Investigation of Scratchpad Memory for Preemptive Multitasking

Jack Whitham, Robert I. Davis and Neil Audsley*
Sebastian Altmeyer**
Claire Maiza***

* RTS Group, University of York (UK)
** Compiler Design Lab, Saarland University (Germany)
*** Verimag, INP Grenoble (France)
Part 1

In this paper...

- We compare two varieties of local memory, for a preemptive multitasking real-time system, using schedulability tests for the comparison.
Schedulability Test

- Given a task set:
  - $n$ tasks: $\tau_1, \tau_2, ..., \tau_n$
  - Deadline, period, etc. defined for each $\tau$
- and given a system:
  - CPU, memory, RTOS, resource policies
- are the tasks guaranteed to meet their deadlines?
  - Are they *schedulable*?
Schedulability *Comparison*

- Two schedulability tests together
- Same task set:
  - $n$ tasks: $\tau_1, \tau_2, ..., \tau_n$
  - Deadline, period, etc. defined for each $\tau$
- Two different systems:
  - CPU, memory, RTOS, resource policy 1
  - CPU, memory, RTOS, resource policy 2
- Interesting case: when the task set is schedulable with one system and not the other
Local Memory

- External memory accesses are slow (latency)
- Tasks store frequently-used code/data in local memory
- Two alternative ways to manage local memory:
  - Cache
  - Scratchpad Memory (SPM)
Local Memory: Cache

- Cache holds a copy of recently-accessed code/data from external memory
  - Cache is filled as a side-effect of execution
Local Memory: Cache

- Easy to write tasks that use cache
- Quite difficult to *analyse* tasks that use cache
- Determining a precise bound on the execution time:
  - Not possible for all types of cache (pessimism, tool support)
  - Not possible for all types of task
Local Memory: SPM

- SPM is used explicitly by the task
  - Code/data moved to/from SPM as required
Local Memory: SPM

- Easy timing analysis
- But, it is harder to write tasks that use SPM
  - Tricky memory management issues
  - Limited tool support
- Cache vs. SPM may be regarded as a tradeoff between difficulty of programming and difficulty of timing analysis
Preemptive Multitasking

- At all times, the highest priority runnable task is executed by the CPU.
Multitasking and Cache

- If local memory is cache:
  - Cache hardware is not aware of task switches
  - Different tasks compete for cache space and can evict each other's cache blocks (e.g. due to preemption)
  - Schedulability test considers the time cost of reloading evicted cache blocks
Multitasking and SPM

- If local memory is SPM:
  - SPM is not aware of task switches
  - RTOS must manage SPM as part of the task context
  - To do this, we apply a “multitasking SPM reuse scheme” (MSRS) at run-time*
    - MSRS pages SPM space in/out as required
    - Schedulability test considers the time cost of paging

* see [10] and section I in the paper
Part 2

Preemption-related delays and response time analysis
Response Time Analysis (RTA)

- Worst-Case Response Time, $R_i$ – the maximum interval between release and completion of $\tau_i$
Response Time Analysis (RTA)

- The famous RTA equation determines $R_i$:

$$R_i = C_i + \sum_{j \in h(p(i))} \left\lfloor \frac{R_i}{T_j} \right\rfloor C_j$$

- Used as a schedulability test: $R_i \leq D_i$
Idealism 1

- Eqn ignores *context switching* time

Incorporated by adding $C_{S_{to}}$, $C_{S_{from}}$ to RTA equation
Idealism 2

- Eqn ignores *blocking* time

Incorporated by adding $B_i$ to RTA equation (blocking due to task $\tau_i$)
Idealism 3

- Eqn ignores *preemption related delay*
  - Distinct from blocking, context switching

- Preemption related delay is additional execution time imposed upon low-priority tasks as a result of preemption
Preemption Related Delay

- X is a resource used by both tasks:
Non-ideal RTA Equation

\[ R_i = C_i + \sum_{j \in \text{hp}(i)} \left[ \frac{R_i}{T_j} \right] C_j \]

Execution and interference only

\[ R_i = B_i + CS^{\text{to}} + C_i + \sum_{j \in \text{hp}(i)} \left[ \frac{R_i}{T_j} \right] (CS^{\text{to}} + C_j + CS^{\text{from}} + \gamma_{i,j}) \]

Execution and interference, context-switching, blocking, and preemption-related delay
Cache-Related Preemption Delay

- Preemption-related delay caused by eviction of cache blocks
- Consider a small cache containing two blocks A, B
- Cache states represented as:

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty</td>
<td>τ₁</td>
<td>τ₂</td>
</tr>
<tr>
<td>A, B in use by same task</td>
<td>τ₂</td>
<td></td>
</tr>
<tr>
<td>A, B in use by different tasks</td>
<td>τ₂</td>
<td></td>
</tr>
</tbody>
</table>
```
Cache-Related Preemption Delay

- Example of CRPD:

\[ \tau_1, \tau_2 \]

Time

Task priority

\( \tau_2 \) uses cache blocks A, B

Cache state

A
B

\( \tau_2 \)

\( \tau_2 \)
Cache-Related Preemption Delay

- Example of CRPD:

\[ \tau_1 \] uses cache block B
\[ \tau_2 \] uses cache blocks A, B

Diagram:
- \( \tau_1 \) uses cache block B
- \( \tau_2 \) uses cache blocks A, B
- Cache state:
  - A
  - B
- Time
- Task priority
Cache-Related Preemption Delay

- Example of CRPD:

