An Optimal Real-Time Voltage and Frequency Scaling for Uniform Multiprocessors

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Gabriel Moreno
Dionisio de Niz

August, 2012

Motivation

Chip multiprocessors are the way to deal with increasing computational load in embedded real-time systems

- Power consumption, heat dissipation, and other physical constraints render single processors impractical

Power consumption is a concern in battery-powered real-time systems

- battery life time
- battery weight

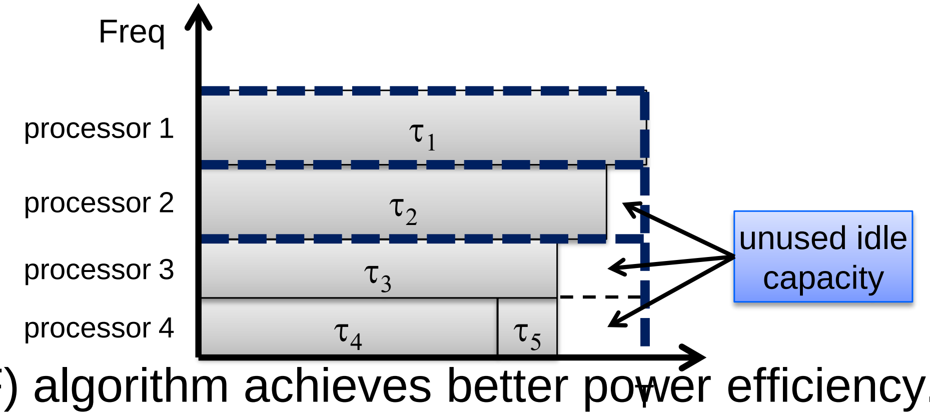
Overview

Voltage and frequency scaling (VFS) allows reducing the power consumption of a processor, and its speed.

VFS in real-time systems must ensure that the system remains schedulable.

Existing VFS algorithms for multiprocessors leave unused idle capacity.

- processor constraints
- algorithm constraints



Growing Minimum Frequency (GMF) algorithm achieves better power efficiency.

- removes algorithm constraint
- reduces impact of processor constraints

Problem Description

Given:

- multiprocessor platform supporting independent VFS, and
- a set of periodic tasks with implicit deadlines

Compute:

- frequency assignment that minimizes power consumption while meeting tasks' deadlines

Processor Power-Frequency Relationship

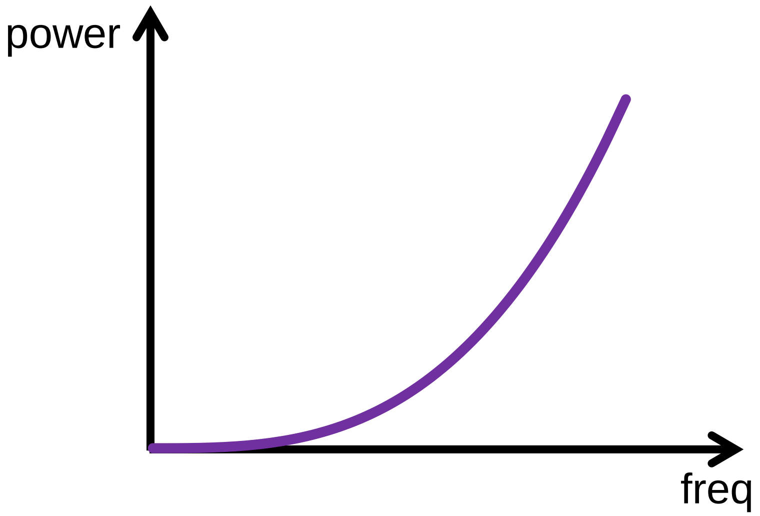
Dynamic power in processors is proportional to the product of the processor frequency and the square of the supply voltage.

$$P \propto V^2 f$$

In VFS, power can be reduced by reducing the frequency, which allows a corresponding reduction in the voltage.

Since voltage is proportional to the frequency we can approximate as

$$P \propto f^3$$



Task and Platform Model

Tasks

n : number of tasks

C_i : execution time of task τ_i, measured at the highest frequency

T_i : period of task τ_i

$D_i = T_i$: implicit deadlines

$u_i = {}^{C_i}/_{T_i}$: utilization of task τ_i

$U = \sum_{i=1}^{n} u_i$: total utilization

Platform:

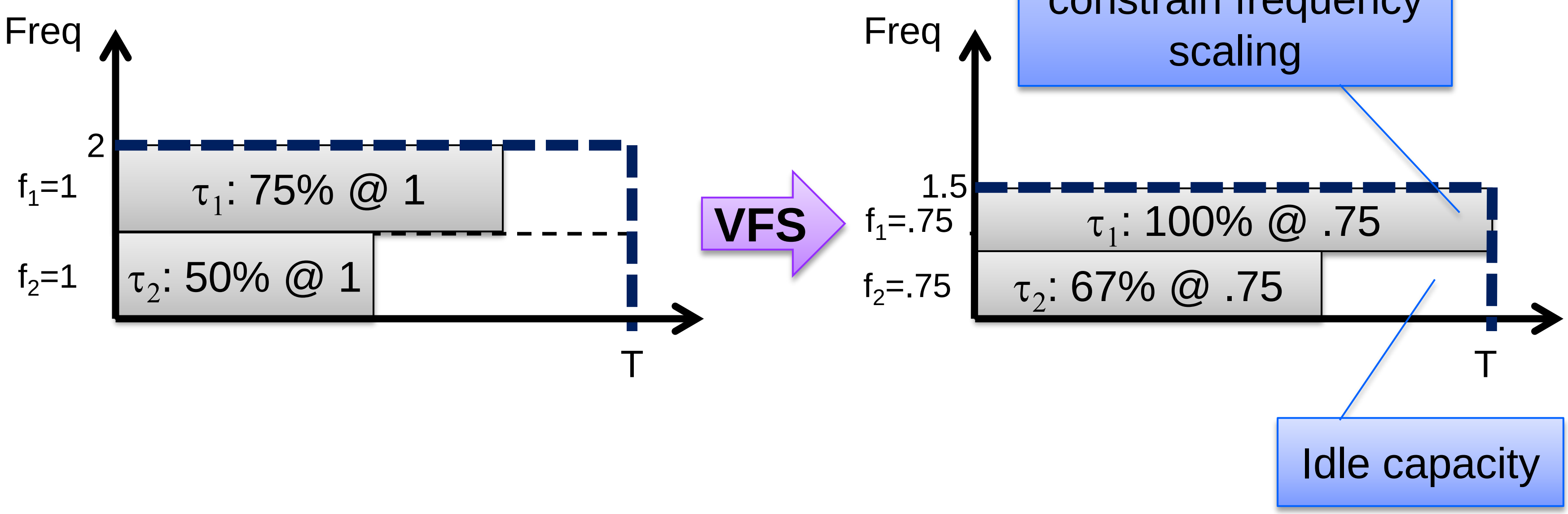
m : number of processors (all identical)

