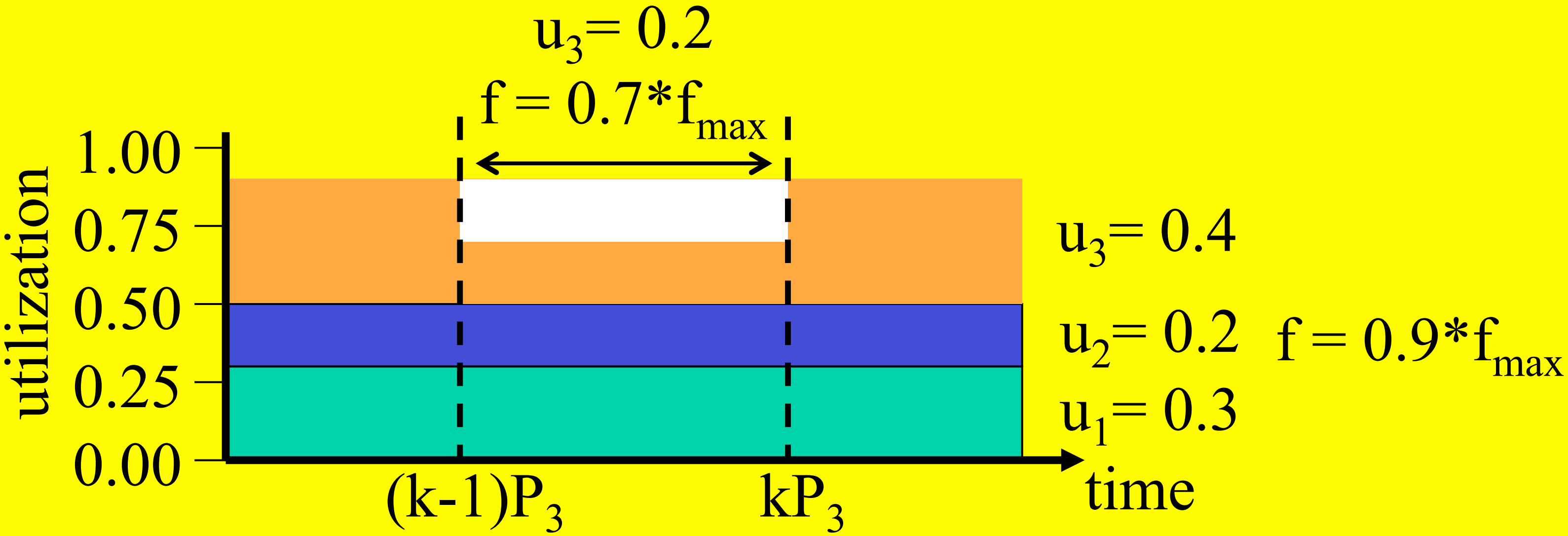
Simulation Results

- 8 tasks in a task set
- 100 random task sets
- Workload = total worst-case utilization
- 3 freq./volt. settings:
  - 5V, $1.0*f_{max}$
  - 4V, $0.75*f_{max}$
  - 3V, $0.5*f_{max}$



