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

Author: Padmanabhan Pillai and Kang G. Shin

Domgmin Song
Real-Time Computing and Communications Lab.
Hanyang University

2006. 9. 25

Contents

Background and Motivation

- Background
  - Dynamic Voltage Scaling is an energy-saving technique that consists of varying the frequency and voltage of a microprocessor in real-time according to processing needs
- Motivation
  - For a large class of applications in embedded real-time systems , the variable operating frequency interferes with their deadline guarantee mechanisms
  - To provide real-time guarantees, DVS must consider deadlines and periodicity of real-time tasks, requiring integration with the real-time scheduler

Goals and Contributions

- Goals
  - As mobile systems, they should be designed to maximize battery life, but they need powerful processors
  - These processors Consume more energy than those in simpler devices, thus reducing battery life using RT-DVS
- Contributions
  - To provide energy-saving DVS capability in a system requiring real-time deadline guarantees, we have developed a class of RT-DVS algorithms

RT-DVS

- Why DVS?
  - Trade off (computational power ⟷ battery life)
    - Power requirements are one of the most critical constraints in mobile computing applications
    - Low speed/low power processor → long battery life
      → poor performance
    - The faster processor → the higher the energy cost
    - Fixed embedded device and battery size → fixed available energy
    - Power consumption affects the battery life of device
  - DVS can provide the performance to meet peak computational demands
  - reduce power consumption
  - Available on low-perfomance processors

Screen	CPU subsystem	Disk	Power
On	Idle	Spinning	13.5 W
On	Idle	Standby	13.0 W
Off	Idle	Standby	7.1 W
Off	Max. Load	Standby	27.3 W

Table 1: Power consumption measured on Hewlett-Packard N3350 laptop computer

RT-DVS

- Real-time issues
  - Most DVS algorithms are based on solely average computational throughout
    - Do not consider real-time constraints
    - Cannot provide timeliness guarantees
    - Simple feedback mechanism
  - DVS must ensure that both of these conditions hold
    - the task set is schedulable
    - no task exceeds its worst-case computation bound

RT-DVS

- Static RT-DVS for RM and EDF

EDF_test (α):
if ($C_1/P_1 + \cdots + C_n/P_n \leq \alpha$) return true;
else return false;

RM_test (α):
if ($\forall T_i \in \{T_1, \ldots, T_n | P_1 \leq \cdots \leq P_n\}$
$\lceil P_i/P_1 \rceil * C_1 + \cdots + \lceil P_i/P_i \rceil * C_i \leq \alpha * P_i$)
return true;
else return false;

select_frequency:
use lowest frequency $f_i \in \{f_1, \ldots, f_m | f_1 < \cdots < f_m\}$
such that RM_test(f_i/f_m) or EDF_test(f_i/f_m) is true.

Figure 1: Static voltage scaling algorithm for EDF and RM schedulers

RT-DVS

- Static RT-DVS example

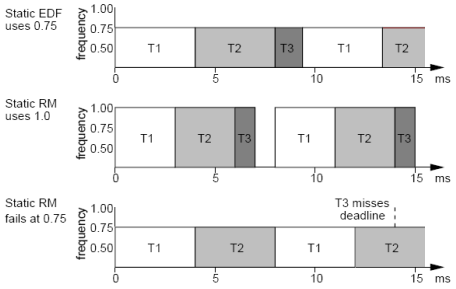

Figure 2: Static voltage scaling example

Task	Computing Time	Period
1	3 ms	8 ms
2	3 ms	10 ms
3	1 ms	14 ms

Table 2: Example task set, where computing times are specified at the maximum processor frequency

RT-DVS

- Cycle-conserving RT-DVS
  - When the task release
    - How much computation it will actually require
    - The conservative assumption that it will need its specified worst-cast processor time
  - When the task completes
    - Compare the worst-case with the actual processor cycles
  - Instead of idling for extra processor cycles, we can devise DVS algorithms that avoid wasting cycles by reducing the operating frequency

