Segment-Fixed Priority Scheduling for Self-Suspending Real-Time Tasks

Junsung Kim,
Björn Andersson†, Dionisio de Niz†, and Raj Rajkumar
Carnegie Mellon University
† Software Engineering Institute

A motion planning algorithm outputs the best path using:
- Parallel threads that calculate the cost of each possible path
- Master thread that picks up the best path

Parallelism is essential for the motion-planning algorithm to meet its deadline.
Motion Planning with Parallelism

**Multi-core Processor**
- A **quad-core** Intel processor was used.
  - We proposed a scheduling algorithm for this at ICCPS 2013.
- Some **challenging cases** are still present.
  - When a difficult maneuver is necessary (e.g. parking lot)
  - When there are just too many obstacles on our path

**Many-core Processor**
- Server-class processors are not an option due to space, heating and cost constraints.
- Using a GP-GPU is a good option.
  - Task executes on CPU, suspends, executes on GPU, and then resumes execution on CPU → **self-suspension**

**How to deal with tasks that self-suspend?**
Our Goals

- Dealing with tasks that self-suspend
  → Does not satisfy the assumptions of RMS
- Identify cases in which $RMS$ is still optimal
- Find a utilization bound if possible
- Discover a way to schedule self-suspending tasks when RMS is sub-optimal

Glimpse of the solution

- Assign a priority to each segment of a job
  - Segment: a continuous portion of execution without self-suspension
- Leverage phase enforcement for each segment
Task Model

Model of a self-suspending real-time task

\[ \tau_i: \left( (C_{i,1}, G_{i,1}, C_{i,2}, \ldots, G_{i,s_i-1}, C_{i,s_i}), T_i \right) \]

- \( s_i \) is the **number of task segments** for \( \tau_i \).
- \( C_{i,j} \) is the **worst-case execution time for the \( j^{th} \) execution segment**, and there are \( s_i \) execution segments.
- \( G_{i,j} \) is the **worst-case suspension time for the \( j^{th} \) suspension segment**, and there are \( s_i - 1 \) suspension segments.
- \( T_i \) is the **period** of \( \tau_i \), and an implicit deadline is assumed.
Outline

- Motivation, Goals, and Models
- Task-Fixed Priority Scheduling for Self-Suspending Tasks
- Segment-Fixed Priority Scheduling
- Evaluation
- Conclusion and Future Work
Task-Fixed Priority Scheduling for Self-Suspending Tasks

**Assumptions**
- $G_{i,j} = G_{i,j}^{MAX} = G_{i,j}^{MIN}$
- $\tau_{i,j}$ always runs for $C_{i,j}$.
- No phase enforcement is used.

**Glimpse of the results**
- The conventional critical instant does *not* always hold.
- When it does hold, a utilization bound exists.
Failure of L&L Critical Instant

- One self-suspending task and one non-suspending task $\tau_1: (1, 1, 3, 5)$ and $\tau_2: (2, 7)$

- $\tau_2$ misses its deadline at the second job release.
- This violates the conventional definition of a critical instant.
- It actually depends on which segment of $\tau_1$ is released with $\tau_2$. 
Critical Instant Failure (cont’d)

- For a taskset with one self-suspending task $\tau_1$ and one non-suspending task $\tau_2$:
  - Critical instant candidates arise when $\tau_2$ arrives at the same time as any of the segments of $\tau_1$
    - Pick the worst (paper has the proof)

- For a taskset with one self-suspending task $\tau_1$ and many non-suspending tasks:

\[ \tau_1: (1,2\epsilon, 2, 5) \]
\[ \tau_2: (\epsilon, 5 + \epsilon) \]
\[ \tau_3: (3\epsilon, 5 + 2\epsilon) \]

The critical instant for $\tau_2$ may be different from that of $\tau_3$.

* The paper provides an algorithm to find the critical instant of each task.
When Does RMS Hold Good?