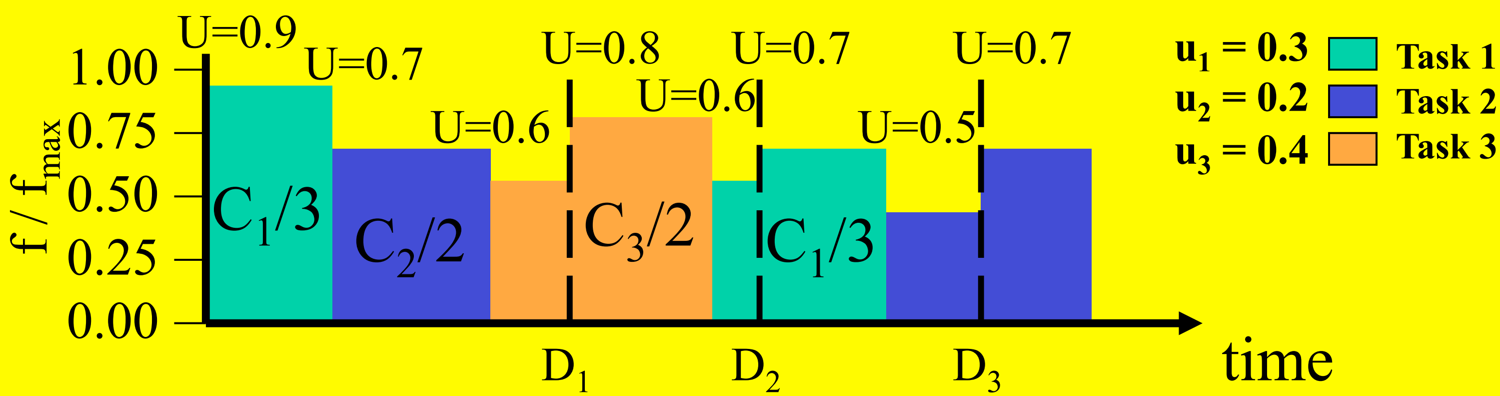
Dynamic Scaling

- If each job uses less than WCET, we can use lower frequency during its invocation interval



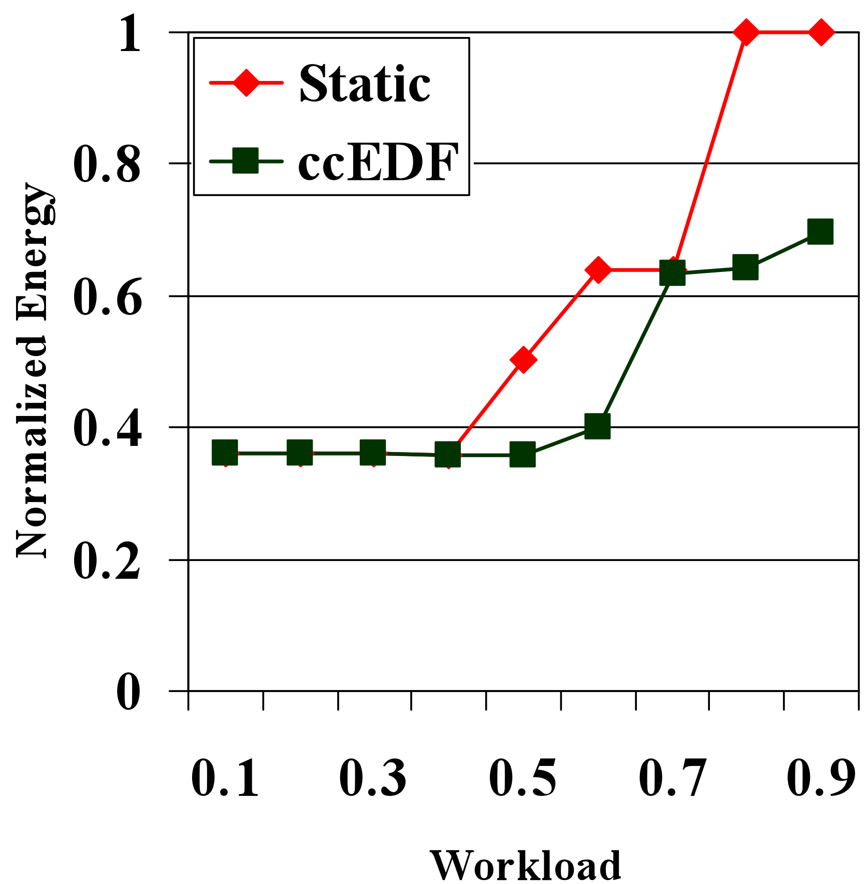
Cycle-Conserving EDF

- $f_{desired} = f_{max} * \sum u_i$
- at T_i release set $u_i = C_i / P_i$
- at T_i finish set u_i = actual execution time / P_i



Simulation Results

- 8 tasks in a set
- 70% WCET each invocation
- 3 freq./volt. settings:
  - 5V, 1.0*f_{max}
  - 4V, 0.75*f_{max}
  - 3V, 0.5*f_{max}