```
\text{\tau}_1 \text{ uses cache block } B \\
\text{\tau}_2 \text{ uses cache blocks } A, B \\
\text{\tau}_2 \text{ uses } B \text{ again} \\
\text{Cache miss due to preemption}
```
CRPD Modeling

- CRPD may be bounded by considering the size of set unions and intersections:
  - The set of cache blocks used by a task (evicting cache blocks, ECBs)
  - The set of cache blocks reused by a task (useful cache blocks, UCBs)
- Various investigations in previous work*

* see section II in the paper
Scratchpad-Related Preemption Delay (SRPD)

- Preemption-related delay is caused by “multitasking SPM reuse scheme” (MSRS)
- RTOS pages SPM space in/out at each context switch as required by each task
- The time cost of paging is SRPD
Multitasking SPM Reuse Scheme

Example: \( \tau_1 \) uses 1 SPM block, \( \tau_2 \) uses 2

“Save” - RTOS unloads \( \tau_2 \) from 1 SPM block and loads \( \tau_1 \) instead

“Restore” - RTOS restores \( \tau_2 \) usage of SPM
Part 3

Experiments and Results
Experimental Implementation

- Working model built on FPGA:
  - Has both SPM and Cache (use one or the other)
  - DMA unit for fast copies to/from SPM
Experimental Method

- **Task Set**
  - CRPD analysis
  - CRPD-RTA analysis
  - Schedulable with cache?

- SRPD analysis
  - SRPD-RTA analysis
  - Schedulable with MSRS?

**Comparison results**

- Assume cache hardware
- Assume SPM hardware with MSRS policy
Experimental Method

Generated task sets

- Tasks are benchmark programs
- WCET analysis using aiT software
- System timings ("Save" / "Restore" etc.) from FPGA implementation
- Tasks partitioned into regions for SPM
Experimental Method

- Upper bound on preemption-related delay computed by either CRPD or SRPD for each pair of tasks
Experimental Method

- Response-time analysis using CRPD/SRPD
- Task periods are the same for both systems
- Other parameters (e.g. $C$, $B$) are somewhat implementation-dependent
Experimental Method

- Schedulability test repeated for 100,000 task sets for each utilization
  \[ U = \{0.01, 0.02, \ldots, 0.99\} \]
  and for both types of system
Results

Fig 5, simplified, SRPD-RTA (real) and CRPD only

100,000 task sets of size 15 generated
Fig 4 (modified) based on SRPD (real) results
MSRS and Cache Comparison

- Incomparable
  - Some task sets are schedulable with one and not the other – neither dominates
- When is each preferable?
- A weighted measure of schedulability allows us to compare across many different utilizations
  - Approximately, the area under the curve
Effect of Task Set Size

Weighted Measure of Schedulability

Task Set Size

MSRS
Cache
Contention for local memory

- MSRS is most successful when there is a great deal of contention for local memory space
  - e.g. many tasks
  - e.g. small local memory
Conteintion for local memory

- Contention for *cache blocks* occurs whenever a preempting task evicts a block being reused by a preempted task
  - More likely with more tasks
  - More likely with smaller memory
- Contention for *SPM blocks* always occurs
  - Cost is independent of the number of tasks
    (Cost depends only on the preempting task)
Observations

- MSRS is similar to cache for schedulability
  - Results are (generally) close
  - Some task sets are better suited to cache or MSRS, due to contention

- MSRS may be improved
  - We assumed a naïve implementation
  - Subsequent work considers improvements
Conclusions

- Compared two approaches for sharing local memory between tasks in a real-time system (cache/MSRS)
- MSRS is better than cache for some task sets – in most cases, it is similar
- Both local memory types are valid choices for real-time systems
Thank you!
Is the highest priority task more likely to miss a deadline with MSRS? According to our experiments, this isn't significant. We performed SRPD-RTA and CRPD-RTA for task sets randomly picked with U in [0.3, 0.8] and $n = 15$, and if a task set was schedulable with only one, we found the highest-priority task that missed its deadline and added it to this chart.

→ Whether you use cache or MSRS, there is a similar distribution.
→ The usual cause for higher priorities is *blocking*, not MSRS
The set of available benchmarks depends on the memory size – which is why the graph has this strange step shape. The SPM approach cannot make use of more than about 2Kb – but the cache can, which is why it does really well with large local memory. Baseline was 128 blocks ($2^7$)
Simulator trace of an RTOS with four tasks (plus idle) running with MSRS. Black line = execution. Coloured marks = MSRS operations.
Previous slide, replotted for cache. Coloured marks represent cache misses. Some of these are due to preemption.