f_i : normalized frequency (1 = highest frequency) for processor *i*

Uniform Frequency Scaling

- All processors assigned the same frequency
- Tasks scheduled with an optimal global scheduler (e.g., LNREF)

$$f = \max\left(U/m, u_1, \ldots, u_n\right)$$



Non-Uniform Frequency Scaling 1

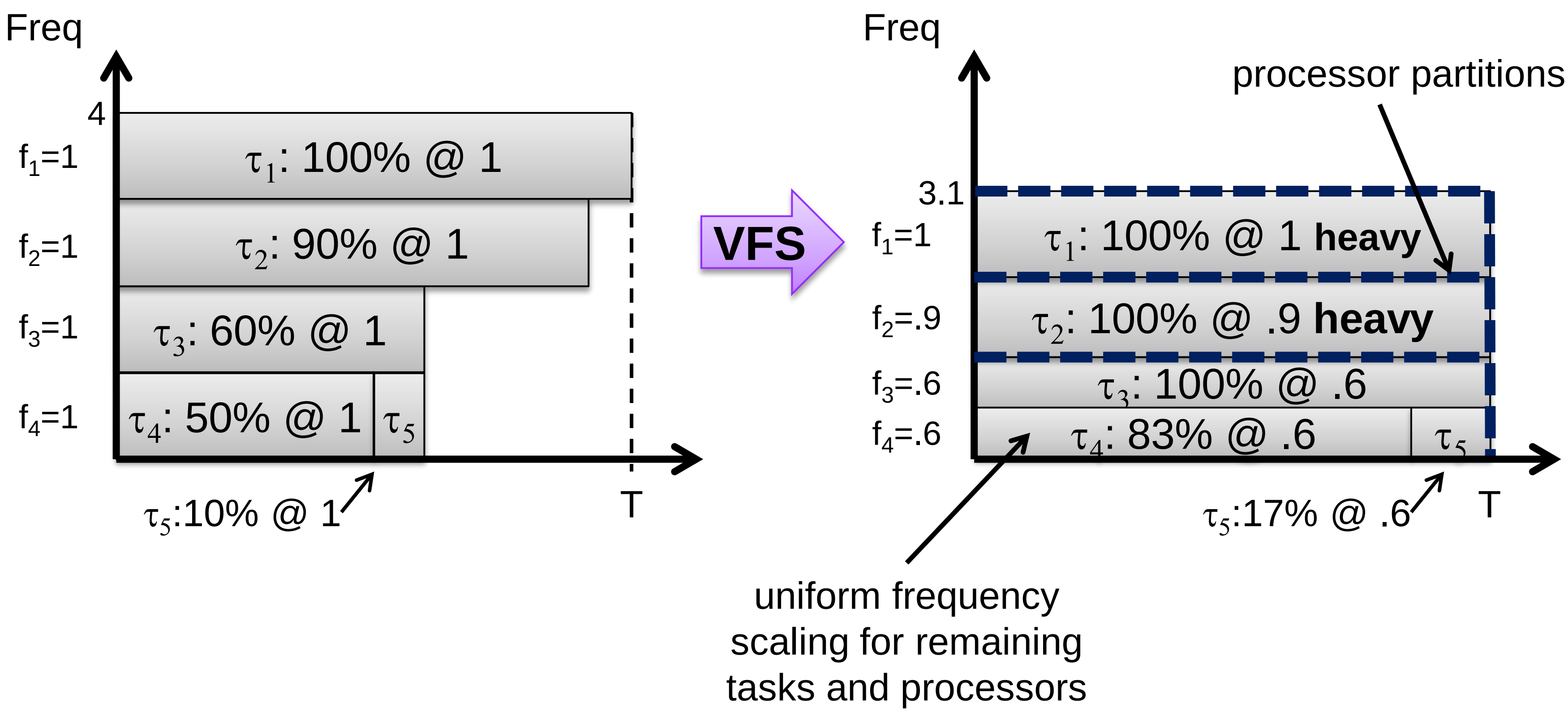
- Processor frequencies are assigned independently

Decide Independent Frequency algorithm avoids heavy task bottleneck [Funaoka 2008]

- Task τ_i is heavy if its utilization would drive up the uniform frequency assignment for the remaining processors, i.e. $u_i > \frac{\sum_{j=i}^{n} u_j}{m-i+1}$
- Each heavy task is assigned its own processor
- Remaining light tasks globally scheduled in remaining processors with uniform frequency assignment

Non-Uniform Frequency Scaling 2

Decide Independent Frequency is optimal if frequency can be scaled continuously (i.e. to any frequency in a range)

