Multiprocessor Scheduling III: Semi-Partitioned Scheduling

Prof. Dr. Jian-Jia Chen

LS 12, TU Dortmund

01, July, 2014
Weakness of Partitioned Scheduling

• Restricting a task on a processor reduces the schedulability
• Restricting a task on a processor makes the problem \(NP\)-hard
• Example: Suppose that there are \(M\) processors and \(M+1\) tasks with the same period \(T\) and the (worst-case) execution times of all these \(M+1\) tasks are \(\frac{T}{2} + \epsilon\) with \(\epsilon > 0\)
  • With partitioned scheduling, it is not schedulable
  • The least upper bound (LUB) for partitioned scheduling is no more than 50% of the EDF or RM.
• Within this part, we will focus only for *identical processors* and periodic tasks with *implicit deadlines*. 
Benefits by Allowing Task Migration

- The schedulability can be improved.
- The $NP$-completeness for EDF does no hold any more if the migration has *no overhead*.
- The $NP$-completeness for schedulability test of RM still holds since 1 processor is already $NP$-complete.
- For the above example, we will have 100% LUB if all the tasks have the same period with $D_i = T_i$, and $C_i \leq T_i$ for task $\tau_i$. 

![Diagram showing task scheduling with split and unsplit tasks]
Limits of Task Migration

- Migration indeed requires overhead
  - Migration should be performed only when it is necessary.
  - The number of migrations should be reduced as much as possible.
  - This property is the killing argument for fine-grained task migration schedulers, e.g., p-fair scheduling.
- Migration has to be done carefully so that a job of a task should not be executed simultaneously on more than one processor.
  - The execution of a migrating task should be divided into blocks on different processors such that the blocks do not overlap.
  - This property forces the scheduler to make smarter decisions for task migrations
Articles for This Module

- Björn Andersson, Konstantinos Bletsas: Sporadic Multiprocessor Scheduling with Few Preemptions. ECRTS 2008: 243-252

**Dynamic Priority**


**Static Priority**
Further Readings


Outline

Introduction

Semi-Partitioned EDF Scheduling

Semi-Partitioned Static-Priority Scheduling
Approaches

- Use resource reservation for task migration.
  - Suppose that $T_{\min}$ is the minimum period among all the tasks.
  - By a user-designed parameter $\kappa$, we divide time into slots with length $S = \frac{T_{\min}}{\kappa}$.
  - We can use the first-fit approach by splitting a task into 2 subtasks, in which one is executed on processor $m$ and the other is executed on processor $m+1$.
  - Execution of a split task is only possible in the reserved time window in the time slot.
  - Applying first-fit algorithm, by taking $SEP$ as the upper bound of utilization on a processor.
    - If a task does not fit, split this task into two subtasks and allocate a new processor, one is assigned on the processor under consideration, and the other is assigned on the newly allocated processor.
Reservation for EDF

For each time slot, we will reserve two parts.

\[ S = \frac{T_{\text{min}}}{\kappa} \]

If a task \( \tau_i \) is split, the task can be served only within these two pre-defined time slots with length \( x_i \) and \( y_i \).

A processor can host two split tasks, \( \tau_i \) and \( \tau_j \). \( \tau_i \) is served at the beginning of the time slot, and \( \tau_j \) is served at the end.

The schedule is EDF, but if a split task instance is in the ready queue, it is executed in the reserved time region.
Inflation for Reservation

To meet the utilization request of a split task $\tau_i$ by splitting $lo\_split(\tau_i)$ portion on processor $m$ and $high\_split(\tau_i)$ portion on processor $m + 1$, we should guarantee that

$$\frac{x_i + y_i}{S} \geq lo\_split(\tau_i) + high\_split(\tau_i) = \frac{C_i}{T_i} = U_i.$$

