Power-Aware Scheduling

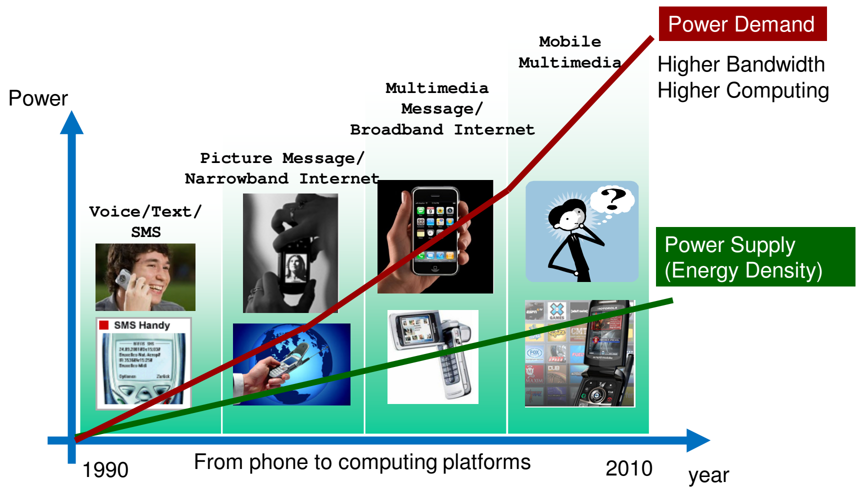
Raj Rajkumar

Lecture #10

Outline

- Need for Power Management
  - DVS and DFS
- Constant Speed Energy Minimization
- ***SysClock***
- ***PM-Clock***
- ***Dynamic PM-Clock***
- Effects of Hardware Limitations
- Evaluation

Power Trend in Mobile Devices



The power supply & demand relationship is according to unpublished research by the Boston Consulting Group

Motivation

- The power consumption of CMOS circuits is given by

$$P \;\alpha\; C_L * V^2_{dd} * f$$

- f $\alpha\,(V_{dd}-V_{th})^{\alpha} / V_{dd}$; For a 0.25-m technology, $\alpha \approx 1.3\text{-}1.5$
- Energy = P * Delay $\approx k * f^{2/(\alpha-1)} \approx k * f^{x}$

➡ Running a CPU at lower speed consumes less energy

- Real-Time Tasks are required to complete at their deadline.

➡ Reducing the CPU speed to complete the task just before its deadline achieves the same performance but saves energy

Issue of Operating Voltage/Frequency

- Predominant device technology is CMOS
- In CMOS, slower (faster) frequency means lower (higher) number of transitions
  - Hence, lower power is consumed
- Power is proportional to V^2f
  - P → Power
  - V → Voltage
  - *f* → Frequency
- Can reduce unit computation energy by reducing frequency and voltage
- Maximum gate delays inversely related to voltage

> • Applies to processors.
> • Can be applied to memory as well.
> • Check specifications carefully.

Scaling Techniques

- **DFS: Dynamic Frequency Scaling**
  - Lower or increase clock frequency
  - With lower frequency, the same amount of work (# of instructions to be executed) takes longer
  - Hence, power may go down, but energy <u>may</u> remain the same
- **DVS: Dynamic Voltage Scaling**
  - Lower or increase clock voltage
  - As frequency increases, higher voltage is required
  - Conversely, as frequency is lowered, only a lower voltage is required
- **DVFS: Dynamic Voltage/Frequency Scaling**
  - Scale voltage up (down) when frequency is scaled up (down)
  - Needs processors supporting software adjustable PLL, voltage regulator
    - e.g., XScale, SpeedStep, PowerNow!, Crusoe

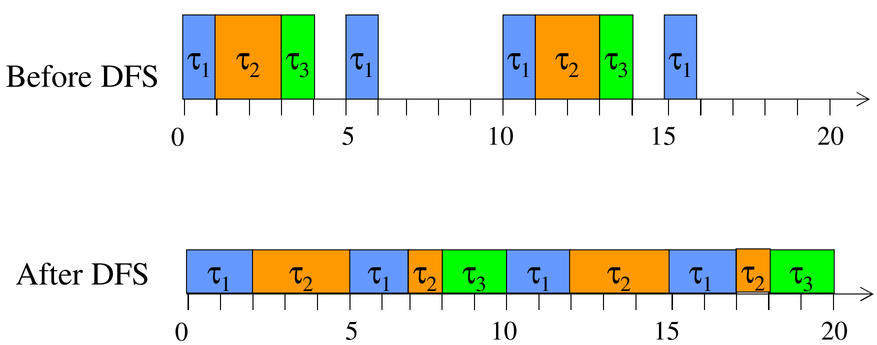
Electrical & Computer ENGINEERING 18-648: Embedded Real-Time Systems Carnegie Mellon