- For a taskset with one self-suspending task and \( n - 1 \) non-suspending tasks:
  - When \( R_1 = C_1 + G_1 < C_i \leq T_1 - R_1 \), where \( i \leq n \)
    - Please refer to the paper for the proof.
  - Then, the utilization bound (UB) for this case is given by
    \[
    U_{RM-SS}(n, k) = n \left( (2 + 2k)\frac{1}{n} - 1 \right) - k
    \]
    where, \( k = \frac{G_1}{T_1} \)
    - \( k \) lies in \([0, 2^{n-1} - 1]\)
    - When \( k = 0 \), it simply becomes the Liu and Layland bound.
Motivation → TFPS → SFPS → Evaluation → Conclusion

UB with One Self-Suspending Task

\[ U_{RM-SS}(n) = n \left( (2 + 2k)^{\frac{1}{n}} - 1 \right) - k, 0 \leq k \leq 2^{\frac{1}{n-1}} - 1 \]
Case of Multiple Self-Suspending Tasks

- Need to consider all possible release offsets
- Consider a set of two tasks scheduled with RMS:
  - $\tau_1: (1,1,1), 5$
  - $\tau_2: (2,5,2), 10$
  - $\tau_{1,2}$ and $\tau_{2,1}$ arrive together

Observation:
There is enough slack, but it is not used well.

Key Idea:
Assign a different priority to each segment.
Multiple Self-Suspending Tasks (cont’d)

- Idea: assign a higher priority to a segment that needs to complete sooner

- Consider a set of two tasks:
  - $\tau_1: (1,1,1), 5$
  - $\tau_2: (2,5,2), 10$
  - New priority assignment: $\tau_{2,1} > \tau_{1,1} > \tau_{1,2} > \tau_{2,2}$

- $\tau_{1,2}$ and $\tau_{2,1}$ arrive together
- $\tau_{1,1}$ and $\tau_{2,1}$ arrive together
Outline

- Motivation, Goals, and Models
- Task-Fixed Priority Scheduling for Self-Suspending Tasks
- Segment-Fixed Priority Scheduling
- Evaluation
- Conclusion and Future Work
Segment-Fixed Priority Scheduling

- **Assumptions**
  - $G_{i,j}$ lies in $[G_{i,j}^{MIN}, G_{i,j}^{MAX}]$.
  - $\tau_{i,j}$ always runs for at most $C_{i,j}$.
  - This can be extended to sporadic tasks.

- **Glimpse of the results**
  - Segment-fixed priority scheduling
  - An exact schedulability analysis based on MILP
  - The optimal priority and phase assignment
  - Some heuristics
Segment-Fixed Priority Scheduling

- Each segment $\tau_{i,j}$ has its own priority.
- $\tau_{i,j}$ does not arrive before $\phi_{i,j}$.
  - The varying suspension time yields many different phase offsets.
Mixed-Integer Linear Programming

For an exact schedulability analysis of **task**-fixed priority scheduling:

- Find $R_1$, $R_2$, and $R_3$ such that
  
  $R_1 = C_1$
  
  $R_2 = C_2 + \left\lceil \frac{R_2}{T_1} \right\rceil C_1$
  
  $R_3 = C_3 + \left\lceil \frac{R_3}{T_1} \right\rceil C_1 + \left\lceil \frac{R_3}{T_2} \right\rceil C_2$

- $R_1 \leq D_1$
- $R_2 \leq D_2$
- $R_3 \leq D_3$

The taskset is schedulable if a feasible solution exists.

A traditional response-time test
Mixed-Integer Linear Programming

For an exact schedulability analysis of task-fixed priority scheduling:

- Find $R_1$, $R_2$, and $R_3$ such that

\[
R_1 = C_1 \\
R_2 = C_2 + I_{2,1} C_1 \\
R_3 = C_3 + I_{3,1} C_1 + I_{3,2} C_2
\]

\[
I_{2,1} = \left\lfloor \frac{R_2}{T_1} \right\rfloor \\
I_{3,1} = \left\lfloor \frac{R_3}{T_1} \right\rfloor \\
I_{3,2} = \left\lfloor \frac{R_3}{T_2} \right\rfloor
\]

