Voltage Scheduling

Pering, Burd, & Brodersen
ISLPED 2000

- Simulation study – lpARM
- 1.1V to 3.3V – 10 MHz to 100 MHz
- Core and cache together consume between 1.8 mW and 220 mW
- Voltage scheduler separate from “temporal” scheduler
- Use “deadlines”

lpARM System

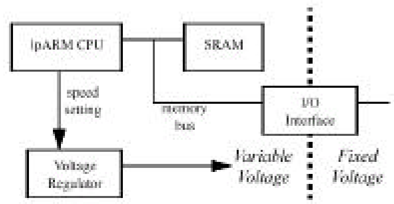


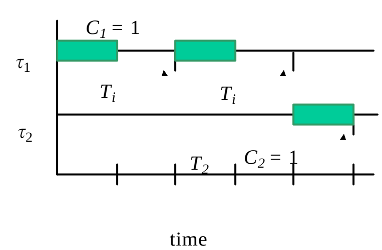
Figure 2: lpARM System Block Diagram

- Speed-control register
- Processor cycle ctrs
- System sleep control

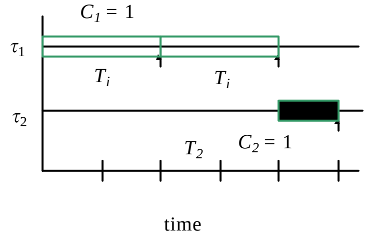
Based on Earliest Deadline First

- Dynamic algorithm
- Priorities are assigned to tasks according to the deadlines of their current request
- With EDF there is no idle time prior to an overflow
- For a given set of *m* tasks, EDF is feasible iff
  $C_1/T_1 + C_2/T_2 + \dots + C_m/T_m \leq 1$
- If a set of tasks can be scheduled by any algorithm, it can be scheduled by EDF

Intuition



Intuition



Workload

- High priority tasks – ignored by voltage scheduler
- Rate-based tasks – turn into deadlines internally
- True deadline-based tasks
  - Allowing missed deadlines enables voltage scheduler to accommodate ave-case workload instead of worst-case.

Scheduling Algorithm

$$speed = \underset{i<=n}{\mathrm{MAX}} \left(\frac{\sum_{j<=i} work_j}{deadline_i - currenttime} \right)$$

Exponential moving average

- Sort in EDF order
- Invoked when thread added or removed or deadline reached
- Includes non-runnable in scheduling decision

	StartTime	Deadline	Workload	Σ Workload	Σ /Deadline
Task A	0	3	144	144	48
Task B	2	5	74	218	44.4
Task C	7	9	88	306	34

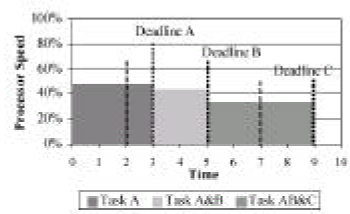


Figure 3: Example Voltage Schedule

Results

Lowering voltage until benchmark doesn't complete

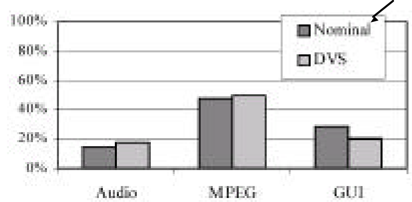
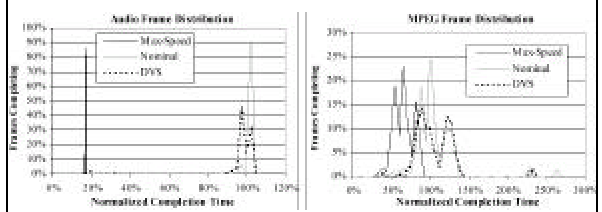


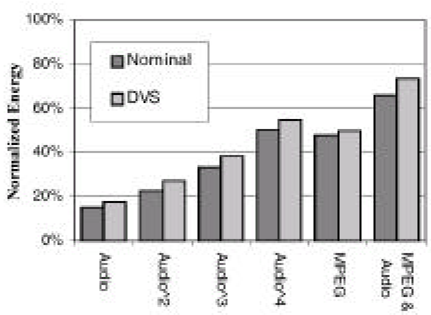
Figure 4: Basic Voltage Scheduling Results

Frame by frame histograms



GUI example: affected by hard idle times – waiting for external events not related to clock speed.

Mixed Workload



Conclusions

- In all cases, less than 2% of energy going to scheduling thread execution
- Up to 80% reduction in energy
- Application information required

Multiprocessor Affinity Scheduling

- Question: Where (on which node) to run a particular thread during the next time slice?
- Processor's POV: favor processes which have some residual state locally (e.g. cache)
- What is a useful measure of affinity for deciding this?
  - Least intervening time or intervening activity (number of processes here since "my" last time) *
  - Same place as last time "I" ran.
  - Possible negative effect on load-balance.

Affinity Analogies?