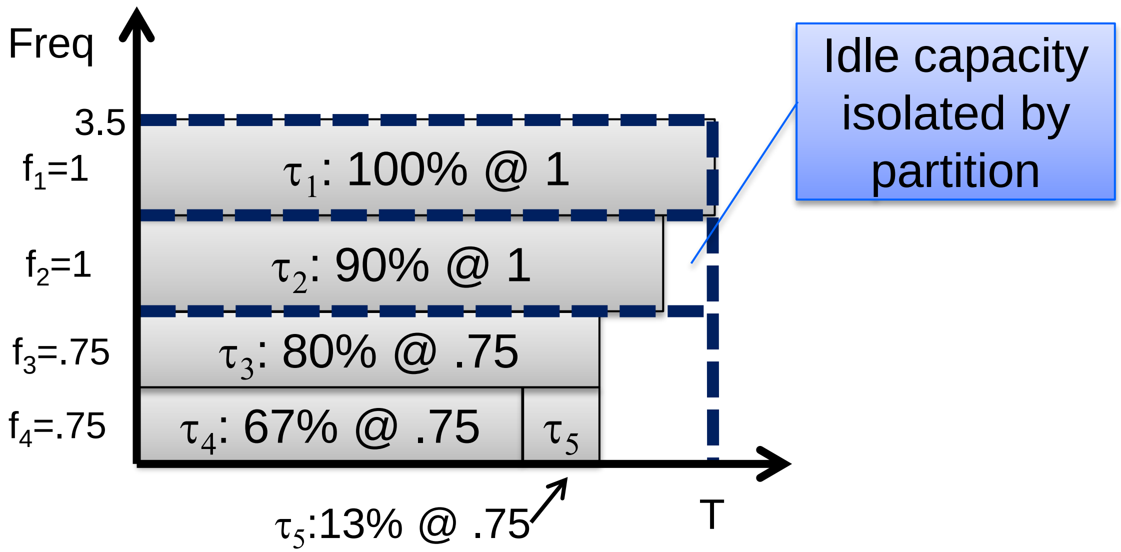


Discrete Frequency Steps

However, processors support a limited number of frequencies.

- DIF is not optimal in that setting
- Computing the optimal partition of processors and the frequency assignment is NP-Hard

Example: supported frequencies: 1, .75, .5



Achieving Better Power Efficiency

Two problems

- Discrete frequency steps force leaving idle capacity in processor partitions
- Unused capacity in a processor partition cannot be used by tasks assigned to other partitions

Observation: if we can optimally schedule tasks allowing them to migrate between processors running at different frequencies we can do better

- avoid the set partition problem (and its computational complexity)
- achieve better power efficiency
  - no fragmentation of platform capacity
  - capacity left by heavy tasks is not wasted

U-LLREF

U-LLREF [Funk 2010] is an optimal global scheduling algorithm for uniform multiprocessors

- an extension of LLREF (a DP-fair algorithm)
- processors can run at different frequencies

A task set is schedulable by U-LLREF on a platform if the following holds

$$\sum_{i=1}^{k} u_i \leq \sum_{i=1}^{k} f_i \qquad \forall k \in \{1, \ldots, m-1\}$$

$$\sum_{i=1}^{n} u_i \leq \sum_{i=1}^{m} f_i$$

where $\boldsymbol{u_1} \geq \cdots \geq \mathbf{u_n}$ and $\boldsymbol{f_1} \geq \cdots \geq \mathbf{f_m}$

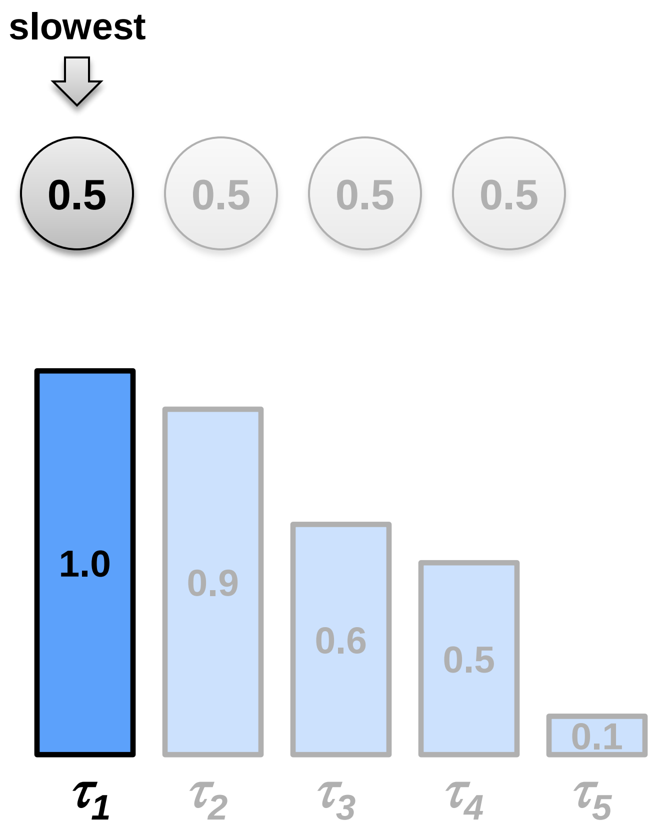
Growing Minimum Frequency Algorithm

Overview: satisfy each condition of the U-LLREF test using the most power efficient assignment of frequencies (lowest possible and distributed as uniformly as possible)

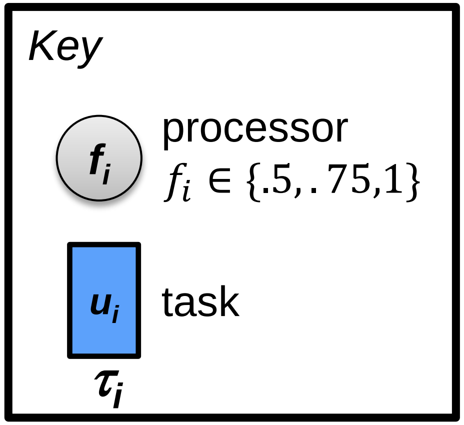
```
assign the lowest frequency to all the processors
for k = 1 to m do
  while kth U-LLREF condition not satisfied do
    increase the frequency of the slowest
       processors in subset 1..k to the next
       frequency step
  end while
end for
```