- $I_{2,1}$, $I_{3,1}$, and $I_{3,2}$ are integers.

\[
R_1 \leq D_1 \\
R_2 \leq D_2 \\
R_3 \leq D_3
\]

The taskset is schedulable if a feasible solution exists.
Mixed-Integer Linear Programming

For an exact schedulability analysis of task-fixed priority scheduling:

- Find $R_1$, $R_2$, and $R_3$ such that
  \[ R_1 = C_1 \]
  \[ R_2 = C_2 + I_{2,1}C_1 \]
  \[ R_3 = C_3 + I_{3,1}C_1 + I_{3,2}C_2 \]
  \[ I_{2,1}T_1 - T_1 < R_2 < I_{2,1}T_{3,1} \], and $I_{3,2}$ are integers.
  \[ I_{3,1}T_1 - T_1 < R_3 < I_{3,1}T_1 \]
  \[ I_{3,2}T_2 - T_2 < R_3 < I_{3,2}T_2 \]
  \[ R_1 \leq D_1 \]
  \[ R_2 \leq D_2 \]
  \[ R_3 \leq D_3 \]

The taskset is schedulable if a feasible solution exists.
Mixed-Integer Linear Programming

- For an exact schedulability analysis of task-fixed priority scheduling:
  - Find $R_1$, $R_2$, and $R_3$ such that
    
    \[
    R_1 = C_1 \\
    R_2 = C_2 + I_{2,1} C_1 \\
    R_3 = C_3 + I_{3,1} C_1 + I_{3,2} C_2 \\
    I_{2,1} T_1 - T_1 \leq R_2 \leq I_{2,1} T_1 \\
    I_{3,1} T_1 - T_1 \leq R_3 \leq I_{3,1} T_1 \\
    I_{3,2} T_2 - T_2 \leq R_3 \leq I_{3,2} T_2 \\
    R_1 \leq D_1 \\
    R_2 \leq D_2 \\
    R_3 \leq D_3
    \]

  Changing $<$ to $\leq$ does not affect the feasibility.
  (See the paper)

The taskset is schedulable if a feasible solution exists.
For an exact schedulability analysis of task-fixed priority scheduling:

- Find $R_1$, $R_2$, and $R_3$ such that

\[
\begin{align*}
R_1 &= C_1 \\
R_2 &= C_2 + I_{2,1} C_1 \\
R_3 &= C_3 + I_{3,1} C_1 + I_{3,2} C_2 \\
I_{2,1} T_1 - T_1 &\leq R_2 \leq I_{2,1} T_1 \\
I_{3,1} T_1 - T_1 &\leq R_3 \leq I_{3,1} T_1 \\
I_{3,2} T_2 - T_2 &\leq R_3 \leq I_{3,2} T_2 \\
R_1 &\leq D_1 \\
R_2 &\leq D_2 \\
R_3 &\leq D_3
\end{align*}
\]

This is linear, and an MILP problem.

The taskset is schedulable if a feasible solution exists.
MILP for Priority Assignment

- Introducing a few parameters for priority assignment
  - Find $R_1, R_2, R_3, x_{1,2}, x_{1,3}, x_{2,1}, x_{2,3}, x_{3,1},$ and $x_{3,2}$ such that
    
    $R_1 = C_1 + \left[ \frac{R_1}{T_2} \right] C_2 x_{1,2} + \left[ \frac{R_1}{T_3} \right] C_3 x_{1,3}$
    
    $R_2 = C_2 + \left[ \frac{R_2}{T_1} \right] C_1 x_{2,1} + \left[ \frac{R_2}{T_3} \right] C_3 x_{2,3}$
    
    $R_3 = C_3 + \left[ \frac{R_3}{T_1} \right] C_1 x_{3,1} + \left[ \frac{R_3}{T_2} \right] C_2 x_{3,2}$
    
