Integration of Cache Partitioning and Preemption Threshold Scheduling to Improve Schedulability of Hard Real-Time Systems

Chao Wang¹, Zonghua Gu¹, Haibo Zeng²

¹Zhejiang University, China
²Virginia Tech, USA

{superwang1988, zgu}@zju.edu.cn, hbzeng@vt.edu

July 30, 2015
For preemptive scheduling, task preemptions cause Context Switch Cost (CSC) and Cache Related Preemption Delay (CRPD).

- Upon task preemption, Context Switch Cost (CSC) is incurred at context-switch time; CRPD is incurred after pre-emption as $\tau_2$ accesses cache lines that $\tau_1$ evicted from cache. (figure from [Lunniss 2014])
- CSC can also be considered as part of CRPD
Cache Partitioning

- **Cache partitioning**: dedicate a portion of the cache space to a task or a set of tasks
  - Helps reduce or eliminate CRPD
  - Can be coarse-grained or fine-grained
## Approaches to Cache Partitioning

- **Way based cache partitioning**
  - Coarse-grained
  - Needs processor hardware support, e.g., lockdown register, supported by ARM Cortex A9, Freescale P4080, TI OMAP4460

<table>
<thead>
<tr>
<th></th>
<th>Way 0</th>
<th>Way 1</th>
<th>Way 2</th>
<th>...</th>
<th>Way 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Approaches to Cache Partitioning

- Set based cache partitioning
  - Fine-grained
  - Page coloring; can be implemented in software with the OS page table mechanism

<table>
<thead>
<tr>
<th></th>
<th>Way 0</th>
<th>Way 1</th>
<th>Way 2</th>
<th>...</th>
<th>Way 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Approaches to Cache Partitioning

- Combined cache partitioning
  - Even finer granularity
- Our work is independent of the cache partitioning mechanism.

<table>
<thead>
<tr>
<th></th>
<th>Way 0</th>
<th>Way 1</th>
<th>Way 2</th>
<th>...</th>
<th>Way 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Per-Task Partitioning

- Traditional cache partitioning strategy: **per-task** cache partitioning
  - Each task occupies a cache partition exclusively
- Drawback: each task has smaller cache space available
  - Increased \# tasks → increased WCET of each task → reduced schedulability
Each task $\tau_i$ has two priorities: nominal priority $P_i$ upon release; preemption threshold (PT) $\pi_i \geq P_i$ while running.

- Preemption graph: an edge from $\tau_i$ to $\tau_j$ means $\tau_i$ can preempt $\tau_j$, iff $P_i > \pi_j$
  - If forall $i$, $\pi_i = P_i$, then turns into preemptive scheduling
  - If forall $i$, $\pi_i = \infty$, then turns into non-preemptive scheduling
  - By adjusting $\pi_i$, we can adjust degree of preemptiveness
  - PTS benefits: minimizing stack space requirement; reducing # preemptions; improving schedulability (sometimes)

(a) Fully Preemptive Scheduling
(b) Preemption Threshold Scheduling
(c) Preemption graph for task-set in (b)
Combine PTS and Cache Partitioning

- Observation: a group of mutually non-preemptive tasks can share the same cache partition without causing CRPD overheads.
- Form Non-Preemptive Groups (NPG) by assigning the same PT to all tasks in the same NPG, equal to or higher than the highest priority of the tasks in that subset. Assign a separate cache partition to each NPG.
  - There is no CRPD among tasks in the same NPG, since there is no preemption among tasks in the same NPG.
  - There is no CRPD between different NPGs, since tasks in different NPGs do not share cache.
  - Optimization problem: How to arrange tasks in NPGs, and how to assign cache partition sizes to NPGs, to maximize schedulability?
Motivational Example