GMF Example 1

k = 1

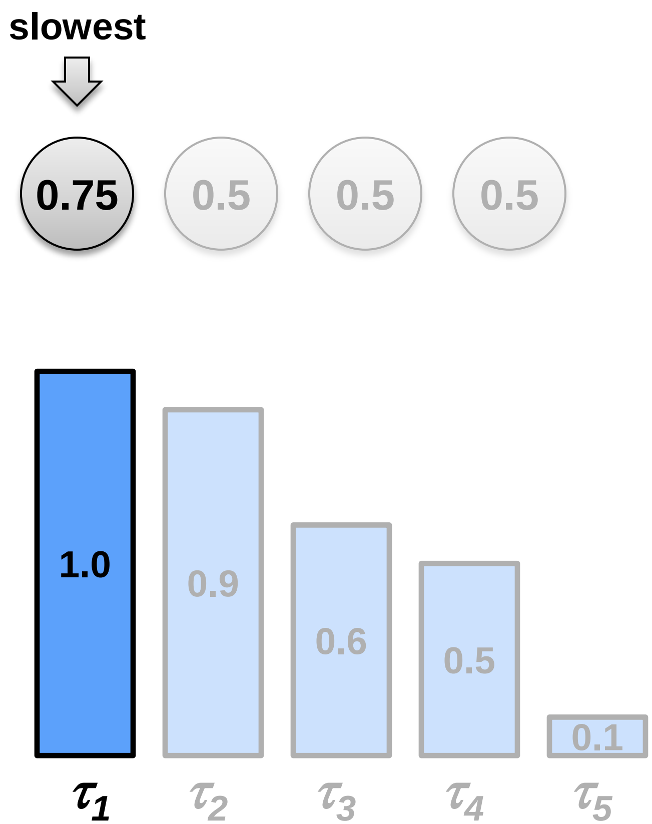


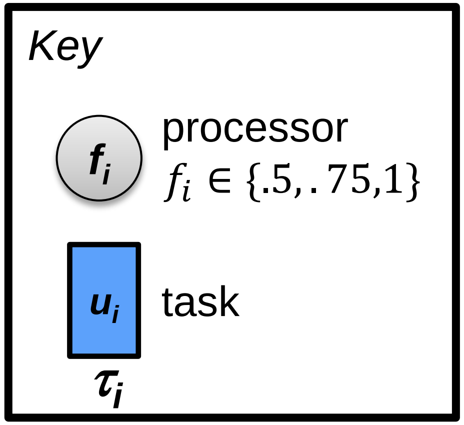
$$\sum_{i=1}^{k} u_i = 1.0 \quad > \quad \sum_{i=1}^{k} f_i = 0.5$$