- Assigning only $x_i + y_i = S \cdot U_i$ does not work, since we might just miss the available reserved time slot.
- Let’s inflate $x_i$ and $y_i$ by a constant portion $f$, in which $x_i = S \cdot (f + lo\_split(\tau_i))$ and $y_i = S \cdot (f + high\_split(\tau_i))$.
- To ensure the schedulability, we need

$$\frac{\kappa(x_i + y_i)}{(\kappa + 1)S - (x_i + y_i)} \geq \frac{C_i}{T_i}.$$

- The above inequality comes from the worst case that we miss the reserved time slot at the beginning of a time slot, and have to finish before the reserved time slot at the end of a time slot.
How Much to Inflate?

\[
\frac{\kappa (x_i + y_i)}{\left(\kappa + 1\right) S - (x_i + y_i)} \geq \frac{C_i}{T_i} 
\]

\[
\Rightarrow \frac{T_{\min} (2f + U_i)}{T_{\min} + \frac{T_{\min}}{\kappa} - \frac{T_{\min}}{\kappa} (2f + U_i)} \geq U_i 
\]

\[
\Rightarrow \frac{2f + U_i}{1 + \frac{1}{\kappa} - \frac{1}{\kappa} (2f + U_i)} \geq U_i 
\]

\[
\Rightarrow f \geq \frac{U_i - U_i^2}{2\kappa + 2U_i} 
\]

Therefore, by simple calculus, given \( U_i \), the minimum \( f \) happens when \( f = \frac{2}{\kappa} \left(\kappa + 1\right) - \kappa \).
Algorithm

We can assign all the tasks $\tau_i$ with $U_i > SEP$ on a dedicated processor. So, we only consider tasks with $U_i \leq SEP$.

1: $m \leftarrow 1, U_m \leftarrow 0$
2: for $i = 1$ to $N$, where $N = |T|$ do
3: \hspace{1em} if $\frac{C_i}{T_i} + U_m \leq SEP$ then
4: \hspace{2em} assign task $\tau_i$ on processor $m$;
5: \hspace{2em} $U_m \leftarrow U_m + \frac{C_i}{T_i}$;
6: \hspace{1em} else
7: \hspace{2em} assign task $\tau_i$ on processor $m$ with $lo\_split(\tau_i)$ set to $SEP - U_m$
\hspace{2em} and on processor $m + 1$ with $high\_split(\tau_i)$ set to $\frac{C_i}{T_i} - (SEP - U_m)$;
8: \hspace{1em} $m \leftarrow m + 1$ and $U_m \leftarrow \frac{C_i}{T_i} - (SEP - U_m)$;

• When executing, the reservation to serve $\tau_i$ is to set $x_i$ to $S(f + lo\_split(\tau_i))$ and $y_i$ to $S(f + high\_split(\tau_i))$.
• SEP is set as a constant.
Two Split Tasks on a Processor

For split tasks to be schedulable, the following sufficient conditions have to be satisfied:

- \( \text{lo}\_\text{split}(\tau_i) + f + \text{high}\_\text{split}(\tau_i) + f \leq 1 \) for any split task \( \tau_i \).
- \( \text{lo}\_\text{split}(\tau_j) + f + \text{high}\_\text{split}(\tau_i) + f \leq 1 \) when \( \tau_i \) and \( \tau_j \) are assigned on the same processor.

Therefore, the “magic value” \( \text{SEP} \)

\[
\text{SEP} \leq 1 - 2f \leq 1 - 2(\frac{2}{\sqrt{\kappa(\kappa + 1)}} - \kappa).
\]

However, we still have to guarantee the schedulability of the non-split tasks. It can be shown that the sufficient condition is

\[
\text{SEP} \leq 1 - 4f \leq 1 - 4(\frac{2}{\sqrt{\kappa(\kappa + 1)}} - \kappa).
\]

The proof is omitted here.
Magic Values: $f$

Figure: $f$ versus $\kappa$
Magic Values: $SEP$

Figure: $SEP$ versus $\kappa$
By taking $SEP$ as $1 - 4\left(\sqrt{\kappa(\kappa + 1)} - \kappa\right)$ and $f = \sqrt{\kappa(\kappa + 1)} - \kappa$, the above algorithm guarantees to derive feasible schedule if $\sum_{\tau_i \in T} \frac{C_i}{T_i} \leq M' \cdot SEP$ and $\frac{C_i}{T_i} \leq 1$ for all tasks $\tau_i$. 
Outline

Introduction

Semi-Partitioned EDF Scheduling

Semi-Partitioned Static-Priority Scheduling
Basic Idea

1. Use Liu and Layland’s bound for rate-monotonic scheduling
   - A task set of $N$ implicit-deadline tasks is schedulable (under RM) on one processor if the total utilization is no more than $U_{lub}(RM, N) = N(2^{\frac{1}{N}} - 1) \geq 0.693$.