RT-DVS

- Cycle-conserving RT-DVS example

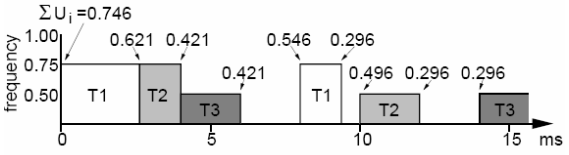

Figure 3: Example of cycle-conserving EDF

Task	Invocation 1	Invocation 2
1	2 ms	1 ms
2	1 ms	1 ms
3	1 ms	1 ms

Table 3: Actual computation requirements of the example task set (assuming execution at max. frequency)

RT-DVS

- Cycle-conserving DVS for EDF scheduler

select_frequency():
    use lowest freq. $f_i \in \{f_1, \ldots, f_m | f_1 < \cdots < f_m\}$
    such that $U_1 + \cdots + U_n \leq f_i/f_m$

upon task_release(T_i):
    set U_i to C_i/P_i;
    select_frequency();

upon task_completion(T_i):
    set U_i to cc_i/P_i;
        /* cc_i is the actual cycles used this invocation */
    select_frequency();

Figure 4: Cycle-conserving DVS for EDF schedulers

RT-DVS

- Cycle-conserving RM example

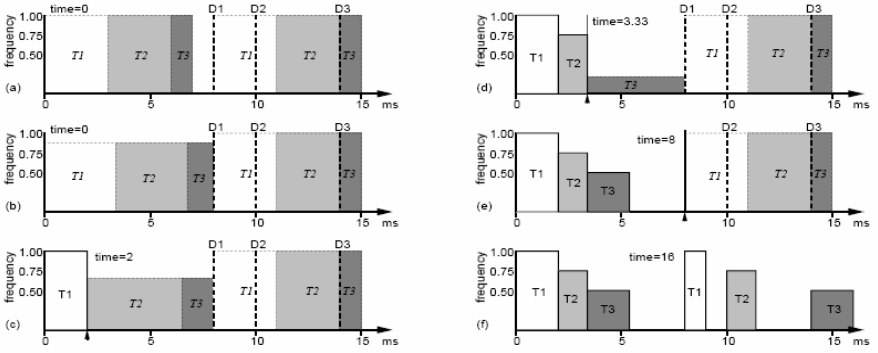

Figure 5: Example of cycle-conserving RM: (a) Initially use statically-scaled, worst-case RM schedule as target; (b) Determine minimum frequency so as to complete the same work by D1; rounding up to the closest discrete setting requires frequency 1.0; (c) After T1 completes (early), recompute the required frequency as 0.75; (d) Once T2 completes, a very low frequency (0.5) suffices to complete the remaining work by D1; (e) T1 is re-released, and now, try to match the work that should be done by D2; (f) Execution trace through time 16 ms.

RT-DVS

- Cycle-conserving RT-DVS for RM

assume f_j is frequency set by static scaling algorithm

select_frequency():
    set s_m = max_cycles_until_next_deadline();
    use lowest freq. $f_i \in \{f_1, \ldots, f_m | f_1 < \cdots < f_m\}$
    such that $(d_1 + \cdots + d_n)/s_m \leq f_i/f_m$

upon task_release(T_i):
    set $c\_left_i = C_i$;
    set s_m = max_cycles_until_next_deadline();
    set $s_j = s_m * f_j/f_m$;
    allocate_cycles (s_j);
    select_frequency();

upon task_completion(T_i):
    set $c\_left_i = 0$;
    set $d_i = 0$;
    select_frequency();

during task_execution(T_i):
    decrement c_left_i and d_i;

allocate_cycles(k):
    for i = 1 to n, $T_i \in \{T_1, \ldots, T_n | P_1 \leq \cdots \leq P_n\}$
        /* tasks sorted by period */
        if ($c\_left_i < k$)
            set $d_i = c\_left_i$;
            set $k = k$ - c_left_i;
        else
            set $d_i = k$;
            set $k = 0$;

