Addressing Cache Related Preemption Delay in Fixed Priority Assignment

Hai-Nam TRAN, Frank SINGHOFF, Stéphane RUBINI, Jalil BOUKHOBZA

Lab-STICC, UMR CNRS 6285, Université de Bretagne Occidentale
Outline

1. Introduction
2. Approach
3. Evaluation
4. Conclusion & Future Works
1. Introduction: Real-Time Critical System

- **System**: uniprocessor with one level of direct mapped instruction cache.
  - Cache → higher performance, lower predictability

- **Fixed-priority Preemptive scheduler**
  - Priorities of tasks are preliminary assigned.
  - Processor executes the highest priority task that is ready to execute
  - **Preemption**: Higher priority task can preempt lower priority task.

- **Cache Related Preemption Delay (CRPD)**
  - Definition: the additional time to refill the cache with the cache blocks evicted by the preemption.

- **Cache Access Profile**
  - A task has a fixed set UCB and ECB.
  - Any partial execution of a task uses all of its UCBs and ECBs.
2. Introduction: Problem Statement

- **CRPD**
  - High, non-negligible preemption cost.
  - Can present up to 44% of WCET of a task (Pellizzoni et al 2007).

<table>
<thead>
<tr>
<th>Cache Size: 512 KB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total task size</strong></td>
</tr>
<tr>
<td>in memory</td>
</tr>
<tr>
<td>1 KB to 512KB</td>
</tr>
<tr>
<td>512 KB to 1024 KB</td>
</tr>
</tbody>
</table>

- **Priority assignment → Number of preemption, preempted/preempting task → Total CRPD.**
  - **NO** fixed priority assignment algorithm guarantees that the system is schedulable when CRPD is taken into account.
  - i.e: non-schedulable system in practice is assumed to be schedulable at design time.

Impact of CRPD to Audsley’s Optimal Priority Assignment Algorithm
2. Introduction: Contribution

- Propose a priority assignment algorithm which verifies the schedulability of a task set, takes into consideration CRPD while assigning priorities to tasks.

- Three solutions to solve the problem with different levels of pessimism, efficiency and complexity.
Outline

1. Introduction
2. Approach
3. Evaluation
4. Conclusion & Future Works
3. Approach: Based on Audsley’s OPA

Assign a priority level to a task and verify its feasibility.

$n$ tasks, $n$ priority levels. Start with the lowest priority level

1. Assign the priority level $i$ to a task $\tau$, other unassigned tasks are assumed to have higher priority than $\tau$.

2. Verify if $\tau$ is schedulable at this priority level
   
   If YES, assign $i$ to $\tau$, move on the higher priority level, back to step 1
   If NO, choose another task, back to step 1

3. Terminate when each task is assigned a priority level

```plaintext
1  for each priority level i, lowest first loop
2       for each unassigned task \tau loop
3          if \tau is schedulable at priority i then
4             assign \tau to priority i
5             break (continue outer loop)
6          end if
7       end loop
8       return unschedulable
9     end loop
10   return schedulable
```
3. Approach

- In step 2, a feasibility test is performed
  - A task $\tau_i$ is schedulable if all its jobs released during the feasibility interval can meet their deadlines.
  - A job of $\tau_i$ released at $t$ is feasible if:

$$\text{Interference}$$

$$\text{Worst Case Execution Time (WCET)} \quad C_i + I^t_i < D_i \quad \text{Deadline}$$

- $I^t_i$ includes
  - Computational requirement of higher priority tasks.

In the original work of Audsley, CRPD is not taken into account!
3. Approach: Extend OPA with CRPD

- In our approach, $I_i^t$ includes
  - Computational requirement of higher priority tasks
  - Cache Related Preemption Delay of
    - (1) *Higher priority tasks preempting* $\tau_i$
    - (2) *Higher priority tasks preempting each others.*

$\rightarrow$ CRPD Interference

The question is how to compute the CRPD Interference?

- In step 1,2 of OPA, relative priorities of higher priority tasks are not set.
- CRPD depends on preemption, which highly depends on priority.
3. Approach: Compute the CRPD Interference

- We propose three solutions with different levels of pessimism, efficiency and complexity to compute an upper-bound of CRPD Interference.