2. We can start to consider the tasks with the non-increasing order of periods

3. To reduce the utilization, we again pick the processor with the minimum task utilization so far when considering task $\tau_i$

4. If a task cannot fit into the picked processor, we will have to split it into multiple (two or even more) parts.

5. Since we consider periods in an non-increasing order, when $\tau_i$ is split, it has higher priority than other tasks that have been considered.

6. Therefore, if $\tau_i$ is split and assigned to a processor $m$ and the utilization on processor $m$ after assigning $\tau_i$ is at most $U_{lub}(RM, N)$, then $\tau_i$ is so far schedulable.
Terminology

- Non-split task: a task that is executed only on one processor.
- Split task: a task $\tau_i$ that is executed on $k_i$ processors, where $k_i \geq 2$.
  - There are $k_i$ subtasks, denoted by $\tau_i^1, \tau_i^2, \ldots, \tau_i^{k_i}$, in which none of them will run at the same time.
  - The $k_i$ subtasks have to be synchronized.
  - Subtask $\tau_i^j$ is with computation time requirement $C_i^j$, relative deadline $\Delta_i^j$, and period $T_i$.
  - Subtask $\tau_i^{k_i}$ is called the **tail subtask** of task $\tau_i$.
  - Subtask $\tau_i^1, \tau_i^2, \ldots, \tau_i^{k_i-1}$ are called **body subtasks** of task $\tau_i$. 

![Diagram of subtasks and processors](image)
Algorithm: Largest Period First (LPF)

Input: $T, M$;
1: re-index (sort) tasks such that $T_i \geq T_j$ for $i > j$;
2: initialize utilization $U_m$ on processor $m$ as 0;
3: initialize $k_i$ of task $\tau_i$ as 1 and $R_i^1$ as 0;
4: for $i = N$ to 1, where $N = |T|$ do
5: find $m^*$ with the minimum utilization, i.e., $U_{m^*} = \min_m U_m$;
6: return “assignment fails” if $U_{m^*} \geq U_{lub}(RM, N)$;
7: if $U_{m^*} + \frac{C_i^{k_i}}{T_i} \leq U_{lub}(RM, N)$ then
8: assign subtask $\tau_i^{k_i}$ onto processor $m^*$, where $U_{m^*} \leftarrow U_{m^*} + \frac{C_i^{k_i}}{T_i}$;
9: else
10: split task $\tau_i$ further with $C_i^{k_i+1} \leftarrow C_i^{k_i} - (U_{lub}(RM, N) - U_{m^*})T_i$
and $C_i^{k_i} \leftarrow (U_{lub}(RM, N) - U_{m^*})T_i$;
11: assign subtask $\tau_i^{k_i}$ onto processor $m^*$, where
$U_{m^*} \leftarrow U_{lub}(RM, N)$;
12: $R_i^{k_i+1} \leftarrow C_i - \sum_{j=1}^{k_i} C_j$;
13: $k_i \leftarrow k_i + 1$ and goto step 5;
14: execute with rate-monotonic scheduling, in which subtask $\tau_i^j$ is with
An Example for LPF

Let’s ignore \( R_i \) first. \( U_{\text{lub}}(RM, 8) \approx 0.724. \)
An Example for LPF

Let's ignore $R_i^j$ first. $U_{lub}(RM, 8) \approx 0.724$. 

Prof. Dr. Jian-Jia Chen (LS 12, TU Dortmund)
An Example for LPF

Let’s ignore $R^j_i$ first. $U_{lub}(RM, 8) \approx 0.724$. 