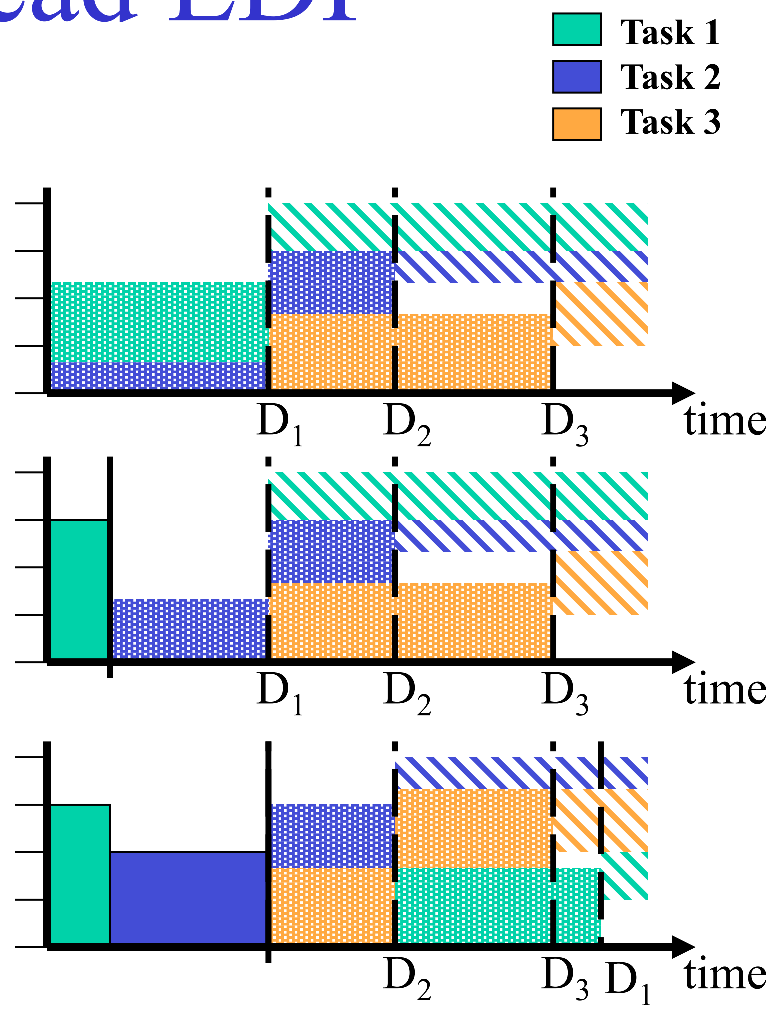


Proactive Techniques

- Tasks typically use much less than WCETs
- Proactively reduce frequencies
- Look ahead to meet future deadlines
- Consider *all* tasks together