Dynamic Frequency Scaling (DFS)

- [Weiser+94]
  - busy system → increase freqency
  - idle system → reduce frequency

Using DFS

- "Static" DFS
- Taskset: τ_1= (1, 5) $\tau_{2\,=}$ (2, 10) τ_3 = (1, 10)
  U = 1/5 + 2/10 + 1/10 = 0.5

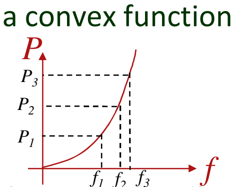


The Problem

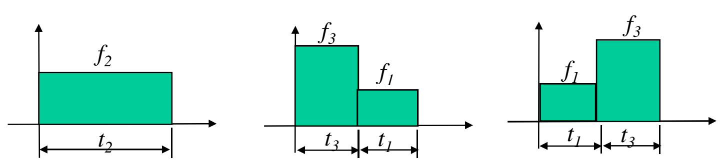
How to determine the CPU clock frequency to satisfy the schedulability of real-time task sets and save energy?

Constant Speed Energy Minimization

- The power-frequency relationship is a convex function
  - $P = k f^x, 0 \leq f \leq f_{max}$



- Energy is minimized if a workload with a given deadline is executed at *constant* speed



- $f_1 \leq f_2 \leq f_3$; $t_2 = t_1 + t_3$; $(f_2 * t_2) = (f_1 * t_1) + (f_3 * t_3)$
- Then $p_2 * t_2 \leq p_1 * t_1 + p_3 * t_3$

Brief Overview

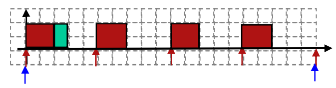
- Example of simple frequency-scaling algorithm for fixed-priority Pre-emptable scheduling policies (RM, DM, FIFO, etc.)
- Static frequency scaling (SFS) algorithms
  - Optimal system-clock-freq assignment: one clock frequency for the task set
  - Task-clock-freq assignment: one clock frequency for each task
- Dynamic voltage scaling (DVS) algorithm
- Experiment results
- Effect of hardware limitation
  - Non-ideal power-frequency characteristic, finite operating frequencies

System Model and Notation

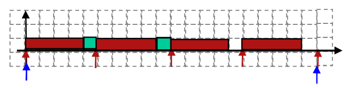
- Task requirement {C, T, D}
  - C: number of cycles
  - T, D: period and deadline in time units
- Tasks are scheduled by deadline-monotonic (DM) scheduling algorithm
- Assume there are *n* tasks, τ_1 to τ_n, and $D_1 \leq D_2 \leq \ldots \leq D_n$
- ν_i denotes CPU clock frequency to execute task τ_i
- **Goal**: minimize energy usage of the task set within the hyper-period
  - Hyperperiod = LCM of all the periods

Static Frequency Scaling Algorithm Example

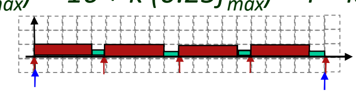
- $\tau_1\{2, 5, 4\}$ and τ_2 $\{1, 20, 20\}$; Hyper-period (H) = 20
- Assume f_{max} = 1 cycle/time unit; Energy (E) = $k*f^3*t$
- If the CPU runs at f_{max}, $E = k\, f_{max}^{3} * 9$



- System clock frequency assignment sets CPU speed to $0.5*f_{max}$, $E = k\,(0.5f_{max})^3 * 18 = k\, f_{max}^{3} * 2.25$



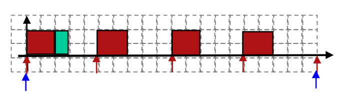
- Task clock frequency assignment sets $\nu_1 = 0.5f_{max}$, $\nu_2 = 0.25\, f_{max}$, $E = k\,(0.5f_{max})^3 * 16 + k\,(0.25f_{max})^3 * 4 = k\, f_{max}^{3} * 2.0625$