![Diagram showing τ8, τ7, τ6, τ5, τ4, τ3, τ2, τ1 with corresponding values: τ8 = 0.452, τ7 = 0.4, τ6 = 0.37, τ5 = 0.3, τ4 = 0.2, τ3 = 0.2, τ2 = 0.15, τ1 = 0.1.](image)
An Example for LPF

Let's ignore $R_i^j$ first. $U_{lub}(RM, 8) \approx 0.724$. 

Prof. Dr. Jian-Jia Chen (LS 12, TU Dortmund) 21 / 35
An Example for LPF

Let's ignore $R_i^j$ first. $U_{lub}(RM, 8) \approx 0.724$. 
An Example for LPF

Let’s ignore $R_i^j$ first. $U_{lb}(RM, 8) \approx 0.724$. 

![Diagram](image-url)
An Example for LPF

Let’s ignore $R^j_i$ first. $U_{lub}(RM, 8) \approx 0.724$. 

Prof. Dr. Jian-Jia Chen  (LS 12, TU Dortmund) 21 / 35
An Example for LPF

Let's ignore $R^j_i$ first. $U_{lub}(RM, 8) \approx 0.724$. 

$.452$  $0.4$  $.37$  $0.3$  $.2$  $.2$  $.15$  $.1$

$\tau_8$  $\tau_7$  $\tau_6$  $\tau_5$  $\tau_4$  $\tau_3$  $\tau_2$  $\tau_1$

$P_1$  $P_2$  $P_3$
An Example for LPF

Let's ignore $R_i^j$ first. $U_{lub}(RM, 8) \approx 0.724.$

$(0.678, 0.724, 0.67)$
An Example for LPF

Let's ignore $R_i^j$ first. $U_{lub}(RM, 8) \approx 0.724$. 

$U = 0.724$
Body Subtasks

Let $m(\tau^j_i)$ be the processor that subtask $\tau^j_i$ is assigned onto;

**Theorem**

If $j < k_i$, i.e., $\tau^j_i$ is a body subtask, $\tau^j_i$ is with the highest priority on processor $m(\tau^j_i)$.

**Proof**

- This simply comes from the algorithm, which considers tasks from the lowest-priority to the highest priority
- When a body subtask is assigned, there is no more (sub)task $\tau^\ell_k$ with $k < i$ going to be assigned on processor $m(\tau^j_i)$ anymore.
How does Scheduling Work?

Let's put $R^j_i$ back.

![Diagram showing scheduling process with release times and ready times for tasks $\tau_i$.](image-url)
Schedulability on Good Partitions

Suppose that $T_m$ is the set of (sub)tasks that are assigned on processor $m$.

**Theorem**

If there are only body subtasks or non-split tasks in $T_m$, then

- the non-split tasks can meet their deadlines and
- the body subtask $\tau^j_i$ is with response time $C^j_i$ with deferred release $R^j_i$.

It should be clear now.
Trouble Maker: Tail Subtasks

- A tail subtask does not have the highest priority...
- It should also be now clear that the offset of a tail subtask $\tau_{ki}$ is equal to $\sum_{j=1}^{k_i-1} C_{ij}$.
- Therefore, to meet the requirement with non-overlapping execution of task $\tau_i$, we have to finish $\tau_{ki}$ within $T_i - \sum_{j=1}^{k_i-1} C_{ij} = T_i - (C_i - C_{ki})$.
- For simplicity, we use $\Delta_i$ to denote $T_i - (C_i - C_{ki})$;
- Suppose that $Y_i$ is the utilization of the tasks that are assigned on the processor $m(\tau_{ki})$ for executing $\tau_{ki}$ and with higher priority than $\tau_{ki}$.
- We would first like to show that $\tau_{ki}$ can finish within $\Delta_i$ if
  $$\frac{Y_i T_i + C_{ki}}{\Delta_i} \leq U_{lub}(RM, N).$$
Proof for Tail Subtasks