GMF Example 2

k = 1

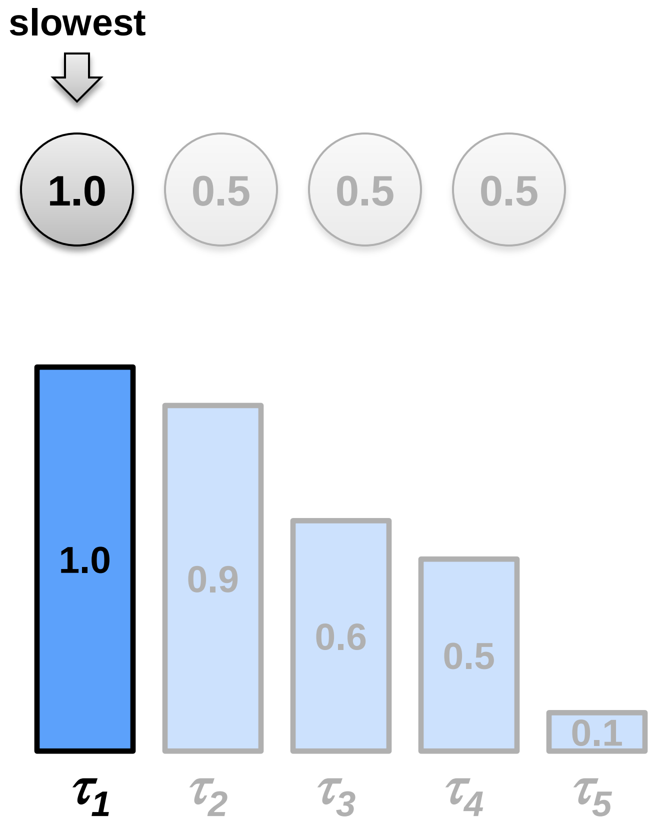




$$\sum_{i=1}^{k} u_i = 1.0 \quad > \quad \sum_{i=1}^{k} f_i = 0.75$$

GMF Example 3

k = 1



Key

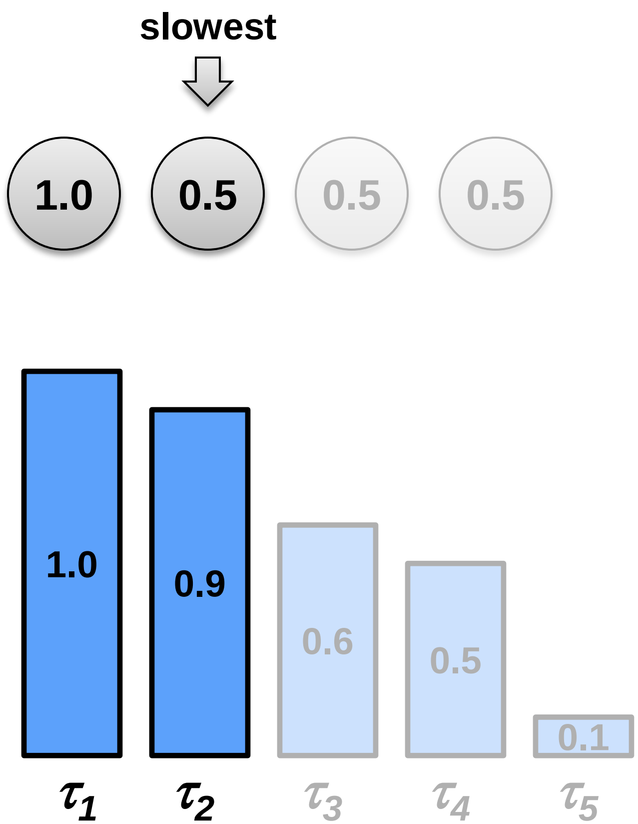
processor $f_i \in \{.5, .75, 1\}$

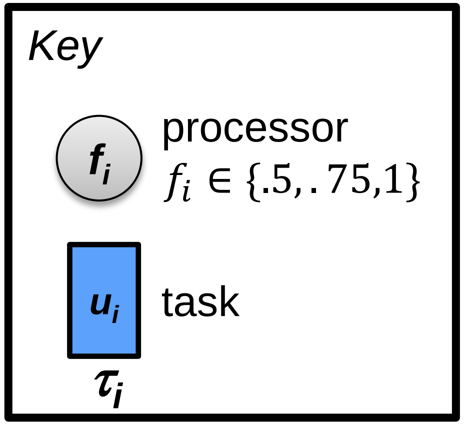
task u_i, τ_i

$$\sum_{i=1}^{k} u_i = 1.0 \quad = \quad \sum_{i=1}^{k} f_i = 1.0$$

GMF Example 4

k = 2

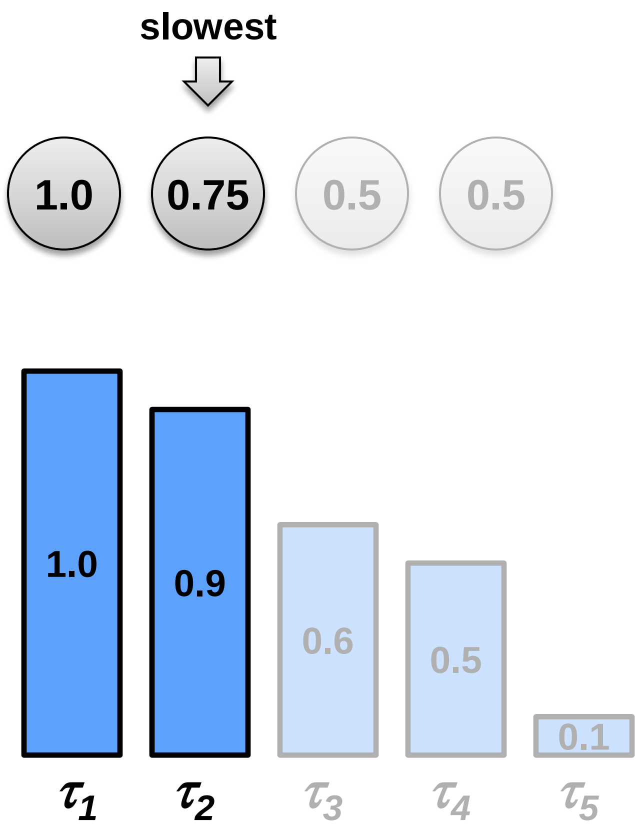


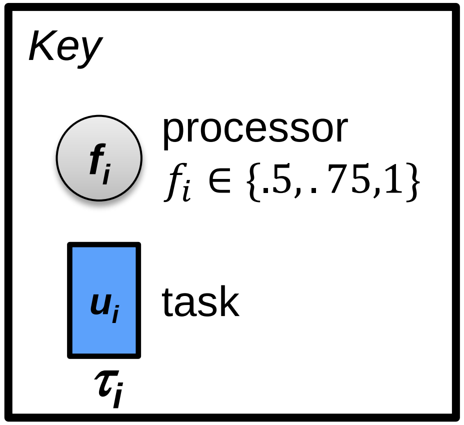


$$\sum_{i=1}^{k} u_i = 1.9 \quad > \quad \sum_{i=1}^{k} f_i = 1.5$$

GMF Example 5

k = 2

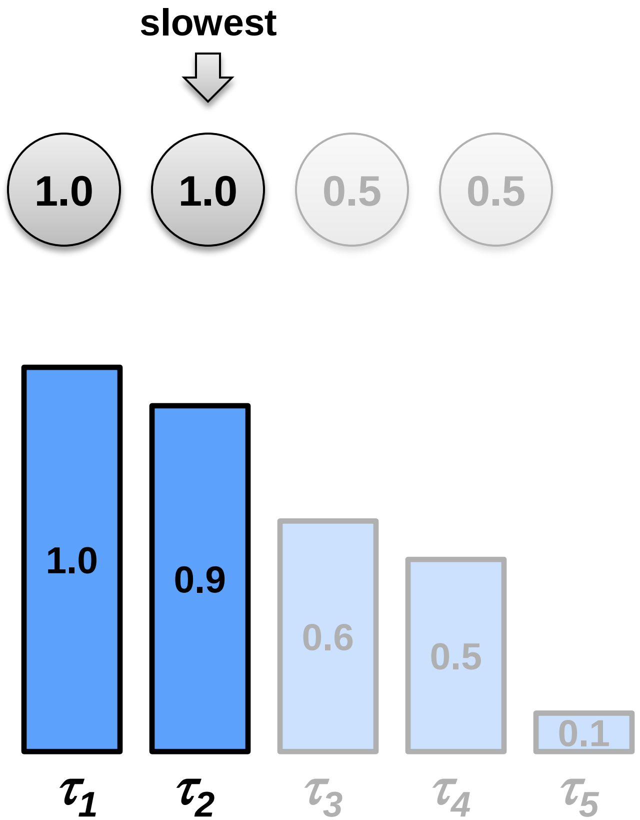




$$\sum_{i=1}^{k} u_i = 1.9 \quad > \quad \sum_{i=1}^{k} f_i = 1.75$$