Basic Intuition Behind Sys-Clock

- Consider the workload needed to complete a given task
  - The execution of higher priority tasks and itself
  - The energy to satisfy τ_i's deadline is obtained by running its higher priority tasks and itself with the lowest possible speed

$\tau_1\{2, 5, 4\}$ and τ_2 $\{1, 20, 20\}$



  - Workload for τ_2 (W_2) = $4*C_1 + C_2$ *cycles*
  - Lowest possible freq. $\Omega_2 = w_2/d_2 = (4*2+1)/20 = 0.45$ *cycles/time unit*
- To maintain schedulability, **<u>all</u>** tasks' constraints must be satisfied
  - With speed *0.45 cycles/time unit,* τ_1 misses its deadline!!
  - Lowest possible freq. $\Omega_1 = w_1/d_1 = 2/4 = 0.50$ *cycles/time unit*
- The CPU needs to run at, at least, *max(0.45,0.50) = 0.50 cycles/time unit* to satisfy the task set

Sys-Clock Algorithm [1]

- Consider a task's workload at critical instant
  - The task's execution and its preemption
- Workload is varied only when the task completes
  - Later the completion time, higher the preemption
  - Hence, **completing a task at its deadline is not always optimal** !!
- Apply constant speed energy minimization concept
  - Consider all possible completion times before deadline
  - Energy-minimizing freq of a task (υ)= the lowest speed

Sys-Clock Algorithm [2]

- For schedulability:

  All task deadlines must be satisfied.

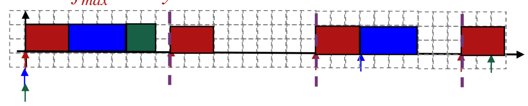
  Sys-Clock-Freq = MAX [υ] for all tasks

- Lower frequency $\rightarrow$ At least one task will miss deadline
- **Sys-Clock is optimal among single-clock-frequency schemes.**

Workload Vs. Frequency

$\tau_1\{3, 10, 10\}$,
$\tau_2\{4, 23, 23\}$,
$\tau_3\{2, 32, 32\}$,

Run at f_{max} = *1 cycle/time unit*

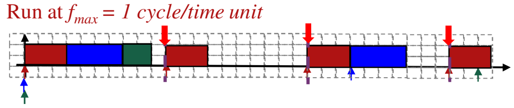


- The workload needed to satisfy each task's constraint varies with the CPU clock frequency
- $\Omega_{3|t=10} = (C_1+C_2+C_3)/10 = (3+4+2)/10 = 0.9$
- $\Omega_{3|t=20} = (2C_1 + C_2+C_3)/20 = (6+4+2)/20 = 0.6$
- $\Omega_{3|t=23} = (3C_1 + C_2+C_3)/23 = (9+4+2)/23 = 0.652$
- $\Omega_{3|t=30} = (3C_1+2C_2+C_3)/30 = (9+8+2)/30 = 0.63$
- $\Omega_{3|t=32} = (4C_1+2C_2+C_3)/32 = (12+8+2)/32 = 0.6875$
- $\Omega_3 = min(\Omega_{3|t})$ for all *t's* $= min(0.9, 0.6, 0.65, 0.63, 0.69) = 0.6$
- *t* is the ending time of an idle period before the deadline

Sys-Clock Example

$\tau_1\{3, 10, 10\}$
$\tau_2\{4, 23, 23\}$
$\tau_3\{2, 32, 32\}$

Run at f_{max} = *1 cycle/time unit*



- *t's* are the endings of idle periods when the workload changes
- Compute freq. needed to complete a task at time *t*
- For task τ_1, υ_1 is given by = $(C_1/T_1) = 3 / 10 = 0.3$
- For task τ_2, υ_2 is computed as
  - $freq(\tau_2)|t=10 = (C_1+C_2)/10 = (3+4)/10 = 0.7$
  - $freq(\tau_2)|t=20 = (2C_1 + C_2)/20 = (6+4)/20 = 0.5$
  - $freq(\tau_2)|t=23 = (3C_1+C_2)/30 = (9+4)/23 = \mathbf{0.565}$

$\upsilon_2 = min$[freq(τ_2) for all *t*]
= *min*[0.7,0.5,0.565]
= 0.5

- For task τ_3, υ_2 is computed as
  - $freq(\tau_3)|t=10 = (C_1+C_2+C_3)/10 = (3+4+2)/10 = 0.9$
  - $freq(\tau_3)|t=20 = (2C_1 + C_2+C_3)/20 = (6+4+2)/20 = 0.6$
  - $freq(\tau_3)|t=30 = (3C_1+2C_2+C_3)/30 = (9+8+2)/30 = 0.63$