- Again, since rate-monotonic is applied, we only have to consider tasks with periods shorter than task $\tau_i$.
- For notational brevity, let’s assume $\tau_{ki}^i$ is on processor $m$;
- Let’s distinguish two types of tasks on $m$ (by the relationship with $\Delta_i$):
  $$T_m^1 = \{ \tau \in T_m, j < i, T_j \geq \Delta_i \}$$
  $$T_m^2 = \{ \tau \in T_m, j < i, T_j < \Delta_i \}$$
- By definition, (note that we do not distinguish between split or non-split tasks by defining $c_j$ as the execution time of task $\tau_j$ or subtask $\tau^l_j$ in $T_m$):
  $$U_{lub}(RM, N) \geq \frac{Y_i T_i + C_{ki}^i}{\Delta_i} = \left( \sum_{\tau_j \in T_m^1} \frac{c_j}{T_j} + \sum_{\tau_j \in T_m^2} \frac{c_j}{T_j} \right) \frac{T_i}{\Delta_i} + C_{ki}^i$$
  $$= \sum_{\tau_j \in T_m^1} \frac{c_j T_i}{\Delta_i} + \sum_{\tau_j \in T_m^2} \frac{c_j T_i}{\Delta_i} + C_{ki}^i \geq \sum_{\tau_j \in T_m^1} \frac{c_j}{\Delta_i} + \sum_{\tau_j \in T_m^2} \frac{c_j}{T_j} + \frac{C_{ki}^i}{\Delta_i}.$$  
- A more difficult task set (by changing $T_j$ to $\Delta_i$ for $\tau_j \in T_m^1$) finishes $\tau_{ki}^i$ within $\Delta_i$, so is the original task set.
Issues of the Tail Subtask

- If we are sure that \( \frac{Y_i T_i + C_i^{k_j}}{\Delta_i} \leq U_{lub}(RM, N) \), we will be fine.
- However, the above inequality only holds for some cases when the utilization of \( \tau_i \) is small enough.
- So, the next question is: What is the maximum \( U_i \) of a split task \( \tau_i \), that enforces \( \frac{Y_i T_i + C_i^{k_j}}{\Delta_i} \leq U_{lub}(RM, N) \)?
- Let \( X_i \) be the total utilization of the tasks on processor \( m \) with lower-priority than \( \tau_i^{k_j} \).
- By definition, we know that

\[
X_i + \frac{C_i^{k_j}}{T_i} + Y_i \leq U_{lub}(RM, N)
\]
Issues of the Tail Subtask

As $\tau_i$ has $k_i$ subtasks, there are $k_i - 1$ body subtasks assigned on other $k_i - 1$ processors. Therefore,

$$U_{lb}(RM, N) - Y_i \geq X_i + \frac{C_{k_i}^i}{T_i} \geq 1 \quad U_{lb}(RM, N) - \frac{\sum_{j=1}^{k_i-1} C_j^i}{(k_i - 1) T_i} + \frac{C_{k_i}^i}{T_i}$$

$$= U_{lb}(RM, N) - \frac{C_i - C_{k_i}^i}{(k_i - 1) T_i} + \frac{C_{k_i}^i}{T_i} \geq 2 \quad U_{lb}(RM, N) - (\frac{C_i}{T_i} - 2 \frac{C_{k_i}^i}{T_i}).$$

$\geq 1$ is because there are $k_i - 1$ processors with utilization lower than or equal to $X_i$ before assigning $\tau_{i, k_i}^i$. $\geq 2$ is because $k_i \geq 2$. Hence,

$$Y_i \leq (\frac{C_i}{T_i} - 2 \frac{C_{k_i}^i}{T_i}).$$

Since $\tau_{i, k_i}^i$ can finish within $\Delta_i$ if

$$\frac{Y_i T_i + C_{k_i}^i}{\Delta_i} \leq U_{lb}(RM, N),$$

we have that if

$$\frac{Y_i T_i + C_{k_i}^i}{\Delta_i} \leq \frac{C_i - C_{k_i}^i}{T_i - (C_i - C_{k_i}^i)} = \frac{T_i}{T_i - (C_i - C_{k_i}^i)} - 1 \leq U_{lb}(RM, N),$$

$\tau_{i, k_i}^i$ can finish within $\Delta_i$. 
Light and Heavy Tasks

- If task $\tau_i$ is with utilization lower than or equal to $\frac{U_{lub}(RM,N)}{1+U_{lub}(RM,N)}$, $\tau_i$ is a light task.