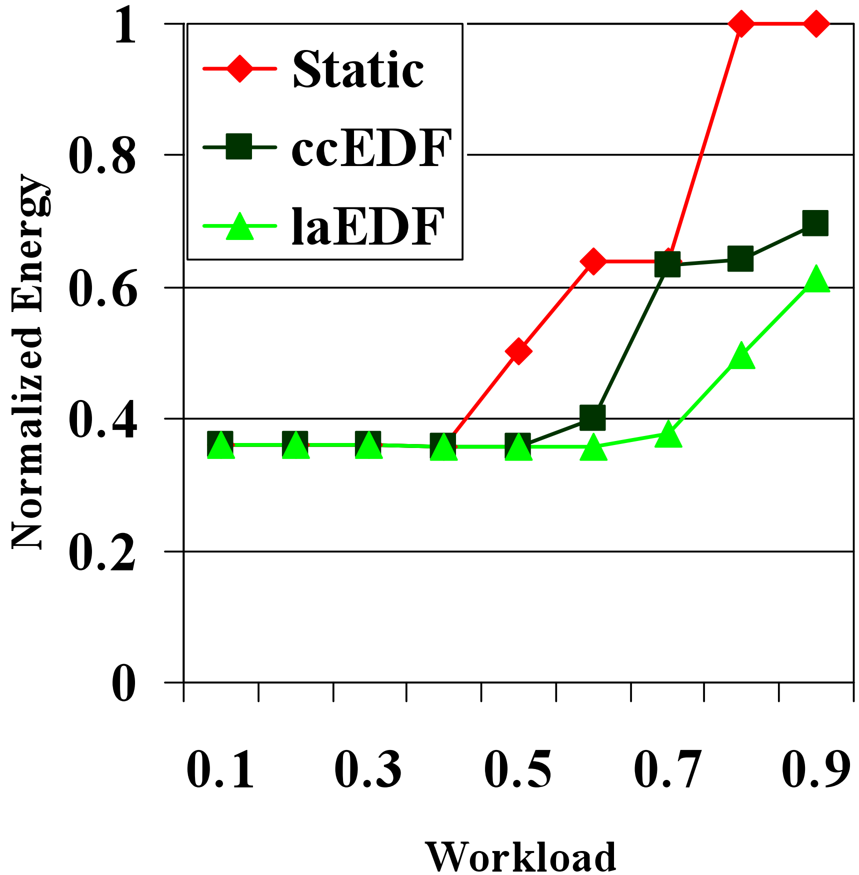
Look-Ahead EDF

- Minimize current frequency
- Trade current savings for potential future loss
- Plan to defer work beyond *next immediate* deadline
- Ensure future deadlines with "reservation"



Simulation Results

- 8 tasks in a set
- 70% WCET each invocation
- 3 freq./volt. settings:
  - 5V, $1.0*f_{max}$
  - 4V, $0.75*f_{max}$
  - 3V, $0.5*f_{max}$



More Simulation Results

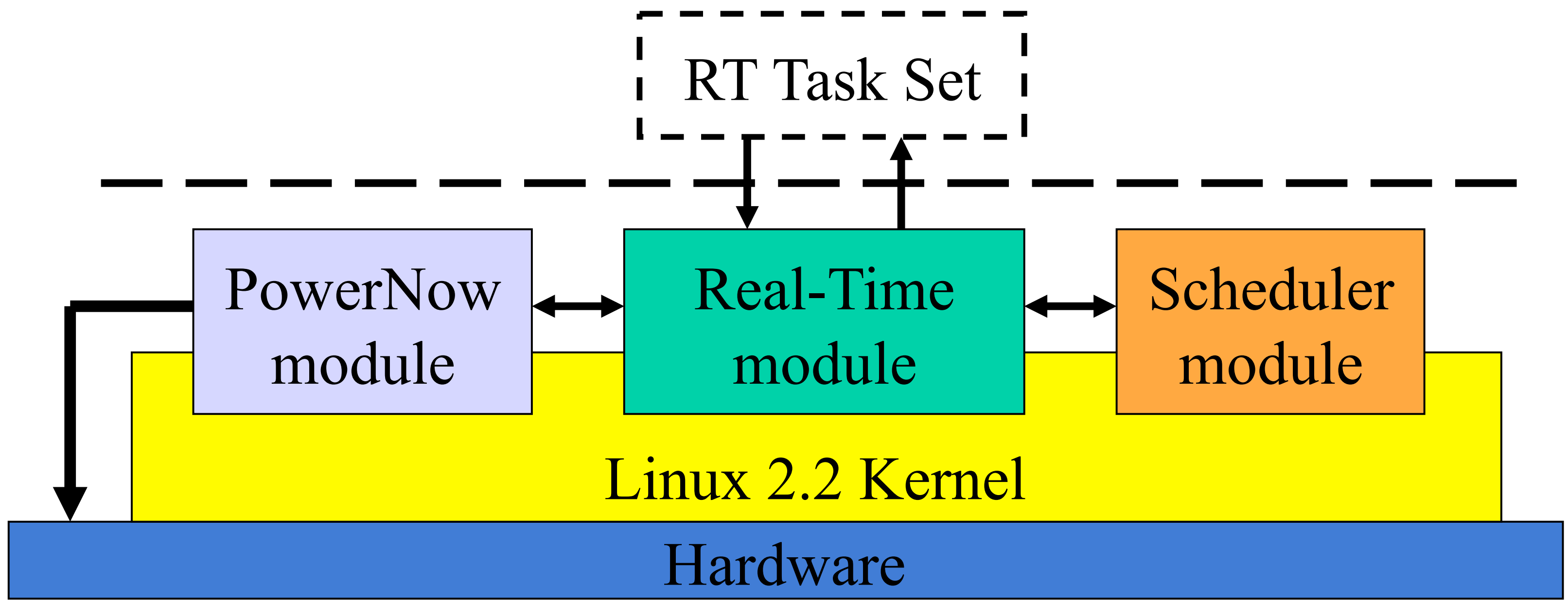
- Number of tasks is not important
- Voltage and frequency settings greatly affect performance
- Look-ahead does not always perform best
- Algorithms can perform close to theoretical lower bound

