An Optimal Semi-Partitioned Scheduler for Uniform Heterogeneous Multiprocessors

Kecheng Yang
James H. Anderson
Dept. of Computer Science
UNC-Chapel Hill
An Optimal Semi-Partitioned Scheduler for Uniform Heterogeneous Multiprocessors
An Optimal Semi-Partitioned Scheduler for Uniform Heterogeneous Multiprocessors
Uniform Platforms

- A uniform platform of $m$ processors with speeds: $s_1 \geq s_2 \geq ... \geq s_m$.
- $n$ implicit-deadline sporadic tasks.
- An implicit-deadline sporadic task $\tau_i = (C_i, T_i)$.
  - $T_i$ is the period, and relative deadline $D_i = T_i$.
  - $C_i$ is the worst-case execution requirement.
- Utilization $u_i = C_i/T_i$.

Feasibility Condition:

$$\sum_{\text{all}} u_i \leq \sum_{\text{all}} s_i \quad \text{and} \quad \sum_{\text{k largest}} u_i \leq \sum_{\text{k largest}} s_i$$

for $k=1,2,...,m-1$
An Optimal Semi-Partitioned Scheduler for Uniform Heterogeneous Multiprocessors
Semi-Partitioned Schedulers

- Semi-partitioned scheduling extends partitioned scheduling by allowing some tasks to migrate when necessary.

- Under semi-partitioned scheduling, each task is allocated a non-zero share on certain processors.
  - Each fixed task has a non-zero share on only one processor.
  - Each migrating task has non-zero shares on multiple processors.

- Assignment Phase: assigning the shares.
- Execution Phase: constructing the schedule to fit the shares.
Assignment Phase

• On identical multiprocessors, an assignment has to observe the following two criteria:

  • the total allocated share on each processor does not exceed the processor’s capacity;
    (each processor is not overutilized)

  • the total allocated share of a task matches the task’s utilization.
    (each task will receive enough processor supply)
Assignment Phase

• On identical multiprocessors, assigning a task to be fixed on a processor that has enough capacity will not make the assignment infeasible.

\[
\tau_3 = (2,3) \\
\tau_1 = \tau_2 = \tau_3 = (2,3) \\
s_1 = s_2 = 1 \\
\]
Assignment Phase

- However, on uniform platforms, assigning a task to be fixed on a processor with enough capacity to accommodate it may result in the remaining system being infeasible.

\[ \tau_2 = (2,1) \]

\[ s_1 = 3 \]
\[ s_2 = 1 \]

\[ \tau_1 = \tau_2 = (2,1) \]

Task executing in parallel is not allowed!
Assignment Phase

- This system is actually feasible.
Feasible Assignment

• How do we determine if assigning a task as fixed will result in the remaining system being infeasible?

• $z_i$: residual capacity

• A **sufficient** condition for the feasibility of the remaining system:

$$
\sum_{\text{all unassigned tasks}} u_i \leq \sum_{\text{all}} z_i \quad \text{and}
$$

$$
\sum_{\text{k largest unassigned tasks}} u_i \leq \sum_{\text{k largest}} z_i
$$

for $k=1,2,...,m-1$
Feasible Assignment

• Our algorithm:
  • Consider tasks from lightest to heaviest by utilization.
  • Use the Best-Fit bin-packing heuristic to assign the currently considered task as fixed if (1) and (2) will be satisfied after the assignment.

• If the initial system is feasible, then it is guaranteed that:
  • At least the first \((n-m)\) lightest tasks can be fixed.
  • The remaining at most \(m\) heaviest tasks and the remaining platform will satisfy (1) and (2).

  \(\text{i.e, it is guaranteed that at most} \ m \ \text{tasks need to migrate.}\)

\[
\sum_{\text{all unassigned tasks}} u_i \leq \sum_{\text{all } z_i} z_i \quad (1)
\]

\[
\sum_{\text{k largest unassigned tasks}} u_i \leq \sum_{\text{k largest } z_i} z_i \quad (2)
\]
Scheduling frame by frame. A frame is a time interval of a constant length of $F$.

The processor allocation in each frame is exactly identical.

Based on (1) and (2) and leveraging a prior algorithm, called Level Algorithm*, that was designed for minimizing makespan of independent one-shot jobs, we can derive an allocation table for a frame.

