Energy-aware computing

Giorgio Buttazzo

g.buttazzo@sssup.it

Retis
Real-Time Systems Laboratory

Scuola Superiore Sant'Anna

Context

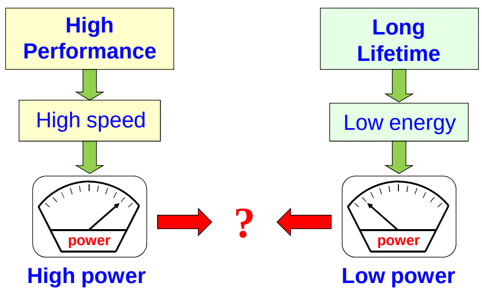
More a more devices are powered by battery:

Required features:
- High performance
- Long lifetime

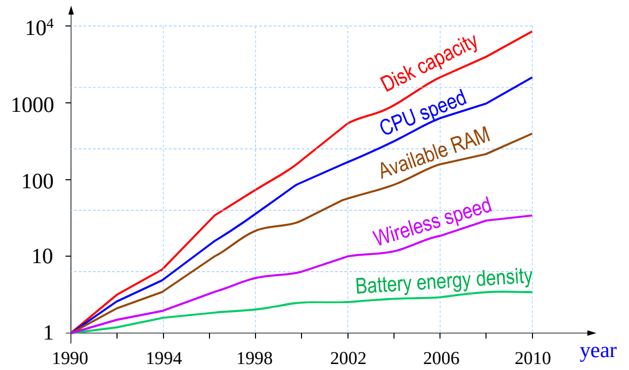
2

Contrasting objectives

The problem is not trivial, because performance and lifetime have opposite energy requirements:



Progress of components



4

How to increase lifetime?

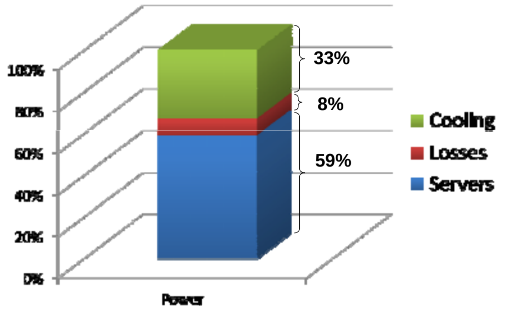
Considering the limited progress of batteries, the only hope to increase system lifetime is to reduce energy consumption by proper power management.

- In real life, and also in embedded systems, a lot of energy is wasted due to bad power management.
- Research work is needed to optimize resource usage and reduce waste.

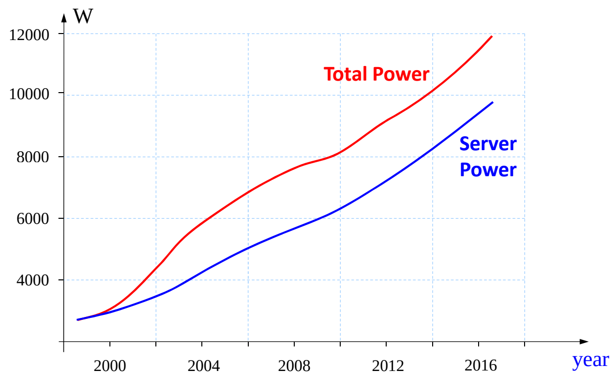
5

Same problem in data centers

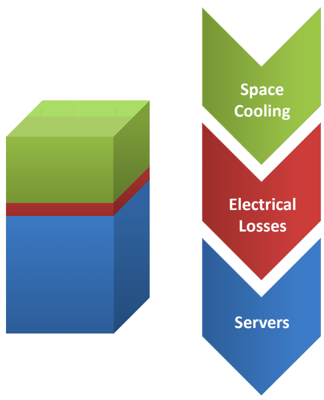
Consumption in data centers



Consumption in data centers



How to reduce?

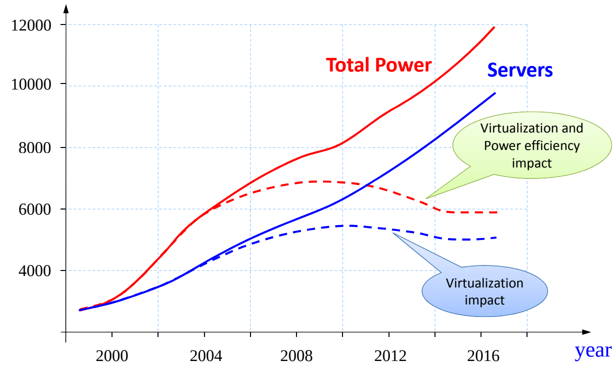


- **Outside air economizers**
- **Expand temperature setpoints**
- **Efficient cooling equipment**

- **Efficient hardware**
- **Optimized distribution**
- **Efficient voltage regulators**

- **Virtualization**

Consumption in data centers

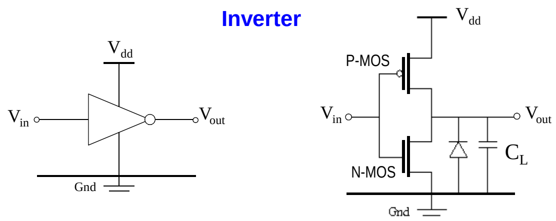


Power model

Power dissipation in CMOS integrated circuits is mainly due to two causes:

- **Dynamic power** (P_d) consumed during operation;
- **Static power** (P_s) consumed when the circuit is off.