GMF Example 6

k = 2



Key

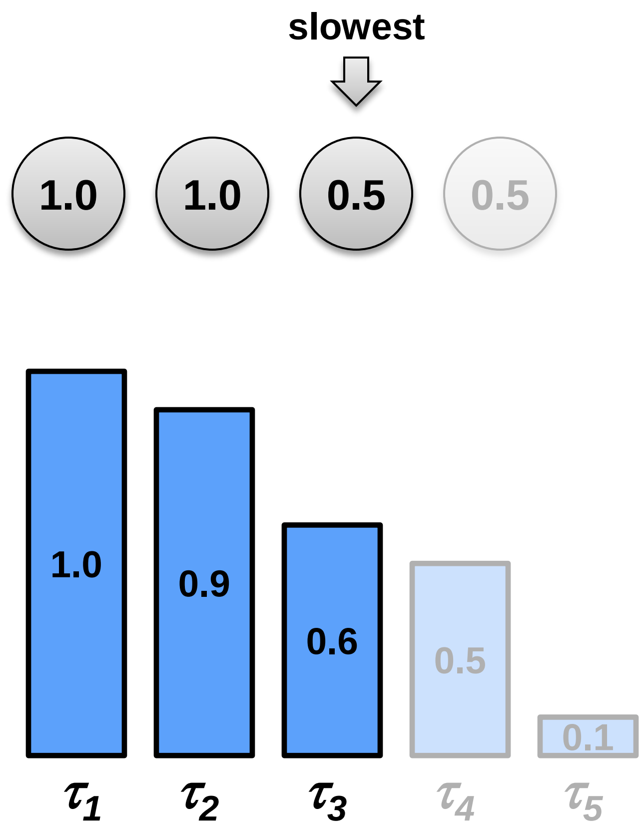
processor $f_i \in \{.5, .75, 1\}$

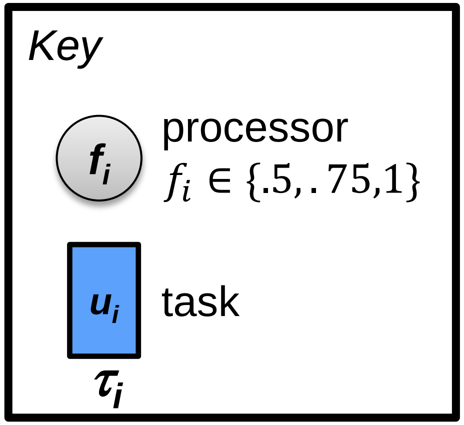
task u_i, τ_i

$$\sum_{i=1}^{k} u_i = 1.9 \quad < \quad \sum_{i=1}^{k} f_i = 2.0$$

GMF Example 6

k = 3

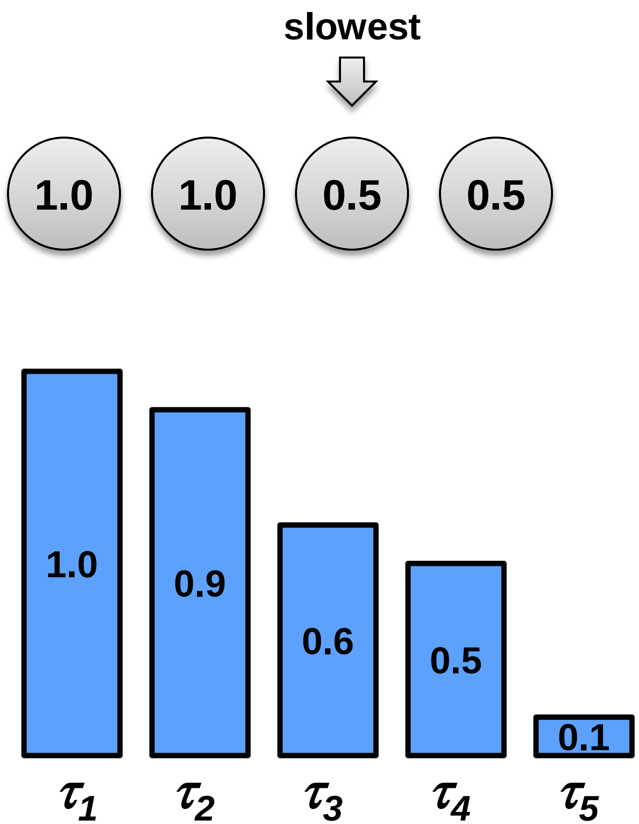


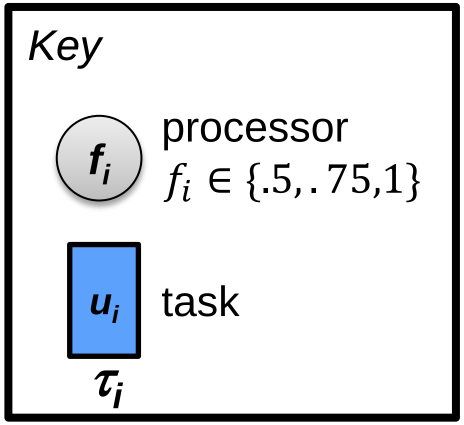


$$\sum_{i=1}^{k} u_i = 2.5 \quad = \quad \sum_{i=1}^{k} f_i = 2.5$$

GMF Example 7

k = 4 = m

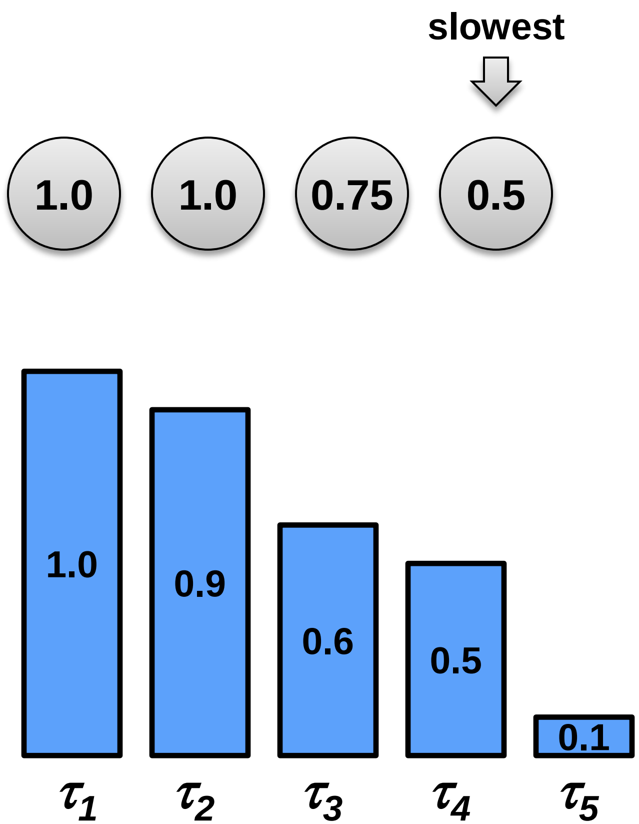


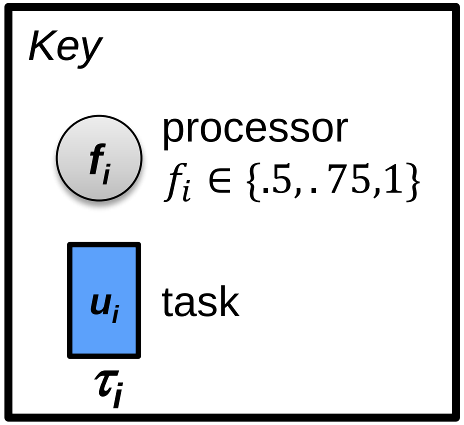


$$\sum_{i=1}^{n} u_i = 3.1 \quad > \quad \sum_{i=1}^{m} f_i = 3.0$$

GMF Example 8

$k = 4 = m$