    $R_1 \leq D_1$
    
    $R_2 \leq D_2$
    
    $R_3 \leq D_3$

- $x_{i,j}$ is a binary variable, and it becomes 1 if $\tau_j$ has higher priority than the priority of $\tau_i$.
  - When $x_{i,j}$ is given, this can be used to check the schedulability.
  - The phase assignment can be found using a similar approach.
Heuristics for Priority and Phase Assignment to Task Segments

- **Idea**: assign a deadline to a task segment such that the Deadline-Monotonic policy can be used
- **Four heuristics that assign a deadline to a segment**:
  - **ED (Equal Density)**: Assign to $\tau_{i,j}$ a segment deadline so that all segment densities for $\tau_i$ are same.
  - **MTD (Minimize Total Density)**: Assign to $\tau_{i,j}$ a segment deadline such that the total density for $\tau_i$ is minimized.
  - **ES (Equal Slack)**: Assign to $\tau_{i,j}$ a segment deadline such that
    $$D_{i,1} - C_{i,1} = D_{i,2} - C_{i,2} = \cdots = D_{i,s_i} - C_{i,s_i}.$$  
  - **PS (Proportional Slack)**: Assign to $\tau_{i,j}$ a segment deadline such that
    $$D_{i,j} - C_{i,j} : D_{i,j+1} - C_{i,j+1} :: U_{i,j} : U_{i,j+1}.$$
Outline

- Motivation, Goals, and Models
- Task-Fixed Priority Scheduling for Self-Suspending Tasks
- Segment-Fixed Priority Scheduling
- Evaluation
- Conclusion and Future Work
Evaluation Parameters

- Generated 100 tasksets per datapoint such that
  - The number of tasks varies from 2 to 16
  - The total utilization of each taskset varies from 0.1 to 1
  - The period of each task is uniformly distributed between 10 and 100
  - The maximum value of $\frac{G_i}{T_i}$ is either 0.1 or 0.6
  - The number of segments is set to 2
  - We assume $D_i = T_i$
Evaluation Parameters (cont’d)

- Techniques compared:
  - **OPT**: Exact schedulability analysis based on MILP
  - **RM**: Rate-Monotonic
  - **ES**: Equal Slack
  - **ED**: Equal Density
  - **MTD**: Minimize Total Density
  - **PS**: Proportional Slack
**When** \[ \max_i \frac{G_i}{T_i} \leq 0.1 \]

- All heuristics perform better than RM.
- ED performs the best among all techniques.
- The performance difference between OPT and ED gets larger as the total taskset utilization becomes larger.
When $\max_i G_i / T_i \leq 0.6$

As the suspension time becomes larger, it is **difficult** for tasks to meet their deadlines (due to tight constraints) regardless of the amount of CPU idle time.
Related Work

- Scheduling self-suspending tasks with task-fixed priority scheduling is **NP-hard** in the strong sense. [Ridouard 04]

- Other contributions

<table>
<thead>
<tr>
<th></th>
<th>Soft Real-time</th>
<th>Hard Real-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suspension-agnostic</td>
<td>[Elliott 12]</td>
<td>[Gai 03]</td>
</tr>
<tr>
<td>Suspension-aware</td>
<td>[Liu 09] [Liu 12]</td>
<td>[Rajkumar 91] [Gai 03] [Lakshmanan 10]</td>
</tr>
</tbody>
</table>

Scheduling self-suspending tasks with task-fixed priority scheduling is NP-hard in the strong sense. [Ridouard 04]
Conclusions and Future Work

- **Dealing with tasks that self-suspend**
  - Determined a new *critical instant* when there is one self-suspending task and multiple non-suspending tasks.
  - Derived a *utilization bound* when RMS is still optimal.
  - Proposed *segment-fixed priority scheduling* (SFPS).
  - Performed *exact schedulability* analysis for SFPS.
  - Gave an *optimal configuration* for priority and phase offset.
  - Four practical *heuristics* were proposed and evaluated.
    - Deadlines that lead to equal density of segments do best.

- **Future work**
  - Multi-core processing
Thank you and Questions?

Junsung Kim: junsungk@cmu.edu