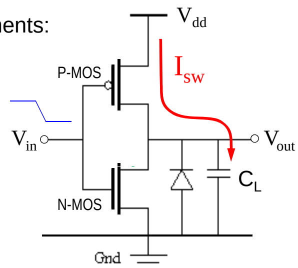
Inverter



Dynamic power

Dynamic power has two components:

1. Switching power P_{sw}
consumed during logic state change (1 → 0) to charge the load capacitance C_L.

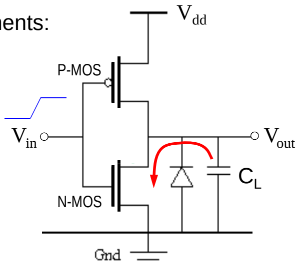


Dynamic power

Dynamic power has two components:

1. Switching power P_{sw}
consumed during logic state change (1 → 0) to charge the load capacitance C_L.

Note that during transition (0 → 1) the capacitance is discharged through the N-MOS.

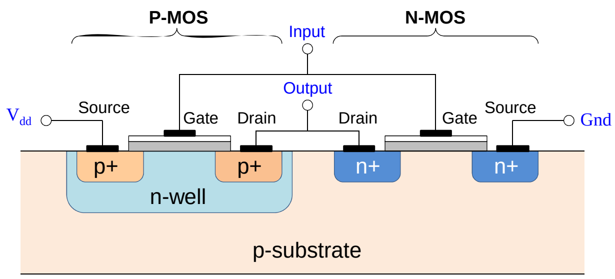


The switching power can be expressed by:

$$P_{sw} \propto C_L \cdot f \cdot V_{dd}^2$$

f = clock frequency

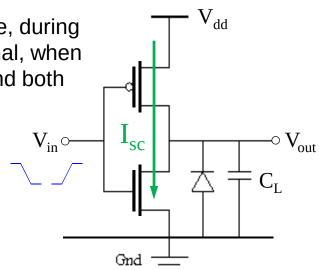
CMOS Inverter



Dynamic power

2. Short circuit power P_{sc}
consumed for a very short time, during the ramp time of the input signal, when input is at threshold voltage and both PMOS and NMOS are ON.

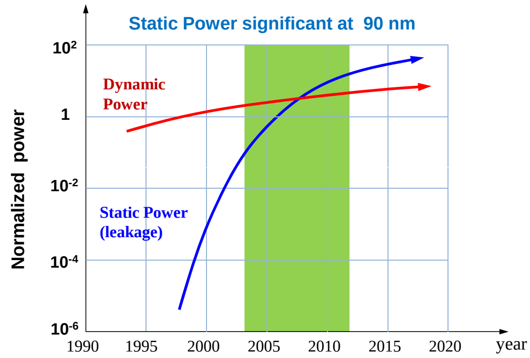
$$P_{sc} \propto V_{dd} I_{sc}$$



Hence, the total dynamic power is dominated by the switching power:

$$P_d = P_{sw} + P_{sc} \cong C_L \cdot f \cdot V_{dd}^2$$

Dynamic vs. static power

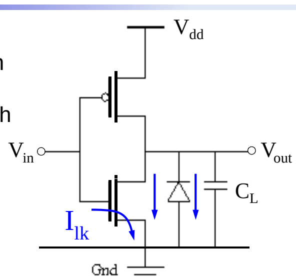


Gate length (nm): 500 350 250 180 130 90 65 45 22

Static power

Static power P_s
is due to a quantum phenomenon where mobile charge carriers (electrons or holes) tunnel through an insulating region, creating a leakage current I_{lk}

$$P_s \propto V_{dd} I_{lk}$$



- Static power consumption is independent of the switching activity is always present if the circuit is on.
- As devices scale down in size, gate oxide thicknesses decreases, resulting in larger leakage current.

In summary

- The dynamic power consumption increases with the supply voltage and with the clock frequency:

$$P_d \propto C_L \cdot f \cdot V_{dd}^2$$

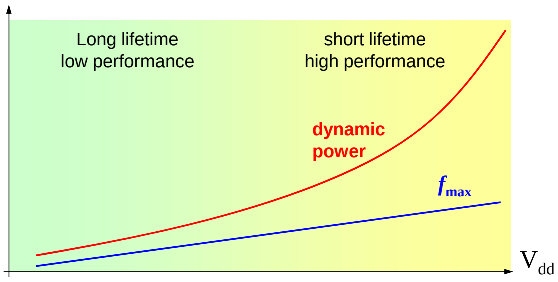
- Moreover, the supply voltage also affects the circuit delay (hence the max clock frequency):

$$D \propto \frac{V_{dd}}{(V_{dd} - V_t)^2} \qquad V_t = \text{threshold voltage}$$

Note that D decreases for higher V_{dd} and lower V_t

Dynamic Volt./Freq. scaling

- Hence, the dynamic power consumed by a system can be controlled by scaling the clock frequency and the voltage at which the processor operates:



Low-power features

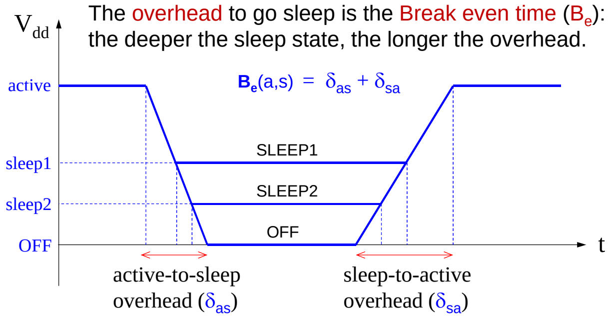
To exploit such a possibility, modern processors are designed to

- work under different operating modes, each characterized by a power consumption P, voltage V and clock frequency f: (P_1, V_1, f_1), (P_2, V_2, f_2), …, (P_m, V_m, f_m)
  - ⇒ Switching between two modes j-k is characterized by a power consumption P_{jk} and time overhead δ_{jk}
- have different low-power states, each characterized by a specific power consumption and transition overheads: $S_1(P_1, \delta_{1as}, \delta_{1sa})$, … $S_L(P_L, \delta_{Las}, \delta_{Lsa})$

Dynamic Power Management

- On the other hand, static power can be controlled by turning the CPU off, or putting it in a sleep state:

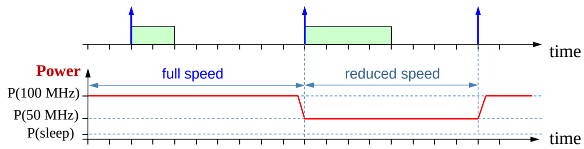
The overhead to go sleep is the Break even time (B_e): the deeper the sleep state, the longer the overhead.



Energy-saving methods

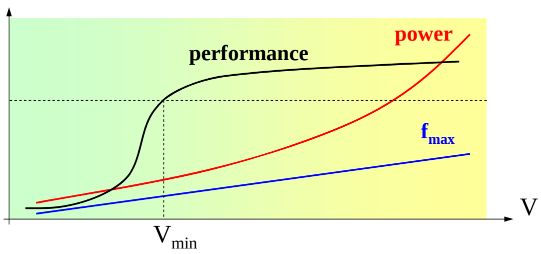
DVFS: Dynamic Voltage and Frequency Scaling

The consumed energy is varied by acting on the supply voltage and clock frequency:



Minimizing energy

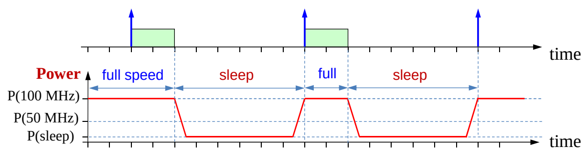
In real-time systems, the problem is to minimize energy consumption still guaranteeing a desired level of performance (schedule feasibility).



Energy-saving methods

DPM: Dynamic Power Management

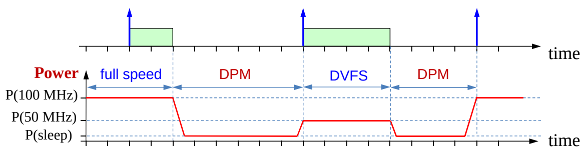
The consumed energy is varied by exploiting the inactive low-power states of the processor:



Energy-saving methods

Hybrid: DVFS + DPM

The consumed energy is varied by exploiting both techniques in different time intervals:



Power model

To take different components into account, power consumption can be modeled as follows [Martin & Siewiorek, 2001]:

$$P(s) = K_3 s^3 + K_2 s^2 + K_1 s + K_0$$

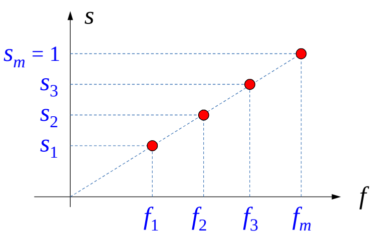
- K_3 expresses the weight of the power components that vary with both voltage and frequency.
- K_2 captures the nonlinearity of DC-DC regulators in the range of the output voltage.
- K_1 is related to the hardware components that can only vary the clock frequency (but not the voltage).
- K_0 represents the power consumed by the components that are not affected by the processor speed.

Normalized speed

To make the analysis more general, instead of using the absolute clock frequencies, f_1, f_2, …, f_m

it is better to use a normalized speed $s \in [0,1]$:

$$s = \frac{f}{f_m}$$



WCET scaling

CC_i = number of clock cycles required by τ_i

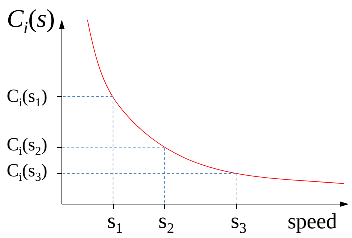
C_i = task computation time

$$C_i(s) = \frac{CC_i}{s} = \frac{C_i^1}{s}$$

where

$$C_i^1 = C_i(s=1) = CC_i$$

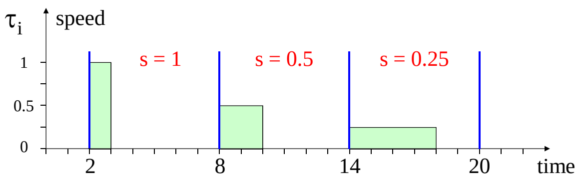
is the shortest execution time achievable at the maximum speed



Notation

When dealing with processors with variable speed, often the schedule is represented in a bi-dimensional diagram, where time is on the x-axis and normalized speed is on the y-axis.