- Applying the Level Algorithm to a set of jobs with execution requirements of $\{u_1 \cdot F, u_2 \cdot F, \ldots, u_m \cdot F\}$ on a uniform platform $\{s'_1 = z_1, s'_2 = z_2, \ldots, s'_m = z_m\}$, where $m'$ is the number of migrating tasks.

- The Level Algorithm guarantees a makespan of at most $F$.

Example

\[ z_i : \text{residual capacity} \]

\[
\begin{align*}
\sum_{\text{all}} u_i & \leq \sum_{\text{all}} z_i \quad (1) \\
\sum_{\text{k largest}} u_i & \leq \sum_{\text{k largest}} z_i \quad (2)
\end{align*}
\]

$z_i$: residual capacity
Example

\[ z_1' = 8 \]
\[ z_2' = 2 \]
\[ z_3' = 1 \]

\[ s_1: z_1 = 8 \]
\[ s_2: z_2 = 2 \]
\[ s_3: z_3 = 1 \]

\[ J_1: u_1 \cdot F = 5F \]
\[ J_2: u_2 \cdot F = 3F \]
\[ J_3: u_3 \cdot F = 3F \]
Within each frame of length $F$, the processor supply guaranteed to a migrating task $\tau_i$ is $u_i \cdot F$.

Within each frame of length $F$, the processor supply guaranteed to the set of fixed tasks on processor $i$ is $\sigma_i \cdot F$.

Fixed tasks are prioritized against each other by EDF.

\[ \sigma_i = \sum u_k \]

$\tau_k$ is a fixed task on processor $i$
Processor-Supply Properties

Property 1. Within any time interval of length $F$, the processor supply guaranteed to a migrating task $\tau_i$ is $u_i \cdot F$.

Property 2. Within any time interval of length $F$, the processor supply guaranteed to the set of fixed tasks on processor $i$ is $\sigma_i \cdot F$. 

[Diagram showing two frames, frame1 and frame2, with identical durations $F$ and interval $2F$.]
An **Optimal** Semi-Partitioned Scheduler for **Uniform** Heterogeneous Multiprocessors
Optimal Schedulability

• Hard Real-Time (HRT) Schedulable: all deadlines are met.
  • HRT-Optimality: for any system that can be guaranteed all deadlines met, a HRT-optimal scheduler will guarantee so.

• Soft Real-Time (SRT) Schedulable: deadlines may be missed; however, the deadline tardiness is bounded.
  • SRT-Optimality: for any system that can be guaranteed all tasks have bounded deadline tardiness, a SRT-optimal scheduler will guarantee so.
    ➢ Bounded deadline tardiness implies bounded response times.
Optimality

Theorem 5. (HRT Optimality) If the frame size $F$ divides the periods of all tasks, then all deadlines will be met.

Theorem 6. (SRT Optimality) Given any frame size $F > 0$, no job will have tardiness exceeding $F$.

For a certain system, the allocation table for a frame has the same layout, regardless the frame size $F$. Therefore, the number of migrations and Level-Algorithm-related preemptions* per frame is fixed when the frame size $F$ varying.

That is, over time, the larger the frame size, the lower the run-time overheads.

This algorithm can make tradeoffs between timeliness and overheads. That is why we call this algorithm EDF-tu (for tunable, uniform).

*I.e., preemptions except those among fixed tasks by EDF.
FAQs

• Why we need to do a semi-partition? Can’t we just directly apply the Level Algorithm to all tasks and compute the allocation table?

That is pretty much what Funk et. al.* considered to establish the feasibility condition on uniform platforms.

The complexity of the schedule for a frame is closely related to the number of tasks that we apply the Level Algorithm to. We only apply the Level Algorithm to the at most \( m \) migrating tasks, instead of totally \( n \) tasks. This improves the overheads significantly.

\[
\sum_{\text{all unassigned tasks}} u_i \leq \sum_{\text{all tasks}} z_i \quad (1)
\]

\[
\sum_{\text{k largest unassigned tasks}} u_i \leq \sum_{\text{k largest}} z_i \quad (2)
\]

FAQs

• While EDF-tu guarantees at most $m$ migrating tasks, can we guarantee fewer migrating tasks?

  ♦ I.e., does there exist an optimal semi-partitioned scheduler that guarantees that there are at most $k$ ($k < m$) migrating tasks?