RT-DVS

- Look-Ahead RT-DVS
  - The look-ahead technique to determine future computation need and defer task execution
    - The look-ahead scheme tries to defer as much work as possible
    - sets the operating frequency to meet the minimum work that must be done now to ensure all future deadlines are met
    - Run at high frequencies later in order to complete all of the deferred work in time

RT-DVS

- Look-Ahead RT-DVS example

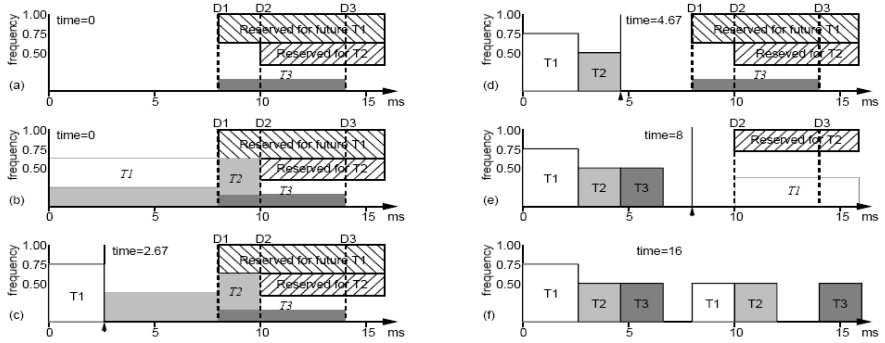

Figure 7: Example of look-ahead EDF: (a) At time 0, plan to defer T3's execution until after D1 (but by its deadline D3, and likewise, try to fit T2 between D1 and D2; (b) T1 and the portion of T2 that did not fit must execute before D1, requiring use of frequency 0.75; (c) After T1 completes, repeat calculations to find the new frequency setting, 0.5; (d) Repeating the calculation after T2 completes indicates that we do not need to execute anything by D1, but EDF is work-conserving, so T3 executes at the minimum frequency; (e) This occurs again when T1's next invocation is released; (f) Execution trace through time 16 ms.

RT-DVS

- Look-Ahead DVS for EDF scheduler

```
select_frequency(x):
        use lowest freq. f_i ∈ {f_1, ..., f_m | f_1 < ··· < f_m}
        such that x ≤ f_i/f_m

upon task_release(T_i):
        set c_left_i = C_i;
        defer();

upon task_completion(T_i):
        set c_left_i = 0;
        defer();

during task_execution(T_i):
        decrement c_left_i;

defer():
        set U = C_1/P_1 + ··· + C_n/P_n;
        set s = 0;
        for i = 1 to n,    T_i ∈ {T_1, ..., T_n | D_1 ≥ ··· ≥ D_n}
                           /* Note: reverse EDF order of tasks */
                set U = U − C_i/P_i;
                set x = max(0, c_left_i − (1 − U)(D_i − D_n));
                set U = U + (c_left_i − x)/(D_i − D_n);
                set s = s + x;
        select_frequency (s/(D_n − current_time));
```

RT-DVS

- Summary of RT-DVS algorithms
  - Fairly easy to incorporate into a real-time operating system
  - do not require significant processing
  - no more than two switches per a task
  - The dynamic schemes all require O(n) computation
  - The most significant overheads may come from the H/W voltage switching times

RT-DVS method	energy used
none (plain EDF)	1.0
statically-scaled RM	1.0
statically-scaled EDF	0.64
cycle-conserving EDF	0.52
cycle-conserving RM	0.71
look-ahead EDF	0.44

Table 4: Normalized energy consumption for the example traces

Simulations

- Simulation Methodology
  - Simulator takes as input a task set
    - task's period, computation requirements and other system parameters
  - Only the energy consumed by the processor is computed
    - Task execution modeling -> counting cycles of execution
  - Task sets are generated randomly
    - Equal probability of having a short(1-10ms), medium(10-100ms) or long(100-1000ms) period
  - Task periods are uniformly distributed
  - Task computation requirements are scaled by a constant chosen such that the sum of the utilizations of the tasks reaches a desired value