For instance, the following schedule represents 3 jobs of a periodic task τ_i with period $T_i = 6$ and WCET at the maximum speed $C_i(1) = 1$, executed at three decreasing speeds:



WCET scaling

In practice, several operations are performed on I/O devices and memory units that do not share the clock with the CPU.

⇒ For instance, hard disk operations mostly depend on the bus clock frequency, the hard disk read/write speed, and the interference caused by other tasks accessing the bus.

Hence, a more realistic model for the task WCET is:

$$C_i(s) = C_i^{fix} + \frac{C_i^{var}}{s}$$

WCET scaling

Note, however, that

$$C_i(s) = C_i^{fix} + \frac{C_i^{var}}{s}$$ it is more precise, but it complicates the analysis

$$C_i(s) = \frac{C_i^1}{s}$$ it is safe, because it represents an upper bound of the previous model

In fact, since $C_i^1 = C_i(1) = C_i^{fix} + C_i^{var}$

we have: $\frac{C_i^{fix}}{s} + \frac{C_i^{var}}{s} \geq C_i^{fix} + \frac{C_i^{var}}{s}$ for any $s \leq 1$

The energy saving problem

In general, we always have that:

scaling up ⇒ shorter execution, higher power consumption

scaling down ⇒ longer execution, lower power consumption

But we are interested in consuming less energy, not less power.

When a processor is active at speed s for a time t,

- the consumed power is P(s)
- the consumed energy is E(s) = P(s) · t

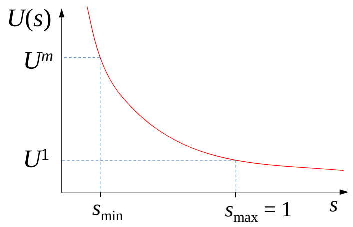
Utilization scaling

Note that, if using the simplified model $C(s) = C^1/s$:

$$U(s) = \sum_{i=1}^{n} \frac{C_i(s)}{T_i} = \sum_{i=1}^{n} \frac{C_i^1}{sT_i} = \frac{U^1}{s}$$

where: $U^1 = \sum_{i=1}^{n} \frac{C_i^1}{T_i}$

is the task set utilization at $s_{max} = 1$



Energy per cycle

Since C_i is a function of the speed, the energy consumed for executing a task τ_i at speed s is:

$$E_i(s) = P(s) \cdot C_i(s)$$

For example, if we consider $C_i(s) = \frac{CC_i}{s}$

we have: $E_i(s) = P(s)\frac{CC_i}{s}$

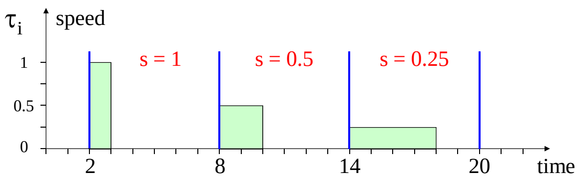
Therefore, what we actually need to minimize is the

Energy per cycle: $$E_c(s) = \frac{P(s)}{s}$$

The energy saving problem

What is the best processor speed *s* that guarantees the application feasibility and minimizes energy consumption?

For example, it is better to execute a task as fast as possible for a short time, or as slow as possible for long time?



Optimal speed

The speed that minimizes the energy per cycle is called optimal speed (or energy efficient speed) s*.

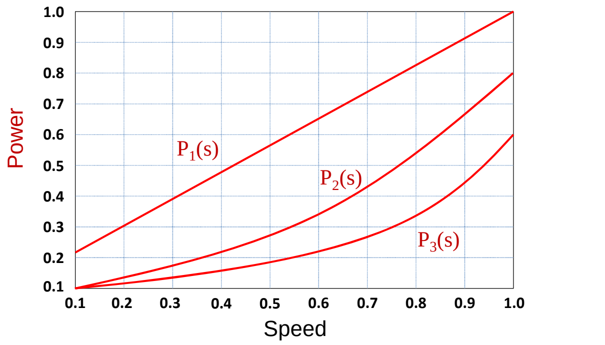
$$P(s) = K_3 s^3 + K_2 s^2 + K_1 s + K_0$$

$$E_c(s) = \frac{P(s)}{s} \Rightarrow E_c(s) = K_3 s^2 + K_2 s + K_1 + \frac{K_0}{s}$$

The value of the optimal speed depends on the specific architecture (i.e., the specific values of K_0, K_1, K_2, K_3).

