WCET-aware Software Based Cache Partitioning for Multi-Task Real-Time Systems

Sascha Plazar | Paul Lokuciejewski | Peter Marwedel

TU Dortmund University
Department of Computer Science 12
Otto-Hahn-Str. 16
44221 Dortmund
Germany
Outline

1. Introduction, Idea
2. Software Based Cache partitioning
3. WCET-aware Cache Partitioning
4. ILP Formulation
5. Workflow
6. Results
7. Conclusion
WCET-aware Cache Partitioning

Problem:
- Caches are a source of unpredictability
- Behaviour in general unpredictable in systems performing preemptive scheduling
- Different tasks could replace each other in cache

Idea:
- Divide the cache into partitions
- Assign one task per partition
  - Tasks can not replace each other
Software Based Cache Partitioning

- Exploit the cache’s modulo addressing function
- Distribute tasks in address space
- Ensure mapping to particular cache lines

- Tasks can not evict each other
  - Additional jumps have to be inserted, branches corrected
WCET-aware Cache Partitioning

Greedy approach*
- Partition size depends on task’s code size
- Example: 4 tasks with the same code size

Better
- Employ an ILP-model to select optimal cache size for every task
- Take execution frequencies into account

[* Frank Mueller, 1995: Compiler Support for Software-Based Cache Partitioning]
ILP Formulation

\( T : \text{set of periodically scheduled tasks } t_1 \ldots t_m \)

\( P : \text{set of partition sizes } p_1 \ldots p_j \)

\[ x_{ij} = \begin{cases} 1, & \text{if } t_i \text{ assigned to } p_j \\ 0, & \text{else} \end{cases} \]

\( WCET_{ij} : t_i \text{’s WCET if assigned to } p_j \)

\( I : \text{scheduling interval} \)

\( c_i : \text{task } t_i \text{ is executed exactly } c_i \text{ times during } I \)

\( \Rightarrow |I| \text{ is the Hyperperiod of } T \text{ (LCM of the tasks’ periods)} \)
ILP Formulation

Each task must have a partition assigned:

$$\forall i = 1..m : \sum_{j=1}^{n} x_{ij} = 1$$

Keep track of the cache size:

$$\sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} * p_j \leq S$$
ILP Formulation

A task’s WCET is determined:

\[
WCET(T_i) = \sum_{j=1}^{n} x_{ij} \cdot WCET_{ij}
\]

Objective function to minimize:

\[
WCET = \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} \cdot c_i \cdot WCET_{ij}
\]
Workflow

**WCET-aware C Compiler (WCC)**

- Supports Infineon TriCore TC1796
- Integrates static WCET analyzer aiT
Results: UTDSP

Average of 100 sets of randomly selected tasks:

- 5 tasks: ~8kB
- 10 tasks: ~18kB
- 15 tasks: ~26kB
Results: MRTC

Average of 100 sets of randomly selected tasks:

- 5 tasks: ~6kB
- 10 tasks: ~12kB
- 15 tasks: ~19kB
Conclusion

- WCET-driven Cache Partitioning presented
- Employed an ILP to select optimal partition sizes w.r.t. the overall system’s WCET
- Partitioning introduces predictability for preemptive scheduled systems
- Average WCET reduction of 12% (5 tasks) up to 19% (15 tasks) compared to greedy approach

Future Work

- Tightly coupling of offline schedulers
- Take task dependencies into account
Thank you for your attention!

Questions?
Distribution of Code

- Achieved by exploiting the linker
- Each portion is assigned to its own section
- Example linker script for two tasks:

```c
{text: {
   _text_begin = .;
   *(.task_part1)
   . = _text_begin + 0x80;
   *(.task2_part1)
   . = _text_begin + 0x100;
   *(.task1_part2)
   . = _text_begin + 0x180;
   *(.task2_part2)
   . = _text_begin + 0x280;
   *(.task2_part3)
} > PFLASH-C
```
Memory usage

\[ s(t_i) : \text{size of task } t_i \]
\[ p(t_i) : \text{partition size of task } t_i \]

\[ \#\text{partitions}(t_i) = \frac{s(t_i)}{p(t_i)} \]

\[ S_{\text{Flash}} \geq S_{\text{Cache}} \times \max(\#\text{partitions}(t_i)) \]

\[ S_{\text{waste}} = S_{\text{Flash}} - \sum_{i=1}^{n} s(t_i) \]
Optimization Time

- Host machine: Dual Xeon L5420 @ 2.50GHz
- Using a single core
- Complete workflow consists of:
  - Compilation: up to 3 minutes
  - Analysis: up to 1 hour / task
  - Optimization: up to 1 minute