$\upsilon_3 = min$[freq(τ_3) for all *t*]
= *min*[0.9,0.6,0.63]
= 0.6

- Sys-clock freq = $max[\upsilon_1, \upsilon_2, \upsilon_3] = max[0.3, 0.5, 0.6]$

Sys-Clock Algorithm

```
During admission control:
    for i = 1 to n:
        α_i = Compute_alpha(τ_i)
        if (α_i > 1) return fail
    end for
    sys_clock = MAX_i(α_i)
    set_system_clock(sys_clock)
```

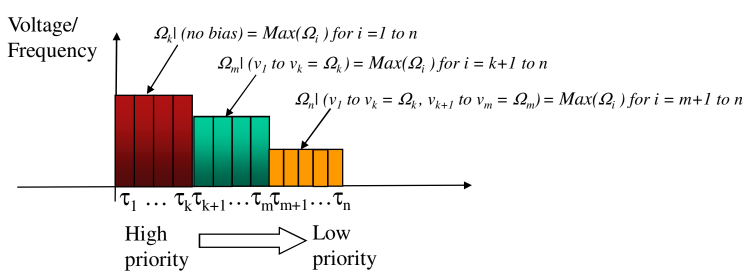
Sys-Clock is optimal among all (global) system clock frequency assignment algorithms

- If the CPU clock frequency is lower than Sys-Clock, at least one task will miss its deadline
- If the CPU clock frequency is higher than *Sys-Clock*, more energy will be consumed

```
Compute_alpha(τ_i):
    IN_BZP = true /* computing busy period or idle period */
    Γ = C_i /* time trace */
    slack = 0 /* the slack time */
    w = 0 /* the workload */
    t^(w) = 0 /* the beginning of the next busy period */
    α = 2.0 /* the energy-minimizing clock frequency normalized to f_max */
    while (Γ ≤ D_i)
        if(IN_BZP)
            Δ = D_i − Γ
            while ((Γ < D_i) and (Δ > 0))
                /* hp(τ_i) is the set of tasks with priorities ≥ τ_i's */
                Γ' = slack + Σ_{τ_j ∈ hp(τ_i)} (⌊Γ/T_j⌋ + 1) * (C_j/f_max)
                Δ = Γ' − Γ
                Γ = Γ'
            end while
            IN_BZP = false
        else
            idle_duration = MIN_{τ_j ∈ hp(τ_i)}[D_i − Γ, (⌈Γ/T_j⌉ * T_j) − Γ]
            Γ+ = idle_duration
            slack+ = idle_duration
            t^(w) = Γ
            w = t^(w) − slack
            α = MIN(α, w/t^(w))
            IN_BZP = true
        end if
    end while
    return α
```


PM-Clock Algorithm

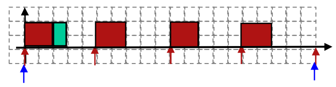
- Each task has its own clock frequency setting
- If a higher priority task needs higher frequency
  - a lower priority task's frequency can be reduced even more
    - Since the higher priority task will complete faster, creating more slack
- PM-Clock is sub-optimal with much less complexity!
- $E_{PM\text{-}Clock} / E_{optimal} \approx 101\%$



Task Clock Frequency Assignment

- Each task has its own clock frequency setting
- At each context switch, CPU scales its clock frequency based on the corresponding frequency of the next task
- PM-Clock frequency: Sys-Clock++
  - If a higher priority task needs higher frequency to satisfy its own deadline than Ω_i, task τ_i can reduce its frequency to reduce the energy.

$\tau_1\{2, 5, 4\}$ and $\tau_2\{1, 20, 20\}$



  - $\Omega_1 = 2/4 = 0.5$

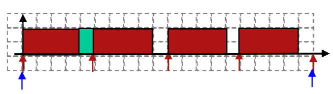
For τ_2, its completion times to consider are given by 5, 10, 15, and 20 (i.e. whenever task τ_1 arrives and τ_2's own period/deadline). Other completion times must be considered in principle, but *each* one of them will yield worse energy savings than these discrete points.

Pick the best (low) frequency obtained among these options.