- If task $\tau_i$ is with utilization higher than $\frac{U_{lub}(RM,N)}{1+U_{lub}(RM,N)}$, $\tau_i$ is a heavy task.

Therefore, if $\tau_i$ is a light task,

$$\frac{T_i}{T_i - (C_i - C_i^{k_i})} - 1 \leq \frac{C_i}{T_i} \leq U_{lub}(RM, N).$$
First Part

**Theorem**

If all the (split) tasks are light tasks, e.g., $U_i \leq \frac{U_{lub}(RM,N)}{1+U_{lub}(RM,N)} \forall \tau_i \in T$ (or for every split task $\tau_i$), LPT guarantees the feasibility of the derived schedule when $\sum_{\tau_i \in T} U_i \leq U_{lub}(RM, N)$ and $U_i \leq 1$ for all the tasks $\tau_i$ on $M$ homogeneous processors.
Assign Heavy Tasks First (HT-LPT)

Key idea: Pre-assign a HEAVY task such that it is not split.

1: pre-assign heavy tasks under some conditions;
2: assign light tasks on processors without pre-assigned tasks as much as possible;
3: assign remaining light tasks to the processors with heavy tasks with a “specific order”;
Assign Heavy Tasks First (HT-LPT)

Key idea: Pre-assign a HEAVY task such that it is not split.

1: $m \leftarrow 1$
2: for $i = 1$ to $N$, where $N = |T|$ do
3: if $\tau_i$ is not a heavy task then
4: continue;
5: if $\sum_{j=i+1}^{N} \frac{C_j}{T_j} \leq (M - m) U_{lub}(RM, N)$ then
6: pre-assign task $\tau_i$ on processor $m$;
7: $m \leftarrow m + 1$;
8: let $T^{pre}$ be the set of tasks that are pre-assigned on $m - 1$ processors after the above loop, and $M^\dagger \leftarrow m - 1$;
9: run algorithm LPF for deciding $T \setminus T^{pre}$ on processor $M^\dagger + 1$ to $M$;
10: while there exists task $\tau_i$ in $T \setminus T^{pre}$ that cannot be assigned on any processor without pre-assigned tasks do
11: repeatedly assign or split task $\tau_i$ as much as possible on the largest-index processor with utilization less than $U_{lub}(RM, N)$ among the first $M^\dagger$ processors;
Example of HT-LPT

\[ \tau_8 = 0.3 \]
\[ \tau_7 = 0.6 \]
\[ \tau_6 = 0.3 \]
\[ \tau_5 = 0.45 \]
\[ \tau_4 = 0.6 \]
\[ \tau_3 = 0.24 \]
\[ \tau_2 = 0.2 \]
\[ \tau_1 = 0.2 \]

\[ P_1 \]
\[ P_2 \]
\[ P_3 \]
\[ P_4 \]
Example of HT-LPT

\[ \tau_8, \tau_7, \tau_6, \tau_5, \tau_4, \tau_3, \tau_2, \tau_1 \]

- \( \tau_8 = 0.3 \)
- \( \tau_7 = 0.6 \)
- \( \tau_6 = 0.3 \)
- \( \tau_5 = 0.45 \)
- \( \tau_4 = 0.6 \)
- \( \tau_3 = 0.24 \)
- \( \tau_2 = 0.2 \)
- \( \tau_1 = 0.2 \)

