Optimising Task Layout to Increase Schedulability via Reduced Cache Related Pre-emption Delays

Will Lunniss\textsuperscript{1} Sebastian Altmeyer\textsuperscript{2} Robert I. Davis\textsuperscript{1}
\textsuperscript{1}Real-Time Systems Research Group, University of York, UK
\{wl510, rob.davis\}@york.ac.uk
\textsuperscript{2}Department of Computer Science, Saarland University, Germany
altmeyer@cs.uni-sb.de

RTNS 2012 - Pont à Mousson, France – 8\textsuperscript{th} & 9\textsuperscript{th} November
Outline

• Brief overview of CRPD
• Task layout
• Optimising task layout
• Case study
• Synthetic taskset experiments
• Conclusions
Background

• Caches sit between memory and the CPU
• Can store instruction, data, or both
  – We only consider instruction caches
• When fetching an instruction
  – First check the cache, if the block containing the instruction is there -> *Cache hit*
  – Otherwise, fetch the block from memory and store it into cache - > *Cache miss*
• Want to maximise cache hits as cache misses can be an order of magnitude slower
Pre-emptions and Cache

Related Pre-empt Delays (CRPD)

• Pre-empting task can evict blocks belonging to the pre-empted task

• CRPD are introduced when the pre-empted task has to reload some of those evicted cache blocks after resuming
CRPD Analysis

• **Evicting Cache Blocks (ECBs)**
  – Loaded into cache and can therefore evict other blocks

• **Useful Cache Blocks (UCBs)**
  – Reused once they have been loaded into cache before potentially being evict by the task
  – If evicted by another task, they may have to be reloaded which intrudes CRPD
  – UCBs are always ECBs
CRPD Analysis

• Example block classification

[Diagram showing ECBs and UCBs]

• Instructions inside loops are often UCBs as they get reused
CRPD Analysis

• There are a number of approaches for Fixed Priority Pre-emptive Scheduling

• Can consider:
  – The pre-empting task
  – The pre-empted task(s)
  – The pre-empted and pre-empting task(s)
CRPD Analysis

• E.g. ECB-Only is the simplest approach
  – It considers just the pre-empting task
  – Assumes that every block evicted by the pre-empting task has to be re-loaded
  – The CRPD caused by task $\tau_j$ pre-empting task $\tau_i$

$$\gamma_{i,j}^{Ec\text{b-only}} = BRT \cdot |\text{ECB}_j|$$
CRPD Analysis

• Used the combined multiset approach by Altmeyer et al. [1]
  – Considers the pre-empted and pre-empting task(s) including the different costs associated with different nested pre-emptions

Memory and Cache Layout

- Memory layout controls the cache layout
- We want to layout tasks in memory, so that the number of evicted UCBs is minimised
Optimising Task Layouts

• Used a *Simulated Annealing (SA)*
  – Starts at a initial ‘temperature’
  – Reduced by a cooling rate each iteration
  – Completes when it reaches an absolute temperature
  – Accepts large negative changes when ‘hot’ during the initial stages
Evaluating Task Layouts

- Perform *Response Time Analysis* (RTA) using integrated CRPD analysis
  - Tells us whether the taskset is schedulable at a specific utilisation
- Find the *Breakdown Utilisation* (BU)
  - Point at which a taskset becomes unschedulable
  - Found by scaling deadlines and periods
  - Driven by a binary search
- ‘Good’ layouts result in a high BU
Modifying Task Layout

• Swap two neighbouring tasks (e.g. 3 and 4)
Modifying Task Layout

- Swap two random tasks (e.g. 2 and 6)
Modifying Task Layout

- Adding a gap (e.g. after task 3)

| 1 | 2 | 3 | 4 | 5 |

- Insert up to ± half the cache size
  - But the gap can never be negative
- Reduced if the gap becomes > cache size
- Gaps are moved when swapping tasks
- Overall size of gaps limited to
  - 0%, 10% and 100% of total task size
SA Algorithm

1. Start
2. Task layout
3. Pick a change
   - Swap two neighbouring tasks
   - Swap two tasks
   - Add a random gap between two tasks
4. Evaluate the layout
5. Have we reached the absolute temperature?
   - Yes: Finish
   - No: Go back to 3
6. Accept layout
   - Yes: Go back to 3
   - No: Reject layout
7. Should we accept it anyway?
   - Yes: Go back to 3
   - No: Stop
Case Study