  - $\Omega_2 = min\{(2+1)/5, (2*2+1)/10, (3*2+1)/15, (4*2+1)/20\} = 9/20 = 0.45$
  - $\rightarrow \tau_1$ *must run at 0.5 while* τ_2 *needs to run the system only at 0.45*
  - $\rightarrow$ *since* τ_1 *runs at 0.5,* τ_2 *can run at a frequency < 0.45!*

PM-Clock Example (cont'd.)

$\tau_1\{2, 5, 4\}$ and $\tau_2\{1, 20, 20\}$



- Now, note that there are two frequencies involved, one for each task:
- $v_1 = 0.5$

With τ_1 running at 0.5, τ_2 can complete at time 5, 10, 15 or 20

- $v_2 = min[1/(5-(2/0.5)),$ /* if τ_2 completes at t=5 */
  $1/(10-(2*2/0.5)),$ /* if τ_2 completes at t=10 */
  $1/(15-(3*2/0.5)),$ /* if τ_2 completes at t=15 */
  $1/(20-(4*2/0.5))]$ /* if τ_2 completes at t=20 */

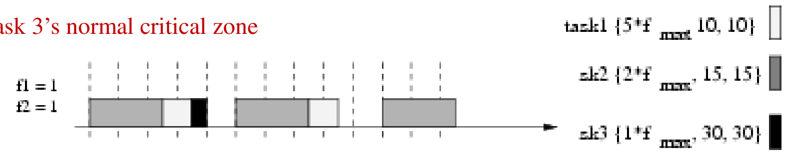
$= min[1/(5-4), 1/(10-8), 1/(15-12), 1/(20-16)]$

$= min[1, 0.5, 0.333, 0.25]$

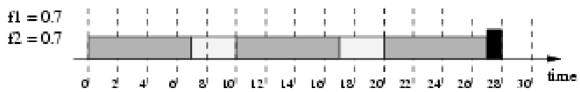
$= 0.25$

PM-Clock with 3 Tasks

Task 3's normal critical zone



Task 3's "inflated" critical zone



First, SysClock computations yield $\varepsilon_1 = 0.5$, $\varepsilon_2 = 0.7$ and $\varepsilon_3 = 0.67$.

→ $v_1 = 0.7$, $v_2 = 0.7$ and v_3 can be improved to less than 0.67.

General Algorithm for PM-Clock Calculation

// Assume a taskset has n tasks and $D_1 \leq D_2 \ldots \leq D_n$
// v_i is the task clock frequency assigned by PM-Clock
During Admission Control:
  For each task τ_i:
    $v_i = 0, \epsilon_i =$ **Energy-Min-Freq**(C_i, T_i, D_i)
  End for
  For $i = 1$ to n:
    // lp(τ_i) are tasks with priorities $\leq \tau_i$'s priority
    v_i = lowest f such that $(f/f_{max}) \geq max_{\forall j \in lp(\tau_i)} \epsilon_j$
    If (i != 1) and $(v_{i-1} > v_i)$ **then**
      $v_i = 0$
      For $j = i$ to n: ϵ_j = **Inflated-f**(C_j, T_j, D_j) **End for**
    End if
    v_i = lowest f such that $(f/f_{max}) \geq max_{\forall j \in lp(\tau_i)} \epsilon_j$
  End for