Light

- \( P_1 \)
- \( P_2 \)
- \( P_3 \)
- \( P_4 \)
Example of HT-LPT

$\tau_8 = 0.3$

$\tau_7 = 0.6$

$\tau_6 = 0.3$

$\tau_5 = 0.45$

$\tau_4 = 0.6$

$\tau_3 = 0.24$

$\tau_2 = 0.2$

$\tau_1 = 0.2$

$P_1$

$P_2$

$P_3$

$P_4$
Example of HT-LPT

\[ \tau_8 = 0.3 \quad \tau_7 = 0.6 \quad \tau_6 = 0.3 \quad \tau_5 = 0.45 \quad \tau_4 = 0.6 \quad \tau_3 = 0.24 \quad \tau_2 = 0.2 \quad \tau_1 = 0.2 \]

\[ P_1 \quad P_2 \quad P_3 \quad P_4 \]
Example of HT-LPT
Example of HT-LPT

\[ \tau_8 \] \[ \tau_7 \] \[ \tau_6 \] \[ \tau_5 \] \[ \tau_4 \] \[ \tau_3 \] \[ \tau_2 \] \[ \tau_1 \]

0.3 0.6 0.3 0.45 0.6 0.24 0.2 0.2

heavy and pre-assign

\[ \tau_4 \] \[ \tau_5 \] \[ \tau_3 \] \[ \tau_2 \] \[ \tau_1 \]

\[ P_1 \] \[ P_2 \] \[ P_3 \] \[ P_4 \]
Example of HT-LPT
Example of HT-LPT

Heavy and pre-assign

$\tau_8$ 0.3
$\tau_7$ 0.6
$\tau_6$ 0.3
$\tau_5$ 0.45
$\tau_4$ 0.6
$\tau_3$ 0.24
$\tau_2$ 0.2
$\tau_1$ 0.2

$P_1$ $\tau_4$
$P_2$ $\tau_5$
$P_3$ $\tau_7$
$P_4$
Example of HT-LPT

\[ \tau_8 \] 0.3

\[ \tau_7 \] 0.6

\[ \tau_6 \] 0.3

\[ \tau_5 \] 0.45

\[ \tau_4 \] 0.6

\[ \tau_3 \] 0.24

\[ \tau_2 \] 0.2

\[ \tau_1 \] 0.2

light

\[ \tau_4 \] \( P_1 \)

\[ \tau_5 \] \( P_2 \)

\[ \tau_7 \] \( P_3 \)

\[ \tau_1 \] \( P_4 \)
Example of HT-LPT

\[\tau_8 \quad \tau_7 \quad \tau_6 \quad \tau_5 \quad \tau_4 \quad \tau_3 \quad \tau_2 \quad \tau_1\]

\[
\begin{array}{cccccccc}
0.3 & 0.6 & 0.3 & 0.45 & 0.6 & 0.24 & 0.2 & 0.2 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
\tau_4 & \tau_5 & \tau_7 & \tau_3 & \tau_6 & \tau_8 \\
P_1 & P_2 & P_3 & P_4 \\
\end{array}
\]
Example of HT-LPT

\[ \tau_8 = 0.3 \]
\[ \tau_7 = 0.6 \]
\[ \tau_6 = 0.3 \]
\[ \tau_5 = 0.45 \]
\[ \tau_4 = 0.6 \]
\[ \tau_3 = 0.24 \]
\[ \tau_2 = 0.2 \]
\[ \tau_1 = 0.2 \]

\[ P_1 \]
\[ P_2 \]
\[ P_3 \]
\[ P_4 \]
Example of HT-LPT
Priority of Heavy Tasks

Let $\tau_i$ be a heavy task, and there are $\eta$ pre-assigned tasks with higher priority than $\tau_i$. Then we know

- If $\tau_i$ is a pre-assigned task, it satisfies

$$\sum_{j > i} \frac{C_i}{T_i} \leq (M - \eta + 1) \times U_{lub}(RM, N).$$

- If $\tau_i$ is not a pre-assigned task, it satisfies

$$\sum_{j > i} \frac{C_i}{T_i} > (M - \eta + 1) \times U_{lub}(RM, N).$$

Theorem

A pre-assigned task has the lowest-priority on its host processor.
Property of HT-LPT (Proofs are omitted)

**Theorem**

HT-LPT guarantees the feasibility of the derived schedule when
\[ \sum_{\tau_i \in T} \frac{U_i}{M} \leq U_{lub}(RM, N) \] and \( U_i \leq 1 \) for all the tasks \( \tau_i \) on \( M \) homogeneous processors.