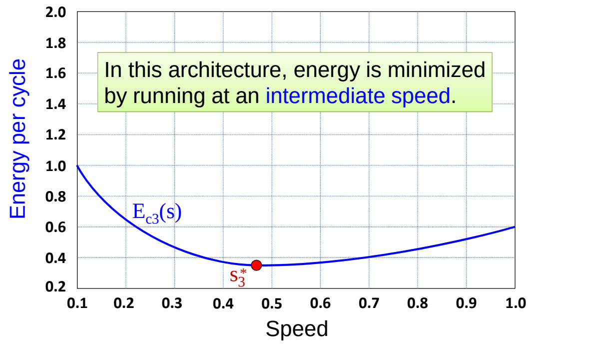
Examples

$P_1(s) = 0.9\,s + 0.1$ $P_2(s) = 0.5\,s^2 + 0.3\,s$ $P_3(s) = 0.5\,s^3 + 0.1$



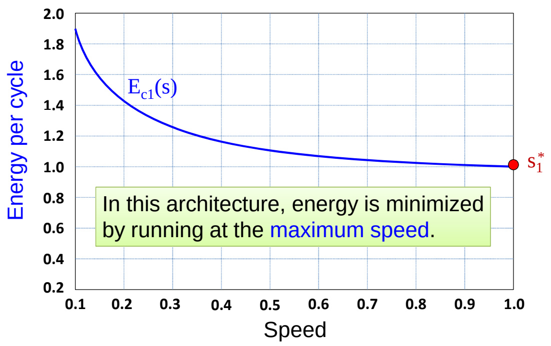
Architecture 3

$P_3(s) = 0.5\,s^3 + 0.1$ $E_{c3}(s) = 0.5\,s^2 + 0.1/s$



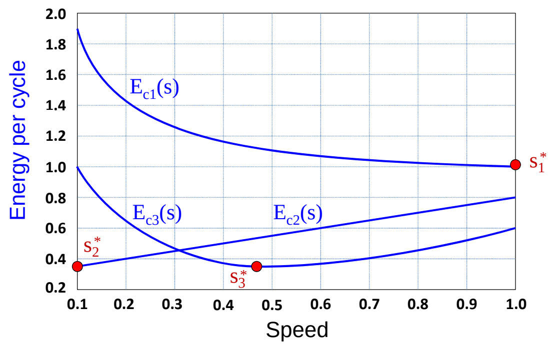
Architecture 1

$P_1(s) = 0.9\,s + 0.1$ $E_{c1}(s) = 0.9 + 0.1/s$



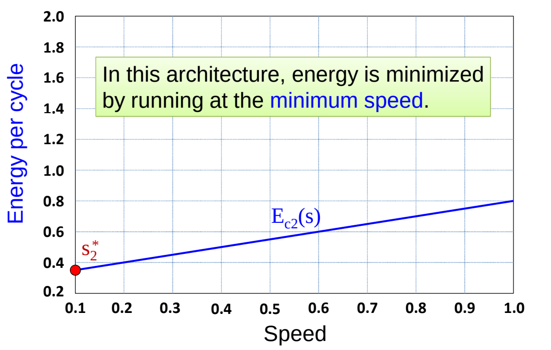
Summary

$E_{c1}(s) = 0.9 + 0.1/s$ $E_{c2}(s) = 0.5\,s + 0.3$ $E_{c3}(s) = 0.5\,s^2 + 0.1/s$



Architecture 2

$P_2(s) = 0.5\,s^2 + 0.3\,s$ $E_{c2}(s) = 0.5\,s + 0.3$



Observations

The energy-aware strategy depends on the specific architecture.

Further energy saving can be achieved by exploiting low-power states.

Problem 1

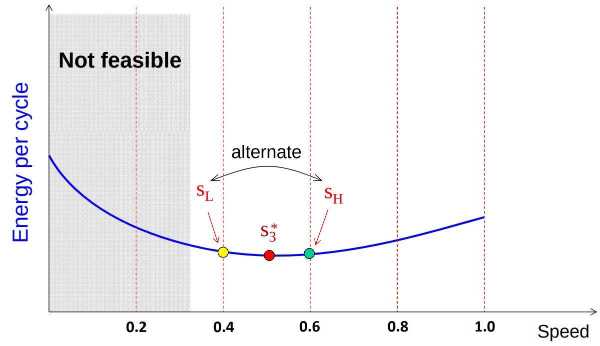
What to do if the task set is not feasible at s*?

Possible solutions

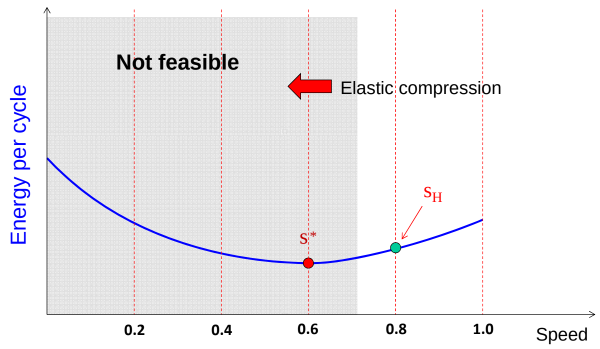
1. Select a higher speed s_H as the smallest speed s_H > s*
   Then apply DPM to exploit extra idle times.