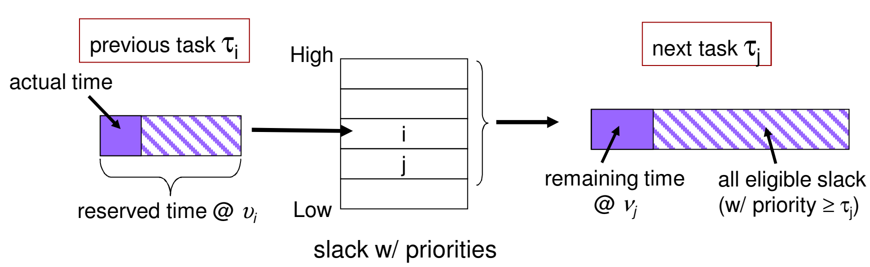
Inflated-f(C_i, T_i, D_i):
  // i_β and β are inflated and scalable workload for time t
  // δ = scalable time, IN_BZP is the busy period flag
  // $hp(\tau_i)$ are tasks with priorities $\geq \tau_i$'s priority
  $S = I = \beta = \Delta = \delta = 0, \alpha = 1$, IN_BZP = TRUE
  $\omega = C_i/f_{max}, \omega' = 0$
  Do while $(\omega < D_i)$
    If (IN_BZP == TRUE) **then**
      $\Delta = D_i - \omega$;
      Do while $(\omega < D_i)$ && $(\Delta > 0)$
        $i\_\beta = 0, \beta = 0$
        For $j = 1$ to n:
          If $(D_j \leq D_i)$ && $(v_j != 0)$) **then**
            $i\_\beta = i\_\beta + \frac{C_j}{v_j * f_{max}} * (\lfloor \frac{\omega}{T_j} \rfloor + 1)$
          Else if $(D_j \leq D_i)$ && $(v_j == 0)$) **then**
            // This task is still scalable
            $\beta = \beta + \frac{C_j}{f_{max}} * (\lfloor \frac{\omega}{T_j} \rfloor + 1)$
          End if
        End for
        $\omega' = i\_\beta + \beta + S, \Delta = \omega' - \omega, \omega = \omega'$
      End while
      IN_BZP = FALSE
    Else
      $I = min_{j \in hp(\tau_i)} \{(T_j * \lceil \frac{\omega}{T_j} \rceil) - \omega, D_i - \omega\}$
      $S = S + I, \omega = \omega + I, t = \omega, \delta = t - i\_\beta$
      If $(\frac{\beta}{\delta} < \alpha)$ **then** $\alpha = \frac{\beta}{\delta}$ **End if**
      IN_BZP = TRUE
    End if
  End while
  return α

Dynamic Frequency Scaling

- Static Voltage Scaling (SFS) assumes that all tasks use worst-case cycles specified in their reserve requirements
- Dynamic Voltage Scaling (DFS)
  - Multimedia applications in the average case use resource less than 10% of their worst-case specifications
  - Detect the earlier completed task, transfer its slack to another low priority task
  - Kernel determines the dynamic frequency of a task based on its reserve specification and available slack

Dynamic PM-Clock Algorithm

- A task may use less resource than reserved
  - Slack is transferred to first task w/ same or lower-priority
  - Reduce voltage/frequency on-the-fly



Dynamic PM-Clock Algorithm

- During admission control: set ν_i and $WCEC_i$ based on PM-Clock
- When the new request arrives: set $\nu_i^{(d)} = \nu_i$, $EC_i = 0$, $CEC_i = 0$ where EC_i is current execution time and CEC_i is completed execution time of task τ_i
- At every context switch:
  - Update EC_i
  - If τ_i completes its job early and τ_j is the next ready task with **lower** priority

$CEC_i = EC_i$

$$\nu_j^{(d)} = \frac{(WCEC_j - EC_j)/\nu_j^{(d)}}{(WCEC_j - EC_j)/\nu_j^{(d)} + (WCEC_i - CEC_i)/\nu_i} * \nu_j^{(d)}$$

Simulation Results

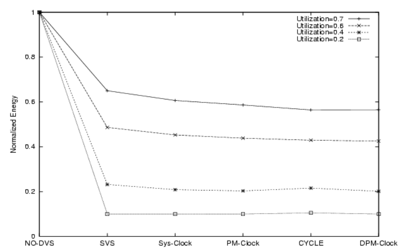
- Each task has a uniform probability of having a short *(1-10 ms)*, medium *(10-100 ms)* or long *(100-1000 ms)* period.
- The task period is uniformly distributed in each range.
- Target Study
  - The effect of Variable Processor Utilization
  - The effect of BCEC/WCEC Ratio
- Comparing Sys-Clock, PM-Clock and Dynamic PM-Clock with SVS, CYCLE
  - SVS – System Clock Frequency that complete the task exactly at the deadline
  - CYCLE – More complex dynamic voltage scaling based on SVS with $O(n^2)$ at every context switching
  - Sys-Clock, PM-Clock: $O(Mn^2)$, Dynamic PM-Clock: additional O(1) at every context switching

Energy vs. System Utilization (1 of 2)