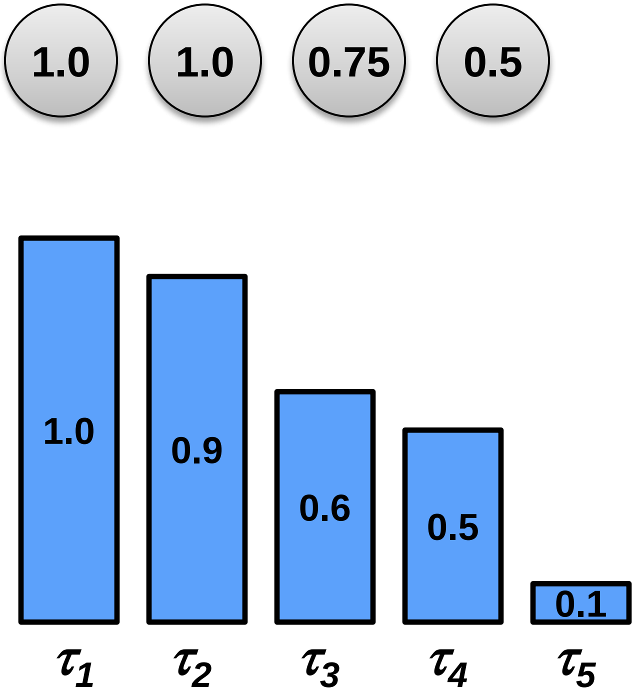




$$\sum_{i=1}^{n} u_i = 3.1 \quad < \quad \sum_{i=1}^{m} f_i = 3.25$$

GMF Example 9

In this case, frequency assignment is the same as in the Exhaustive partition search.



Evaluation

Randomly generated 15,000 tasksets

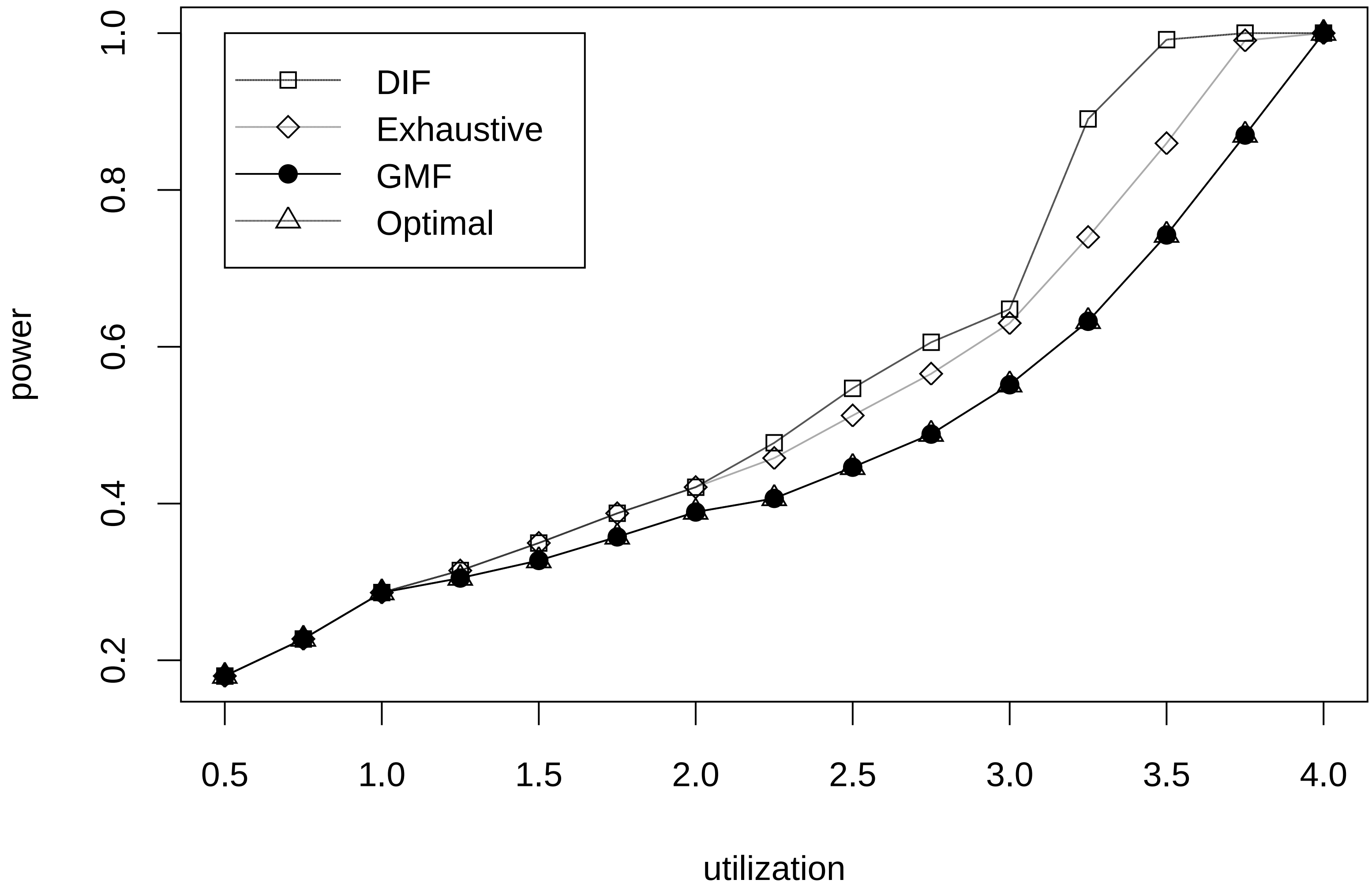
- utilization level ranging from 0.5 to 4 in steps of 0.25
- 1,000 tasksets for each utilization level
- each taskset composed of tasks with random uniform utilization

Used frequencies and voltages of three different quad-core processors

Computed frequency assignment and corresponding power with different multiprocessor VFS algorithms