2. Compress task utilizations (e.g., by applying elastic scheduling) so that the task set is feasible at s*

Problem 2

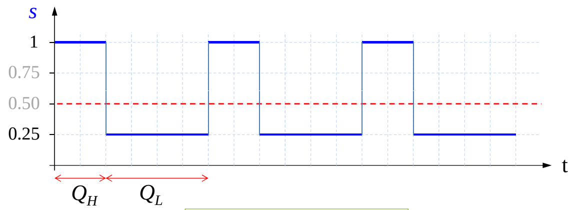


Problem 1



PWM-like execution

Suppose that only two speeds are available ($s_L = 0.25$, $s_H = 1$), but the optimal speed minimizing energy is $s^* = 0.5$.



$$s_{eq} = \frac{s_H Q_H + s_L Q_L}{Q_H + Q_L}$$

Problem 2

What to do if s* is not available on the platform?
This only applies to type-3 architectures

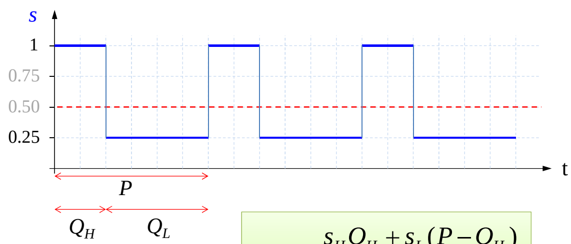
Possible solutions

1. Select a higher speed s_H as the smallest speed s_H > s*
   Then apply DPM to exploit the extra idle times.

2. Alternate execution between two adjacent speeds (s_L, s_H) to approximate s*

PWM-like execution

Given (s_L, s_H), how to find (Q_L, Q_H) that produce s_{eq}?



$$s_{eq} = \frac{s_H Q_H + s_L (P - Q_H)}{P}$$

PWM-like execution

Given (s_L, s_H), how to find (Q_L, Q_H) that produce s_{eq}?

$$s_{eq} = \frac{s_H Q_H + s_L (P - Q_H)}{P} = (s_H - s_L)\frac{Q_H}{P} + s_L$$

$$Q_H = P\left(\frac{s_{eq} - s_L}{s_H - s_L}\right)$$

Example 1

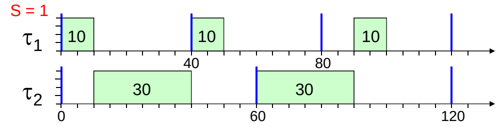
Suppose that the CPU has the following five modes of operation:

Mode	Power (mW)
ACTIVE (s = 1)	100
ACTIVE (s = 0.75)	60
ACTIVE (s = 0.5)	30
ACTIVE (s = 0.25)	15
SLEEP	4

And consider the following application:

$C_1^1 = 10$ τ_1

$C_2^1 = 30$ τ_2

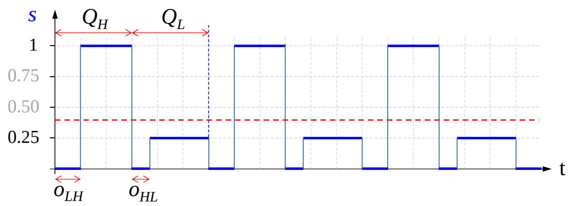


$$U^1 = \sum_{i=1}^{n} \frac{C_i^1}{T_i}$$

$U^1 = 1/4 + 1/2 = 0.75$

PWM-like execution

Considering transition overhead:



$$s_{eq} = \frac{s_H (Q_H - o_{LH}) + s_L (Q_L - o_{HL})}{Q_H + Q_L}$$

Further details on:

- E. Bini, G. Buttazzo, G. Lipari, "Minimizing CPU energy in real-time systems with discrete speed management", ACM Trans. on Embedded Computing Systems, 8(4), 2009.

Example 1

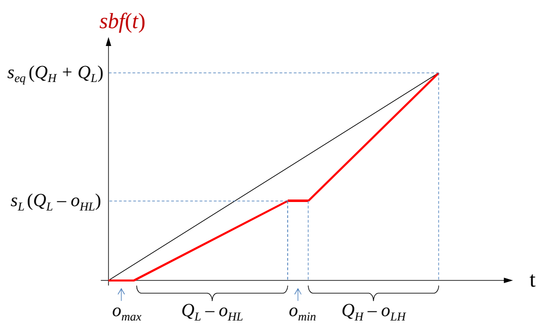
Note that: $$U(s) = \sum_{i=1}^{n} \frac{C_i(s)}{T_i} = \sum_{i=1}^{n} \frac{C_i^1}{sT_i} = \frac{U^1}{s}$$

For the feasibility (under EDF), it must be: $$U(s) = \frac{U^1}{s} \leq 1$$

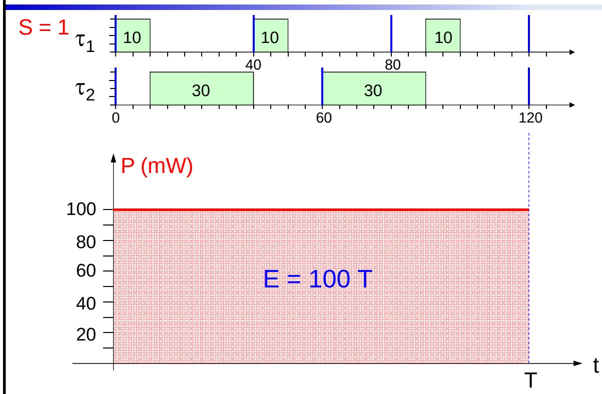
Hence, it must be: $s \geq U^1 = 0.75$

Therefore, the only feasible speeds for the given application are s = 1 and s = 0.75.