- Cache has total size of 2.
- Nonschedulable with fixed-priority preemptive scheduling, and task-based cache partitioning
  - Two tasks share the entire cache: CPU utilization $(2/4 + 2/4) + \text{CRPD} > 1$;
  - Each task is assigned cache size 1: CPU utilization $(3/4 + 3/4) > 1$
- Schedulable with PTS + cache partitioning
  - Assign $\tau_1$’s PT $\pi_1 = p_2 = 2$, so $\tau_1$ and $\tau_2$ are non-preemptive
  - Two tasks share the entire cache with no CRPD: CPU utilization $(2/4 + 2/4) = 1$
Sebastian Altmeyer, Roeland Douma, Will Lunniss, Robert I. Davis,

- Presented an optimal algorithm for cache partitioning on a uniprocessor to improve schedulability of preemptive fixed-priority scheduling, based on exhaustive search with branch-and-bound. They compared two extreme cases: either all tasks share the entire cache, or every task has its own individual cache partition.

- We consider the more general case, where a Non-Preemptive Group of tasks can share the same cache partition.
Task Model

- Task WCETs $\overrightarrow{C_i} = \{C_{i,k} | 1 \leq k \leq K\}$: WCET is $C_{i,k}$ if $\tau_i$ is assigned Cache Size $CS_k$.

- We assume a task’s WCET $C_{i,k}$ vs. $CS_k$ is a monotonically and strictly decreasing function, even though the function of $C_{i,k}$ vs. all possible cache sizes may not be monotonically decreasing.
  - In case of anomaly, where an increased Cache Size causes an increase in task WCET, we omit such Cache Sizes from the set of $CS_k$ under consideration, e.g., Cache Sizes of 1 and 3 are omitted from the set of $CS_k$ values.
  - If WCET stays the same or increases if Cache Size goes from 0 to 1, or from 2 to 3, it is more advantageous to choose the smaller Cache Size of 0 or 2, instead of 1 or 3.
Function of WCET vs. Cache Size is not monotonically decreasing, but over-/under approximations of the WCET function are both monotonically decreasing.

We use over-approximation of the WCET function in the experiments, as an alternative to omitting anomaly points (previous slide)
Parameters and Variables for task $T_i$

- **Constant parameters**
  - $T_i$: Period
  - $D_i$: Deadline
  - $P_i$: Nominal Priority
  - $C_i = \{C_{i,k}|1 \leq k \leq K\}$: WCET

- **Optimization variables**
  - $\pi_i$: Preemption threshold
  - $cps_i$: Assigned cache partition size
MILP Formulation

- Optimization objective is null (any feasible solution is OK.)
- Constraint set:
  - Cache partitioning
    - $a_{i,k} = 1$: Task is assigned with cache partition size $CS_k$
    - $cps_i = \sum_{k=1,\ldots,K} a_{i,k} \cdot CS_k$
  - PT assignment
    - $b_{i,pt} = 1$: Task is assigned with PT $pt$
    - $\pi_i = \sum_{pt={p_i,\ldots,N}} b_{i,pt} \cdot pt$
  - Cache size constraints
    - $b_{i,pt} = 1 \implies s_{pt} = cps_i$: Each PT corresponds to a NPG, all tasks in a NPG share the same cache partition size
    - PT values implicitly determine NPGs and their assigned cache partition sizes
    - $\sum_{pt={1,\ldots,N}} s_{pt} \leq TCS$
Encoding the PTS schedulability test

- Original PTS schedulability test:
  
  \[ S_i(q) = B_i + q \cdot C_i + \sum_{\tau_j \in hp(i)} (1 + \left\lfloor \frac{S_i(q)}{T_j} \right\rfloor) \cdot C_j \]
  
  \[ F_i(q) = S_i(q) + C_i + \sum_{\tau_j \in ht(i)} \left( \left\lceil \frac{F_i(q)}{T_j} \right\rceil - (1 + \left\lfloor \frac{S_i(q)}{T_j} \right\rfloor) \right) \cdot C_j \]
  
  \[ R_i = \max_{q \in \{0, 1, \ldots, \left\lfloor \frac{L_i}{T_i} \right\}} (F_i(q) - q \cdot T_i) \]
  
  \[ R_i \leq D_i \]

- The test checks schedulability of all jobs in the level-\(i\) busy period, which depends on task WCETs, which in turn depends on cache partitioning, which in turn depends on PT assignment, since each PT value uniquely identifies a cache partition.
  