Simulations

- Simulation results
  - Assume DVS-capable platform provides operating frequency / corresponding voltage
    - 0.5/3, 0.75/4, 1.0/5
  - Also include theoretical lower bound for energy dissipation
  - This lower bound is computed by taking the total number of task computation cycles, and determining the absolute minimum energy

Simulations

- Varying number of tasks

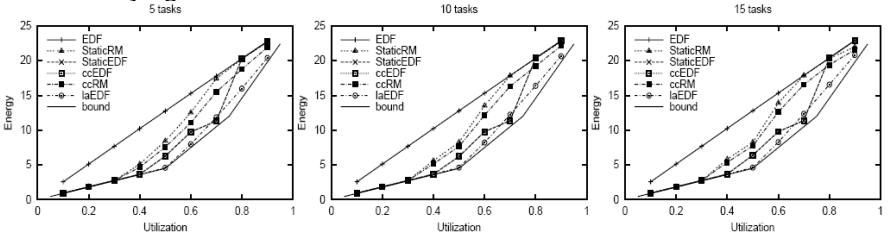

Figure 9: Energy consumption with 5, 10, and 15 tasks

  - Assumption
    - Perfect S/W controlled halt
    - Tasks consume their worst-case
  - staticEDF = ccEDF
  - Significant in mid-range utilization
  - The number of tasks has very little effect

Simulations

- Varying idle level

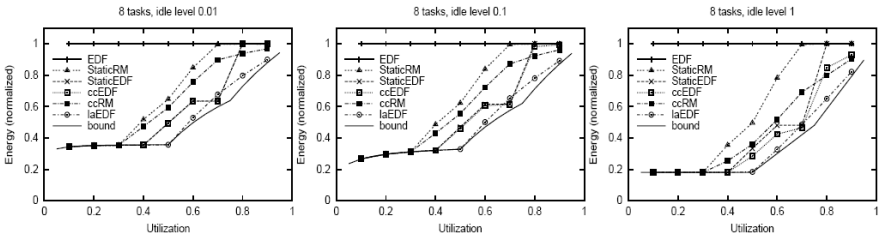

Figure 10: Normalized energy consumption with idle level factors 0.01, 0.1, and 1.0

- Idle level factor : the ratio of energy consumed in a halted / normal cycles
- It shows relative energy consumption by normalized with respect to the unmodified EDF energy consumption
- Energy-aware schedulers are not significantly affected by changing the idle factor

Simulations

- Varying machine specifications

machine 0: { (0.5, 3), (0.75, 4), (1.0, 5) }
machine 1: { (0.5, 3), (0.75, 4), (0.83, 4.5), (1.0, 5) }
machine 2: { (0.36, 1.4), (0.55, 1.5), (0.64, 1.6), (0.73, 1.7), (0.82, 1.8), (0.91, 1.9), (1.0, 2.0) }

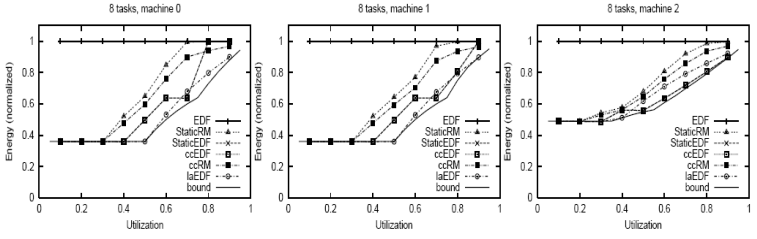

Figure 11: Normalized energy consumption with machine 0, 1, and 2

- normalized with respect to the unmodified EDF energy consumption
- In machine 1, ccEDF is better than ccRM on the crossing point near full utilization
- In machine 2, AMD's PowerNow!™