BCEC = best-case execution time

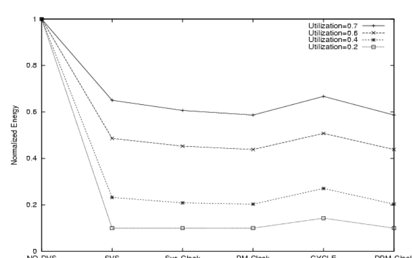
WCEC = worst-case execution time

BCEC/WCEC = 0.5

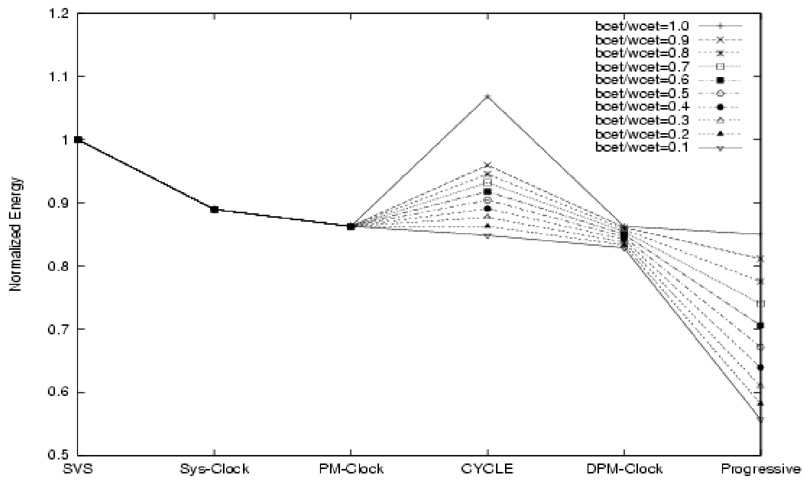


Energy vs. System Utilization (2 of 2)

BCEC/WCEC = 1.0

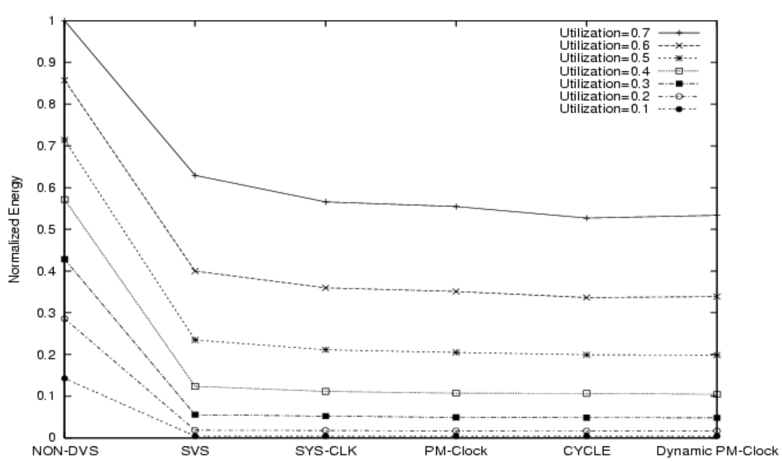


Energy vs. BCEC/WCEC

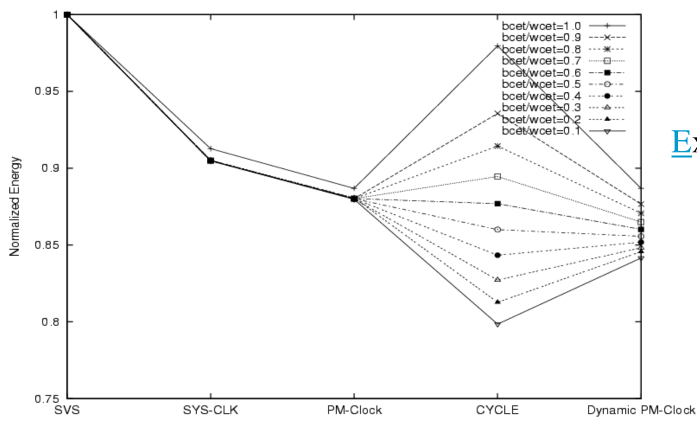


System Utilization = 0.5

The Effect of Processor Utilization at Highest Frequency



The Effect of BCET/WCET Ratio



Ratio of
Best-Case *vs*
Worst-Case
Execution Times

Effect of Hardware Limitations

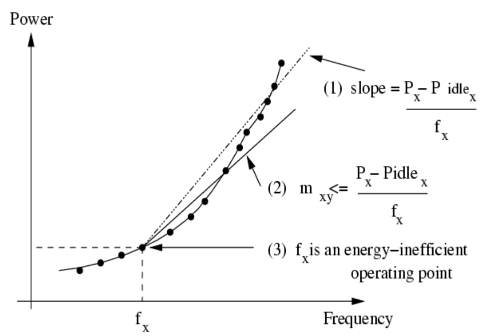
- An Ideal Voltage-Scaling Processor consumes less energy at any given smaller operating frequency
- Some commercially available processors do not!
  - Clock Frequency vs. voltage of Transmeta Crusoe processor
  -