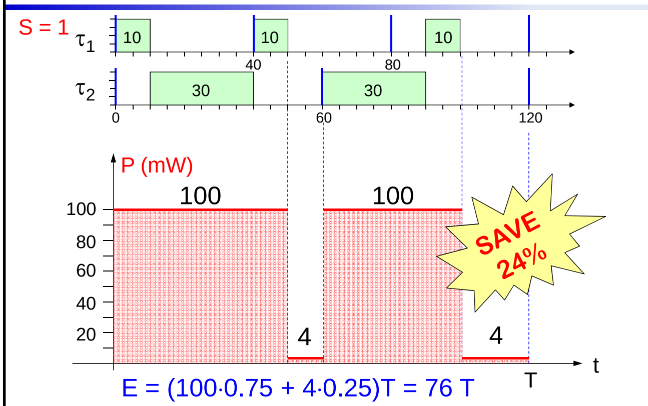
Supply function



Executing at s = 1

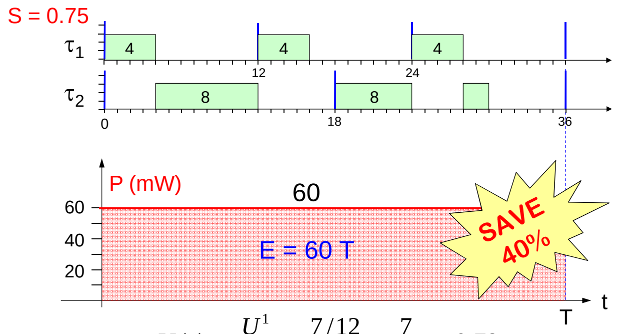


Exploiting the sleep state



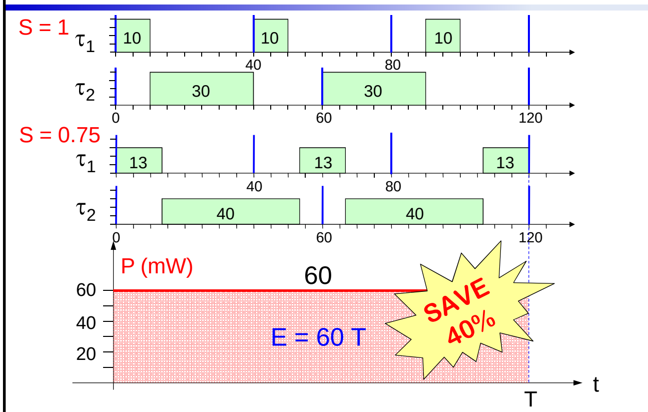
E = (100·0.75 + 4·0.25)T = 76 T

Executing at s = 0.75

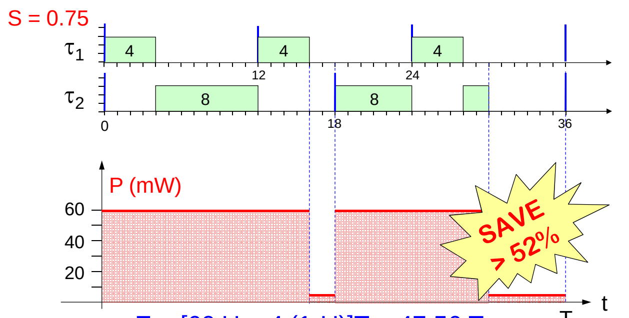


$$U(s) = \frac{U^1}{s} = \frac{7/12}{0.75} = \frac{7}{9} = 0.78$$

Executing at s = 0.75

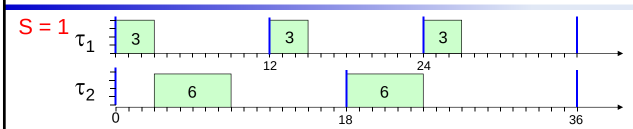


s = 0.75 + sleep state



E = [60·U + 4·(1-U)]T = 47.56 T

Example 2



$U^1 = 1/4 + 1/3 = 0.583$

Mode	Power (mW)
ACTIVE (s = 1)	100
ACTIVE (s = 0.75)	60
ACTIVE (s = 0.5)	30
ACTIVE (s = 0.25)	15
SLEEP	4

In this case, it must be:

$$s \geq U^1 = 0.583$$

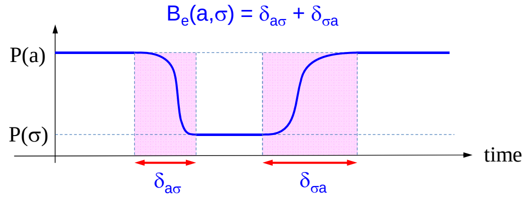
Therefore, the feasible speeds that guarantee the feasibility of the application are s = 1 and s = 0.75.