Implementation

- PC notebook computer
- AMD K6-2+ processor, 550 MHz
- PowerNow! Technology:
  – frequency can be changed 200-550MHz in 50MHz increments
  – voltage selection 1.4V or 2.0V
  – empirical mapping between voltage and frequency
  – switching overheads: 0.4ms (voltage), 41 micros (freq)
- Processing 20W, screen backlight 7W

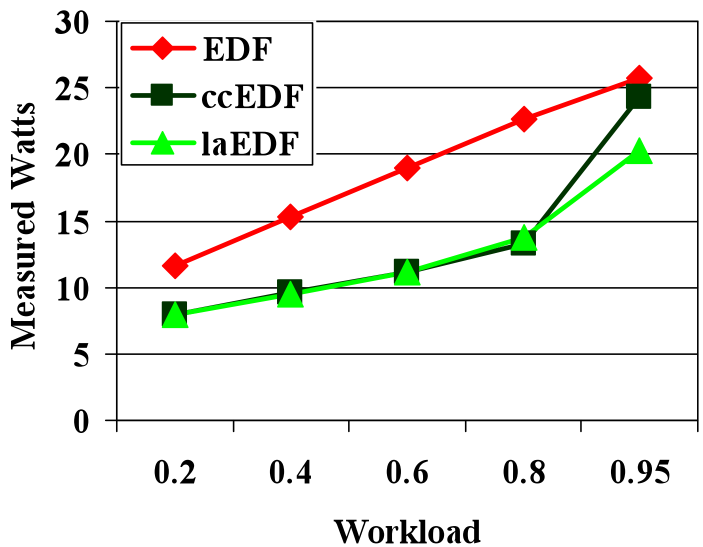
Software Architecture

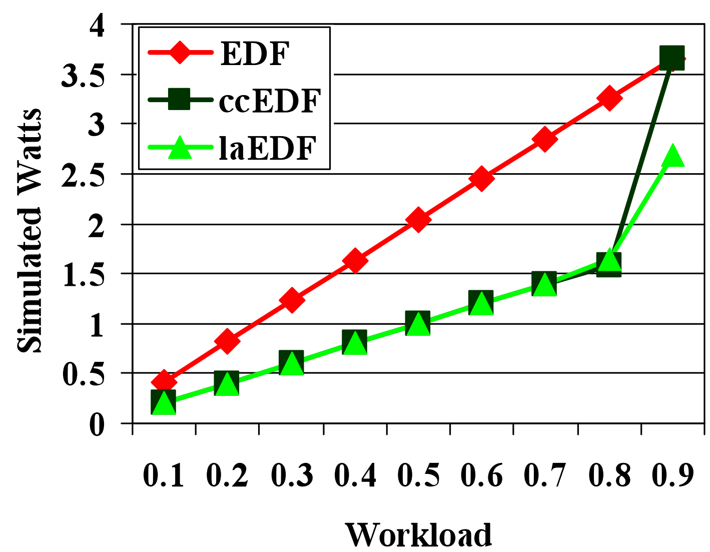
- Real-time extension to Linux 2.2
- Modular design, plug-in schedulers