- Decide Independent Frequency [Funaoka 2008]
- Exhaustive partition/frequency assignment search
- GMF
- Optimal (exhaustive frequency assignment w/o partitions)

Evaluation Results

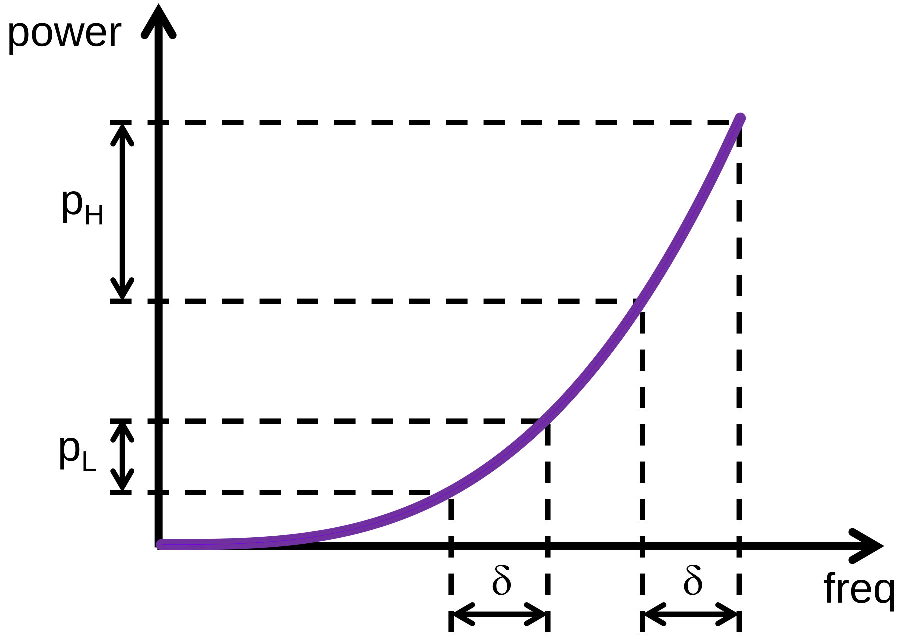


Optimality 1

GMF is optimal when the supported frequency steps are uniform.

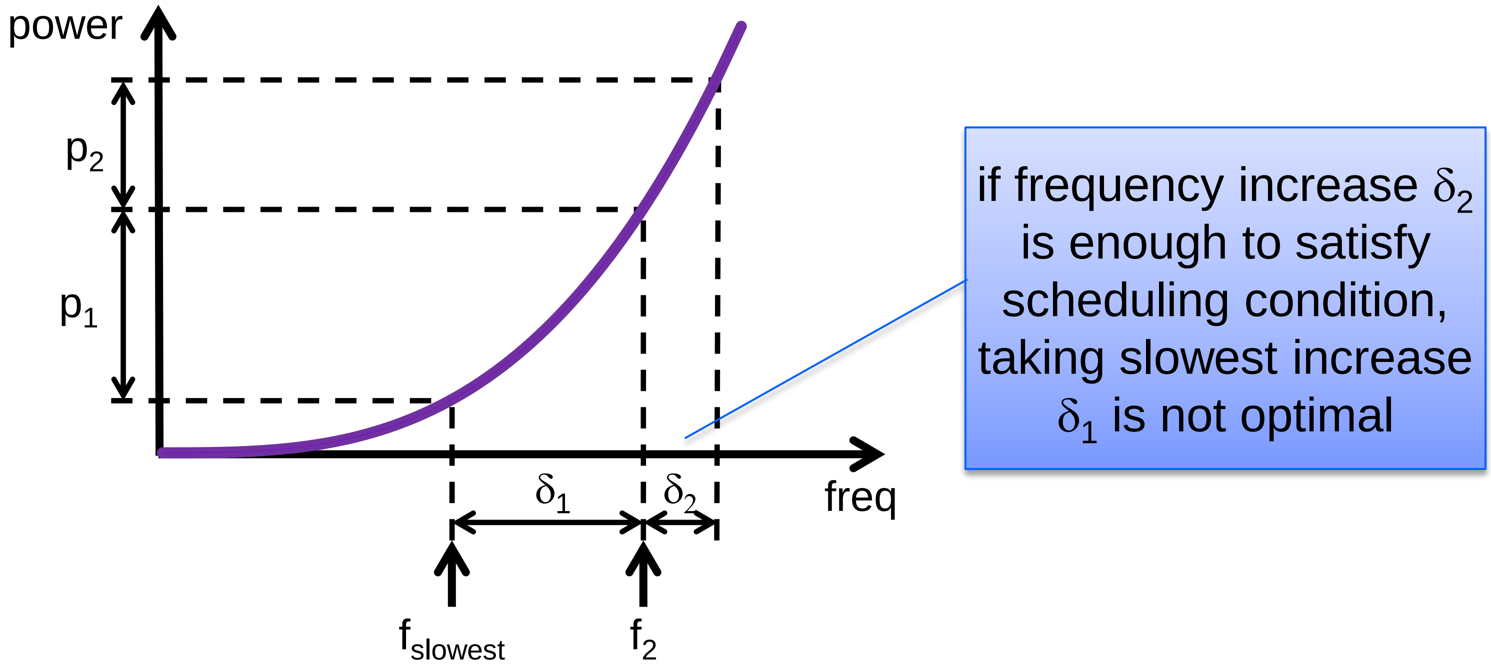
Proof intuition:

- Any frequency step we choose is the same in terms of speed
- Increasing the frequency of the slowest processor requires the smallest power increase
- The optimal frequency assignment for the first *i* conditions bounds from below the optimal assignment for the *i*+1 conditions
- GMF assigns frequencies as even as possible within that bound



Optimality 2

With non-uniform frequency steps, GMF may not optimal



We have observed that for some platforms with non-uniform frequency steps GMF is still optimal

- When the power steps associated frequency steps are non-decreasing

Conclusion

Growing Minimum Frequency (GMF) algorithm computes the optimal frequency assignment to minimize the power consumption of a real-time periodic taskset in a multiprocessor platform.

Evaluation results show up to 30% improvement over previous algorithms.

Avoiding partitioning allows GMF to achieve better power efficiency than optimal partitioned approaches.