Frequency f (MHz)	Voltage V_{DD} (V)	Relative power (%)
600	1.60	100.00
525	1.50	70.00
450	1.35	45.00
375	1.22	33.33
300	1.20	26.67
225	1.10	23.33

  - $E_{225} = P_{225} * t_{225} = 0.233 * t_{225} * P_{max}$
  - $E_{300} = P_{300} * t_{300} + P_{idle} * (t_{225} - t_{300})$
    $= (0.267 * 225/300 * t_{225} + 0.05 * 75/300 * t_{225}) * P_{max}$
    $= 0.21275 * t_{225} * P_{max} < E_{225}$

Energy-Efficient Operating Frequency

- Running Crusoe processor at 300 MHz consumes **less** energy than at 225 MHz.
- The frequency 225 MHz is an *energy-inefficient* operating frequency



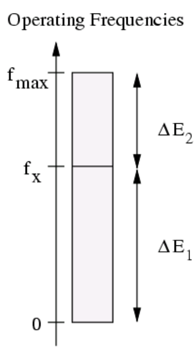
Complexity of finding inefficient frequencies:

- **O(log n)** for a convex non-decreasing power-frequency processor
- $O(n^2)$ for the most general case
- *n* is the number of available operating frequencies

The Effect of Finite Operating Frequencies

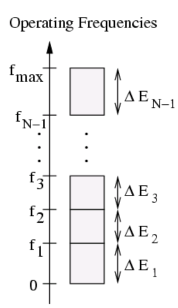
- CPU needs to operate at higher clock frequencies to satisfy the timing constraints if the expected freq. is not available.
- More energy is consumed: *Energy loss* due to energy quantization error
- Goal: Investigate an operating frequency grid which minimizes the worst-case energy quantization error

2-Point Operating Frequencies



- Let ΔE_1 and ΔE_2 be the worst-case energy quantization error when $f_{opt} \in (0, f_x]$ and $(f_x, f_{max}]$
- The worst ΔE_1 occurs when $f_{opt} = \varepsilon$ and $\varepsilon \rightarrow 0$ to compute the task with λ cycles
  - $\Delta E_1 = E_{2\text{-}point} - E_{ideal}$
    $= [k\lambda f_x^2 + P_{idle}(1/\varepsilon - 1/f_x)] - k\lambda\, \varepsilon^2$
- The worst ΔE_2 occurs when $f_{opt} = f_x + \varepsilon$ and $\varepsilon \rightarrow 0$ to compute the task with λ cycles
  - $\Delta E_2 = E_{2\text{-}point} - E_{ideal}$
    $= [k\lambda f_{max}^2 + P_{idle}(1/(f_x+\varepsilon) - 1/f_{max})] - k\lambda\, (f_x+\varepsilon)^2$
- Solving $\Delta E_1 = \Delta E_2$
  - $f_x = f_{max} / \sqrt{2}$ *assuming that P_{idle} is negligible*

N-Point Operating Frequency



In the worst case, having N operating points is 1/N away from the ideal case (of infinite operating points)

$$\forall i \in [1, N-1] : f_i = \sqrt{\frac{i}{i+1}} f_{i+1}$$
$$f_N = f_{max}$$

$$\Delta E = \frac{k\lambda f_{max}^2}{N} = \frac{E_{noDVS}}{N}$$

Summary

- **Sys-Clock** is the optimal system clock frequency assignment with complexity $O(Mn^2)$ at the admission control
- **PM-Clock** is a task clock frequency assignment with same complexity at the admission control
  - The clock frequency assigned to a task is greater than or equal to that assigned to a lower priority task
- **Dynamic PM-Clock** with addition O(1) at every context switch
- **PM-Clock** and **Dynamic PM-Clock** reduce energy by 71% at 50% system utilization
- Determine Energy-inefficient operating frequencies which should be excluded from voltage-scaling algorithm
- Determine the optimal clock frequency grid to minimize energy quantization error when the # of operating frequencies is limited

Clarifications