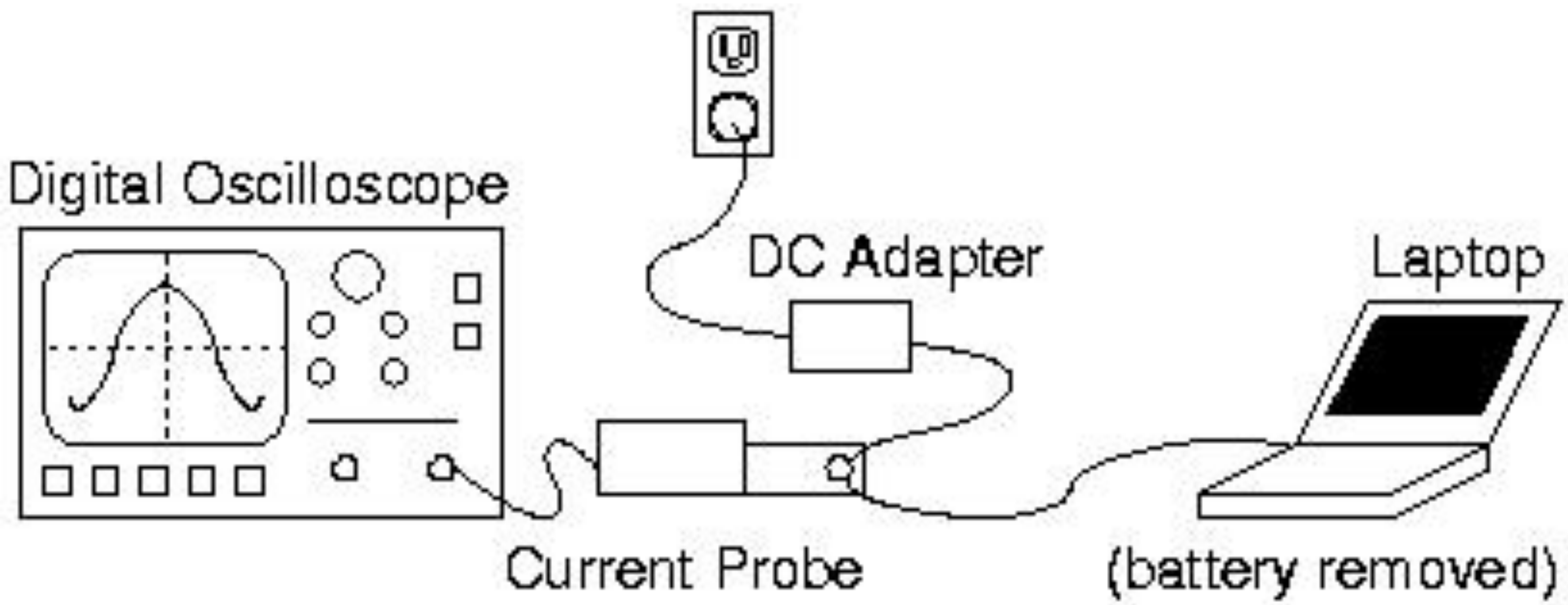
Measurements

- Use oscilloscope with current probe
- Synthetic RT workload w/ backlighting off
- Similar to simulation results (20-40% savings)





Power Measurement



Two Interesting Observations

- The very first invocation of a task may overrun its WCET due to ‘cold’ processor OS states
- Dynamic addition of a task may cause transient deadline misses, especially with more aggressive schemes
  - Insert the task immediately, but release it after completion of current invocations of all existing tasks

Related work

Most are loosely-coupled with OS and based on avg processor utilization

- Many non-RT DVS papers (esp., UCB)
- Offline WCET analysis + online (reclamation) heuristic---no dynamic task set
  - [Krishna+00], [Swaminathan+00]
- Computation-time probability heuristics
  - [Gruian01]
- Compiler-based, application-level DVS
  - [Mosse+00]

Conclusions

- Designed and evaluated 5 DVS algorithms for real-time systems
  - Provide deadline guarantees while scaling freq. and voltage
  - Simple enough to use as online schedulers
- Excellent energy savings, comparable to non-RT DVS
- Implemented on top of Linux
- Onging and future directions:
  - Online RTDVS (RTSS'04)
  - Power-aware virtual memory management (USENIX '03)
  - Hybrid HW-SW memory power management (PACS'04)
  - Larger energy framework, especially energy-aware QoS (EQoS):
  - Power-aware disk I/O (SOSP'05)

Details available in the ACM SOSP01

Code available at: http://kabru.eecs.umich.edu/rtos/