  ♠ No. We show this by giving a counterexample that there exist a feasible system where $m$ tasks have to migrate.
m tasks have to migrate

- Processors: \( s_1 = 1 + m \cdot \epsilon, \quad s_2 = s_3 = \ldots = s_m = 1 \).
- Tasks: \( \tau_1 = \tau_2 = \ldots = \tau_m = (1 + \epsilon, 1) \).

Feasibility Condition:

\[
\sum_{\text{all}} u_i \leq \sum_{\text{k largest}} s_i \\
\sum_{\text{k largest}} u_i \leq \sum_{\text{k largest}} s_i
\]
Thank you!

Questions?
Worst-Fit
Assign tasks from heavy ones?

• While EDF-tu requires heavier tasks migrating, can we require lighter ones instead?
  • Actually, no, provided the HRT- or SRT- optimality is still required.
  • There is a feasible system of n tasks, where the m heaviest ones have to migrate; otherwise, the system will become infeasible.
    • Processors: \( s_1 = 1 + (m+1) \cdot \varepsilon, \quad s_2 = s_3 = \ldots = s_m = 1. \)
    • Tasks: (m heavy ones) \( \tau_1 = \tau_2 = \ldots = \tau_m = (1 + \varepsilon, 1), \)
      (n-m light ones) \( \tau_{m+1} = \tau_{m+2} = \ldots = \tau_{m+n} = (\varepsilon, n-m). \)

Formal explanation is quite mathematical, and is provided in the paper as an appendix.
The intuition is that, if any of the m heavy tasks are fixed on \( s_1, \) then a situation similar to the m-task-have-to-migrate example will occur, where the \( \varepsilon \) extra capacity on \( s_1 \) and the (n-m) light tasks do not matter much.
Linear Programming

• n*m variables.
• $x_{i,j}$ denote the share of $\tau_i$ on processor $j$.

• the total allocated share on each processor does not exceed the processor’s capacity;
  (each processor is not overutilized)

$$\sum_i x_{i,j} \leq s_j \quad \text{for } 1 \leq j \leq m$$

• the total allocated share of a task matches the task’s utilization.
  (each task will receive enough processor supply)

$$\sum_j x_{i,j} \geq u_i \quad \text{for } 1 \leq i \leq n$$
\[
\sum_i x_{i,j} \leq sj \quad \text{for} \ 1 \leq j \leq m
\]
\[
\sum_j x_{i,j} \geq ui \quad \text{for} \ 1 \leq i \leq n
\]

- \( x_{1,1} = 2, \ x_{1,2} = 0 \);
- \( x_{2,1} = 1, \ x_{2,2} = 1 \).

\( \tau_1 = (2,1) \)

\( \tau_2 = (2,1) \)

- \( s_1 = 3 \)
- \( s_2 = 1 \)
- \( \tau_1 = \tau_2 = (2,1) \)

Task executing in parallel is not allowed!
Example

\( s'_1: \; z_1 = 8 \)
\( s'_2: \; z_2 = 2 \)
\( s'_3: \; z_3 = 1 \)

\( J_1: \; u_1 \cdot F = 5F \)
\( J_2: \; u_2 \cdot F = 3F \)
\( J_3: \; u_3 \cdot F = 3F \)
Example

- $s_1 = 8$
- $s_2 = 4$
- $s_3 = 4$

Tasks:
- $J_1: u_1 \cdot F = 5F$
- $J_2: u_1 \cdot F = 3F$
- $J_3: u_1 \cdot F = 3F$

Fixed Tasks
Example

\[ s_1 = 8 \]
\[ s_2 = 4 \]
\[ s_3 = 4 \]

Fixed Tasks

- \( J_1: u_1 \cdot F = 5F \)
- \( J_2: u_1 \cdot F = 3F \)
- \( J_3: u_1 \cdot F = 3F \)
Example

\[ s_1 = 8 \]
\[ s_2 = 4 \]
\[ s_3 = 4 \]

\( J_1: u_1 \cdot F = 5F \)
\( J_2: u_1 \cdot F = 3F \)
\( J_3: u_1 \cdot F = 3F \)

Fixed Tasks
FAQs

- People complain that uniform platforms are totally unrealistic! Some executions, e.g., accesses to memory sub-systems, may not be scaled by CPU frequency (speed).
- Resolved by introducing more pessimism.

- Approximate restricted processor supplies.