  - We cannot know the number of jobs in the busy period without knowing the PT assignments.
  
  - To break the circular dependency, we check schedulability of only the first job of each task in the ILP formulation.
  
  - To reject any false negative solutions, always run the full PTS schedulability test to confirm the schedulability of solutions found by the ILP model.
Heuristic solution

- Divide the problem in two sub-problems
  - How to assign cache partition size to each task
  - How to search feasible NPG assignment while satisfying cache size constraints and schedulability test

- Algorithm details:
  1. Assign minimum cache partition size to each task
  2. Solve a simpler MILP to find feasible PT assignment. If found, then return; else continue
  3. Increase the cache partition size of the task with the most beneficial effect when its cache partition size is increased, measured by the increase in number of schedulable tasks as Cache Partition Size goes from $CS_k$ to $CS_{k+1}$
     \[ \Delta_i = NST(cps_i = CS_{k+1}) - NST(cps_i = CS_k) \]
  4. Goto step 2
Experiments

- **Evaluated approaches**
  - Fully-Preemptive Scheduling (FPS): ideal fixed-priority
    Fully-Preemptive Scheduling with no cache partitioning, where all tasks share the entire cache space (either instruction or data), and CRPD is assumed to be zero, giving it an unfair advantage.
  - Non-Preemptive Scheduling (NPS)
  - FPS-BnB: the B&B approach proposed in [Altmeyer et al. 2014]
  - PTS-ILP: MILP formulation
  - PTS-Heu: Heuristic formulation

- **Random generated tasks**
  - Total utilization: \([0.2, 1]\) in steps of 0.05; UUnifast
  - WCET vs. cache partition size: randomly select one benchmark from PapaBench and the Malardalen benchmark suites for each task, and use over-approximation of the WCET function. (Data provided by [Altmeyer et al 2014]).
  - Task period: log-uniform distribution in \([5, 5000]\)
  - Rate-Monotonic priority assignment
  - 1000 tasksets for each utilization
Random generated tasks: Taskset size $N = 5$

(a) Ratio of task-sets deemed schedulable at different total utilizations (instruction cache with perfect data cache)

(b) Ratio of task-sets deemed schedulable with one approach and not the other (instruction cache with perfect data cache)

(c) Ratio of task-sets deemed schedulable at different total utilizations (data cache with perfect instruction cache)

(d) Ratio of task-sets deemed schedulable with one approach and not the other (data cache with perfect instruction cache)
Random generated tasks: Taskset size $N = 10$

(a) Ratio of task-sets deemed schedulable at different total utilizations (instruction cache with perfect data cache)

(b) Ratio of task-sets deemed schedulable with one approach and not the other (instruction cache with perfect data cache)

(c) Ratio of task-sets deemed schedulable at different total utilizations (data cache with perfect instruction cache)

(d) Ratio of task-sets deemed schedulable with one approach and not the other (data cache with perfect instruction cache)
Random generated tasks: Taskset size $N = 20$

(a) Ratio of task-sets deemed schedulable at different total utilizations (instruction cache with perfect data cache)

(b) Ratio of task-sets deemed schedulable with one approach and not the other (instruction cache with perfect data cache)

(c) Execution time (hour) for heuristic algorithm and Altmeier’s algorithm (instruction cache with perfect data cache)
Industry Case Studies

- Case Study 1: 42 tasks
  - Heuristic algorithm finds feasible solution in 30 seconds

- Case Study 2: 90 tasks
  - Heuristic algorithm finds feasible solution in 1.5 hours

- For both, the optimal approaches (FPS-BnB, PTS-MILP) could not finish in reasonable time (days).
Integration of PTS with cache partitioning on a uniprocessor platform helps improve schedulability compared to fully-preemptive scheduling with shared cache and per-task cache partitioning.

Future work: addressing other common scheduling algorithms such as Earliest Deadline First (EDF); consider multicore processors.
The End