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: \( τ_1, τ_2, \ldots, τ_n \)
- Deadline, period, etc. defined for each \( τ \)

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.

\[ t_1 \text{ released} \]

\[ t_1 \text{ completes} \]

\[ t_2 \text{ preempted} \]

\[ t_2 \text{ resumes} \]

\[ t_1 \text{ and } t_2 \text{ are runnable; } t_2 \text{ experiences interference as } t_1 \text{ has a higher priority.} \]
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 $R_i$:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

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

- Eqn ignores *context switching* time

Incorporated by adding $CS^{to}$, $CS^{from}$ 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:

![Diagram showing preemption related delay](image)
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

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

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:

  ![](diagram)
  
  - Empty
  - A, B in use by same task
  - A, B in use by different tasks
Cache-Related Preemption Delay

- Example of CRPD:
Cache-Related Preemption Delay

- Example of CRPD:

```
\begin{tikzpicture}
  \draw[->] (0,0) -- (0,4) node[above] {task priority};
  \draw[->] (0,0) -- (6,0) node[below] {Time};

  \draw[fill=orange!50] (0.5,0) rectangle (1.5,0.5);
  \node at (1,0.25) {$\tau_2$};

  \draw[fill=orange!50] (2,0) rectangle (3,0.5);
  \node at (2.5,0.25) {$\tau_2$};

  \draw[fill=orange!50] (3.5,0) rectangle (4.5,0.5);
  \node at (4,0.25) {$\tau_2$};

  \draw[fill=orange!50] (5,0) rectangle (6,0.5);
  \node at (5.5,0.25) {$\tau_2$};

  \draw[fill=gray!50] (0.5,1) rectangle (1.5,1.5);
  \node at (1,1.25) {$\tau_1$};

  \draw[fill=gray!50] (2,1) rectangle (3,1.5);
  \node at (2.5,1.25) {$\tau_2$};

  \draw[fill=gray!50] (3.5,1) rectangle (4.5,1.5);
  \node at (4,1.25) {$\tau_2$};

  \draw[fill=gray!50] (5,1) rectangle (6,1.5);
  \node at (5.5,1.25) {$\tau_1$};

  \draw[<->,ultra thick] (0.5,0) -- (1.5,0) node[midway, above] {$\tau_2$ uses cache blocks A, B};

  \draw[<->,ultra thick] (3.5,0) -- (4.5,0) node[midway, above] {$\tau_1$ uses cache block B};
\end{tikzpicture}
```
Cache-Related Preemption Delay

- Example of CRPD:

![Diagram showing cache state and task priority over time]
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?

assume cache hardware
assume SPM hardware with MSRS policy

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
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
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
Results

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 utilisations
  - Approximately, the area under the curve
Effect of Task Set Size

Weighted Measure of Schedulability

Task Set Size

MSRS
Cache
Contestation 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
Contention 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.
Simulator Trace (Cache)

Previous slide, replotted for cache. Coloured marks represent cache misses. Some of these are due to preemption.