  ccEDF outperforms laEDF, because ccEDF is more closely matched

Simulations

- Varying computation time

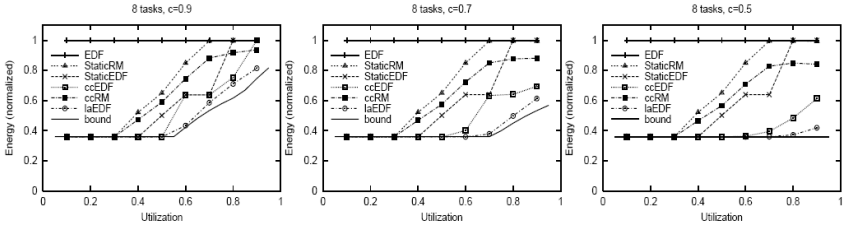

Figure 12: Normalized energy consumption with computation set to fixed fraction of worst-case allocation

- normalized with respect to the unmodified EDF energy consumption
- Statically-scaled mechanisms are not affected, since they are based on worst-case
- ccEDF and laEDF are significant
- Uniform distribution between 0 and worst-case

  Average execution is 0.5 times worst-case

Simulations

- Varying computation time

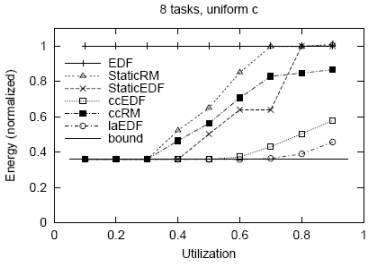

Figure 13: Normalized energy consumption with uniform distribution for computation

Implementation

- H/W Platform
  - Notebook computer
    - Hewlett-Packard N3350
      - AMD K6-2+ (550 MHz)
      - *PowerNow*

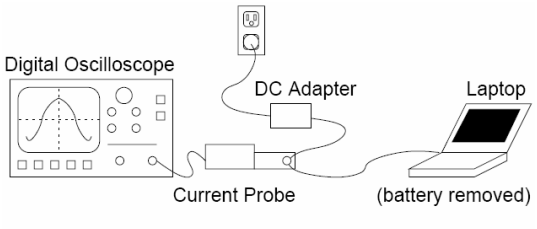

Figure 15: Power measurement on laptop implementation

Implementation

- S/W Architecture
  - Implemented as extension modules Linux 2.2.16 kernel

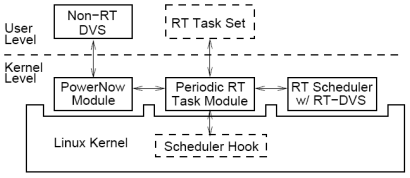

Figure 14: Software architecture for RT-DVS implementation

Measurements and Observations

- Figure 16 shows the actual power consumption
  - Total system power
- Figure 17 shows a simulation with identical parameters
  - Only processor's energy consumption
- There are interesting phenomena
  - cold processor and OS state
  - dynamic addition of a task

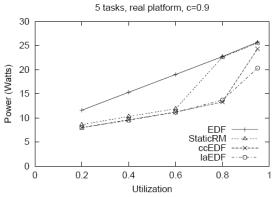

Figure 16: Power consumption on actual platform

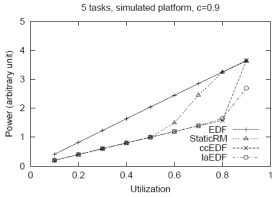

Figure 17: Power consumption on simulated platform

Conclusions

- Presented novel algorithms for RT-DVS
- Can achieve significant energy savings when task management mechanisms and real-time scheduler
- Presented simulation results, showing the most significant parameters
- The number of tasks and energy efficiency of idle cycles do not greatly affect the relative savings of the RT-DVS
- The voltage and frequency settings available on the H/W and the task set CPU utilizations profoundly affect the performance of RT-DVS algorithms
- Look-ahead and cycle-conserving RT-DVS mechanisms can achieve close to the theoretical lower bound on energy
- The measurements indicate that 20% to 40% energy savings can be achieved