Break even time

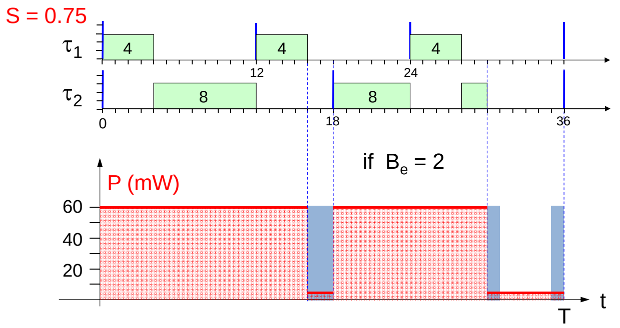
Considering the overheads involved in transitions, the minimum interval that justifies a transition to a low-power state is called *Break-even time*.

If $\delta_{a\sigma}$ and $\delta_{\sigma a}$ are the time overheads required to perform a complete transition from an active state a to a sleep state σ and back, the *Break-even time* is given by:

$$B_e(a,\sigma) = \delta_{a\sigma} + \delta_{\sigma a}$$



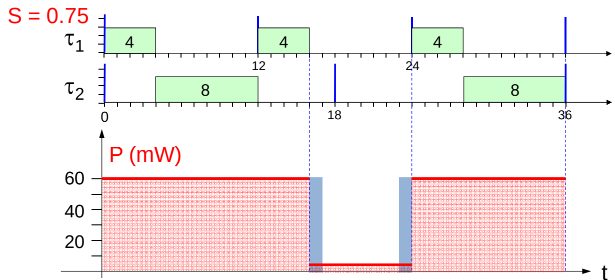
s = 0.75 + sleep state



E = [60·U + 4·(1-U) + 2(B_e/T)(60-4)]T = 53.78 T

Compacted idle times

So, instead of executing tasks according to the scheduler, tasks are delayed to make idle times as large as possible.



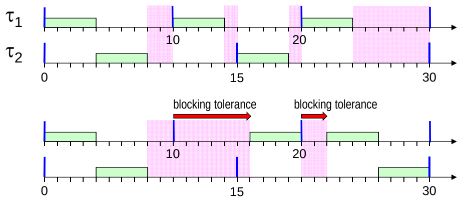
E = [60·U + 4·(1-U) + (B_e/T)(60-4)]T = 50.67 T T

Compacting idle times

To minimize the transition overhead and better exploit sleep states, it is better to switch for long time intervals, rather then switching several times for short intervals.

[Bambagini et al., SIES 2013]

proposed a technique to prolong idle intervals by delaying start times as much as possible, using blocking tolerance.

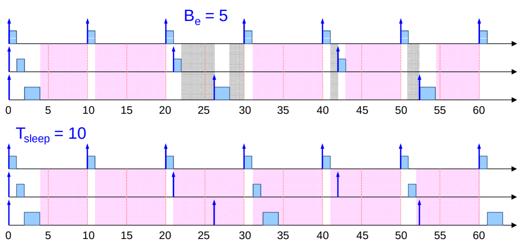


Task harmonizing

[Rowe et al., TII-6(3), 2010]

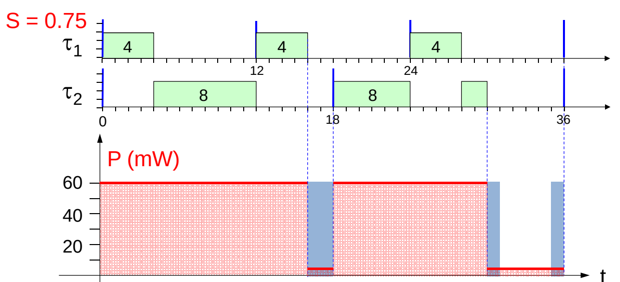
merge idle intervals by a virtual sleep task τ_{sleep} with period T_H (harmonizing period).

Jobs become eligible at the next nearest activation of τ_{sleep}.



Idle time from the scheduler

So, instead of executing tasks according to the scheduler, tasks are delayed to make idle times as large as possible.

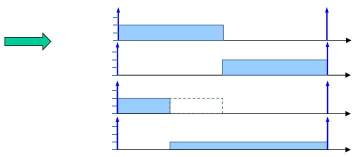


E = [60·U + 4·(1-U) + 2(B_e/T)(60-4)]T = 53.78 T T

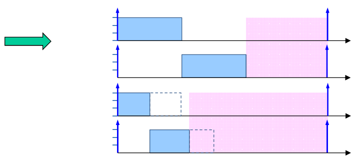
Exploiting early completions

Since mostly jobs execute much less than their WCET, the saved execution can be exploited to further reduce the speed or prolong sleep intervals (depending on the architecture).

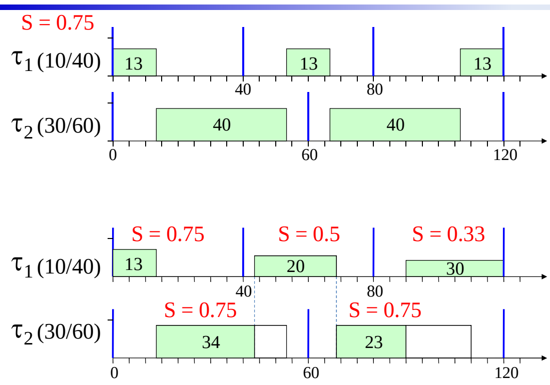
If (s* < 1)



If (s* = 1)



Early completions on DVFS



Early completions on DPM