<table>
<thead>
<tr>
<th>Solution</th>
<th>Description</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. OPA_CRPD-ECB</td>
<td>Update <strong>WCET</strong> of higher priority tasks with worst-case impact of CRPD caused by this task.</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>2. OPA_CRPD-PT</td>
<td>Evaluate all potential preemption points. Compute worst case CRPD at each potential preemption.</td>
<td>$O\left(\frac{n}{2}\right)$</td>
</tr>
<tr>
<td>2’. OPA_CRPD-PT-Simplified</td>
<td>More pessimistic CRPD computation with less complexity</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>3. OPA_CRPD-Tree</td>
<td>Evaluate all possible preemption sequences.</td>
<td>$O(n!)$</td>
</tr>
</tbody>
</table>
3. Approach: OPA_CRPD-PT

- **Potential preemption (PP).**
  A potential preemption amongst jobs of tasks with no complete priority assignment is a preemption that may occur when a job is released while other jobs did not complete their executions.

- **Assumption 1: Number of preemption:** All PP occurs.
- **Assumption 2: CRPD:** A PP occurs with the maximum number of preempted tasks and the maximum number of evicted UCBs.

![Diagram](image)

- Preempting task
- Cache block reload time
- UCBs of task $k$
- $\gamma_{pp(j), j} = BRT. \left( \bigcup_{\forall k \in pp(j)} UCB_k \right) \cap ECB_j \rightarrow ECBs$ of task $j$

Set of in-complete jobs at the release time of a job of task $\tau_j$
- Number of element: max in-complete jobs
- Elements: Jobs with the largest set of evicted UCBs
3. Approach: OPA_CRPD-Tree

- Evaluate all preemption sequences of jobs of higher priorities tasks.
- **Data is stored in tree structure**
  - Branching when there is a potential preemption.
  - 1 branch = 1 sequence of preemption.
  - Implicit priority are set at each potential preemption.
  - No conflict implicit priorities in a branch.
- **CRPD Interference is computed by taking into account the branch with the largest total CPRD.**
Outline

1. Introduction
2. Approach
3. Evaluation
4. Conclusion & Future Works
4. Evaluation: Experiment 1 – Efficiency

  Task set size is fixed at 5, processor utilization is varied from 50 to 95
- **Evaluate:** number of task set found schedulable. Verify the schedulability with scheduling simulation with simulation interval of 2*H. Compare with an exhaustive approach.
- **Results:**
  - All task sets **assumed to be** schedulable are schedulable over the scheduling simulation interval

![Graph showing number of schedulable tasks vs processor utilization](image)
4. Evaluation: Experiment 2 - Complexity

- Configuration is based on the work of Altmeyer et al, 2012. Detailed configuration in the article presented at ETFA 2015. Processor utilization is fixed at 80, task set size is varied from 1 to 30, 1000 tasks sets are generated for each task set size.

- **Evaluate**: Computation time
- **Results**: Significant different in terms of complexity

![Graph showing computation time vs. number of tasks]
5. Conclusion & Future Works

• **Objective**
  - Propose a priority assignment algorithm which verifies the schedulability of a system, takes into consideration CRPD while assigning priorities to tasks.

• **Approach**
  - Based on Audsley’s Optimal Priority Assignment (OPA) algorithm.
  - Extend the feasibility test in OPA to take into account the CRPD.

• **Outcome**
  - Three solutions with different levels of pessimism, efficiency and complexity
5. Conclusion & Future Works

• **Current work**
  - Implement a cache-aware scheduling simulator. (presented at EWiLi 2015)
  - Implement WCRT+CRPD approaches.
    - ECB Only. (Busquet-Mataix et al., 1996)
    - UCB Only. (Lee et al., 1998)
    - ECB_Union_Multiset. (Altmeyer et al., 2012)
    - UCB_Union_Multiset. (Altmeyer et al., 2012)
    - Combined Approach. (Altmeyer et al., 2012)

• **Future work**
  - Tighten the feasibility condition.
  - Study the problem of feasibility interval of system with cache.
  - Compare our approach with other feasibility tests.
THANKS FOR YOUR ATTENTION !
References


4. Evaluation: Base configuration

• **Task configuration**
  - Task periods are uniformly generated from 80 to 200 units of time
  - Processor utilizations (PU) are generated by UUniFast algorithm
  - Task WCETs are set based on the generated processor utilizations and periods.
  - Task offsets are uniformly distributed from 1 to 30 units of time

• **Cache configuration**
  - The cache is direct mapped. The number of cache blocks is 16. Block reload time is 1 unit of time.
  - Cache utilization (CU) of tasks are generated using the UUniFast algorithm, total CU = 5.
  - Cache reuse factor = 0.3

• **Configuration is based on the work of Altmeyer et al, 2012.**