• Based on a code from the Mälardalen benchmark suite to create a 15 task taskset

• Setup to model an ARM7
  – 10MHz CPU
  – 2KB direct-mapped instruction cache
  – Line size of 8 Bytes, 4 Byte instructions, 256 cache sets
  – Block reload time of 8μs
Evaluation

• Compared the SA against
  – No pre-emption cost
    • All cases exclude CSC due to e.g. reloading registers
  – Sequential ordered by priority (SeqPO)
  – 1000 random layouts
  – CS[i]=0 (Aligns all tasks at cache set 0)
## Results

<table>
<thead>
<tr>
<th></th>
<th>Breakdown Utilisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>No pre-emption cost</td>
<td>0.984</td>
</tr>
<tr>
<td>SA</td>
<td>0.876</td>
</tr>
<tr>
<td>SeqPO</td>
<td>0.698</td>
</tr>
<tr>
<td>Random (min, average, max)</td>
<td>0.526, 0.685, 0.882</td>
</tr>
<tr>
<td>CS[i]=0</td>
<td>0.527</td>
</tr>
</tbody>
</table>
Case Study – SeqPO Layout

\[ \tau_1 \]

\[ \tau_5 \]

\[ \tau_{10} \]

\[ \tau_{15} \]

Cache Set

ECBs

UCBs

UCBs that could be evicted
Case Study – SA Layout

No gaps between tasks
Case Study - CRPD/task
Case Study - Explanation

• The layout generated by the SA algorithm vs SeqPO
  – Overall, more UCBs in conflict
  – However, UCBs of lower priority tasks are evicted less often
  – This shifts the CRPD from low to high priority tasks
Synthetic Tasksets

- 10 tasks per taskset
- 1000 tasksets for baseline experiments
- 512 cache sets
- Cache utilisation of 5
- Maximum UCB percentage of 30%
- Grouped UCBs into five groups spread out throughout the task

![Diagram showing ECBs and UCBs]
Baseline Experiment

![Graph showing baseline experiment results with different schedulers and utilisation levels.](image-url)
Weighted Schedulability

• Combines the data across the full range of utilisation levels into a single value
• Individual results are weighted by taskset utilisation
• We use 100 tasksets for weighted schedulability experiments
Baseline Experiment

<table>
<thead>
<tr>
<th>Weighted schedulability</th>
</tr>
</thead>
<tbody>
<tr>
<td>No pre-emption cost</td>
</tr>
<tr>
<td>SA</td>
</tr>
<tr>
<td>SeqPO</td>
</tr>
<tr>
<td>Random</td>
</tr>
<tr>
<td>CS[i]=0</td>
</tr>
</tbody>
</table>

The table shows the weighted schedulability for different scheduling algorithms. The utilisation parameter is shown on the x-axis, and the number of schedulable task sets on the y-axis.
Varying the Maximum Number of UCB Groups

![Graph showing weighted measure vs. maximum UCB groups for different strategies: No Pre-emption Cost, SA, SeqPO, Random, CS[i]=0.](graph.png)
Varying the Cache Utilisation

![Graph showing the relationship between weighted measure and cache utilisation for different strategies: No Pre-emption Cost, SA, SeqPO, Random, CS[i]=0.](image)
Varying the Maximum UCB Percentage
Varying the Number of Tasks

Graph showing the weighted measure against the number of tasks for different algorithms:
- No Pre-emption Cost
- SA
- SeqPO
- Random
- CS[i]=0

The graph indicates a decrease in weighted measure as the number of tasks increases.
Does adding gaps between tasks help?

• Not significantly
  – Varied allowed space from 0%-100%
  – Weighted measure varied from 0.463 to 0.469

• High cache utilisations and scattered UCBs means there will always be conflicts

• Reduces problem to finding the optimum permutation of task ordering

• Good for embedded systems, do not want to waste memory
Brute force comparison

- Tried all 5040 (7!) orderings for 7 tasks
- Feasible for 7 tasks, but not for higher numbers
- SA got very close using just 377
Conclusion

• Task layout has a significant effect on CRPD and schedulability
• Our SA algorithm was able to find near optimal layouts that significantly increased the breakdown utilisation of tasksets
• Found that allowing space between tasks made little difference
• Uses include:
  – Optimising an unschedulable task
  – Allowing a low power system to clocked at a lower frequency
Thank you for listening

Any Questions?