Ensuring Schedulability of Spacecraft Flight Software

Flight Software Workshop
7-9 November 2012

Marek Prochazka & Jorge Lopez Trescastro
European Space Agency
OUTLINE

- **Introduction**
- **Current approach to ensure FSW schedulability**
  - Schedulability analysis approach
  - Worst-case execution time measurements
  - Tools
- **Challenges of the current approach**
  - Analysis techniques
  - Sources of pessimism
  - Hardware
  - ...
- **Challenges coming in the near future**
  - Cache
  - Multi-core
  - Integrated Modular Avionics
  - Model-Driven Software Engineering
- **Conclusions**
Real-time software: Correctness is partially a function of time
- Failure to respond is as bad as a wrong response

Timing requirements
- High-level timing constraints come from system requirements (e.g. GNC), HW/SW interaction analysis, system needs
- E.g. delay between reading a sensor and updating actuator

Schedulability concerns design and implementation
- System requirements are translated to tasks and their timing properties (deadlines, offsets, jitters, latencies)
- Design constraints could come from requirements
- E.g. “statically assigned priority scheme”

Verification through analysis & testing
HARD VS. SOFT REAL-TIME

Hard real-time
- Tasks have “hard” deadline, which must be met at all times
  - Time/value function gives 1 before deadline and 0 after deadline
- This does not imply that missing a deadline has “catastrophic” consequences (that is determined by task/system criticality)

Soft real-time
- Approximate deadline
- Utility of answer degrades with time difference from deadline (time/value function is a curve) or number of missed deadlines
- Stochastic methods, probability of meeting a deadline
- “Hard real-time is hard, but soft real-time is harder”
- But having no deadline at all makes things easier indeed
Mixing tasks with hard deadlines with tasks with soft deadlines in a single system is possible

- Error-prone due to shared resources
  - Temporal, spatial and I/O partitioning could be needed
- Analysis of such system could become more complex

We consider FSW a hard real-time system

- We do not analyse consequence of overruns, instead we make all possible to avoid them (by analysis)
- In some cases missing a deadline would not cause a serious problem
  - I.e. not all deadlines should be defined as “hard”
  - Robustness analysis needed
- Recently we see some systems with tasks having no deadlines, but analysis not always fully adjusted
Ensuring Schedulability of Spacecraft Flight Software | 7-9 November 2012 | Slide 6

ESA UNCLASSIFIED – Releasable to the Public
DYNAMIC DESIGN & COMPUTATIONAL MODEL

- **All system tasks**
  - Their activation signal (external signal/event, processor clock)
  - Restrictions on what they can and cannot do

- **Task communication means** (shared memory, mailbox/message queues, signals, rendez-vous)

- **Handled and non-handled interrupts** (frequency, priority, resources)

- **Scheduling type** (cyclic executive, fixed-priority preemptive)

- **Task synchronisation mechanisms** (including mutual exclusion)

- **Resource protection mechanisms**

- **Inter-node communication and distribution**

- **Weak computational model** → complicated schedulability analysis

- **Design not detailed enough** → complicated schedulability analysis
SCHEDULABILITY ANALYSIS

- **Analysis method**
  - Rate-Monotonic Analysis (with blocking)
  - Response-time analysis
  - Proprietary (offsets, adjusted multi-frame model)

- **Task table with all details**
  - Periodic/sporadic, WCET, period, priority, blocking time, deadline

- **All shared resources**
  - Semaphores, usage, critical sections, priority inheritance

- **Interrupt handlers**

- **System overheads**
  - Preemption, interrupt latency, access to semaphores, interrupt locks, message queues, etc.

- **Use of operational scenarios**
  (i.e. realistic TM/TC traffic, operations)
## Schedulability Analysis

<table>
<thead>
<tr>
<th>Task Id (Name)</th>
<th>Subsystem</th>
<th>Priority</th>
<th>Type</th>
<th>Frequency</th>
<th>Period</th>
<th>Deadline</th>
<th>WCET</th>
<th>Max Blocking</th>
<th>Max Interference</th>
<th>Utilisation</th>
<th>Response Time</th>
<th>Deadline Margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Management</td>
<td>System Control</td>
<td>10</td>
<td>Periodic</td>
<td>100</td>
<td>10</td>
<td>10</td>
<td>0.44</td>
<td>0.090</td>
<td>0.000</td>
<td>4.40%</td>
<td>0.53</td>
<td>94.70%</td>
</tr>
<tr>
<td>1553 Bus Handler</td>
<td>BSW</td>
<td>20</td>
<td>Periodic</td>
<td>50</td>
<td>20</td>
<td>20</td>
<td>4.70</td>
<td>1.120</td>
<td>0.440</td>
<td>2.66%</td>
<td>6.26</td>
<td>68.70%</td>
</tr>
<tr>
<td>AOCS Main Loop</td>
<td>AOCS</td>
<td>25</td>
<td>Periodic</td>
<td>10</td>
<td>100</td>
<td>40</td>
<td>9.70</td>
<td>0.260</td>
<td>27.900</td>
<td>9.70%</td>
<td>37.86</td>
<td>5.35%</td>
</tr>
<tr>
<td>TC Handler</td>
<td>Data Handling</td>
<td>61</td>
<td>Sporadic</td>
<td>5</td>
<td>200</td>
<td>100</td>
<td>6.22</td>
<td>1.120</td>
<td>47.300</td>
<td>3.11%</td>
<td>54.64</td>
<td>45.36%</td>
</tr>
<tr>
<td>Thermal Control</td>
<td>System Control</td>
<td>66</td>
<td>Periodic</td>
<td>5</td>
<td>200</td>
<td>200</td>
<td>11.25</td>
<td>0.850</td>
<td>53.520</td>
<td>5.63%</td>
<td>65.62</td>
<td>67.19%</td>
</tr>
<tr>
<td>OBCP</td>
<td>Data Handling</td>
<td>67</td>
<td>Periodic</td>
<td>2</td>
<td>500</td>
<td>200</td>
<td>68.40</td>
<td>0.850</td>
<td>78.020</td>
<td>13.68%</td>
<td>147.27</td>
<td>26.37%</td>
</tr>
<tr>
<td>Housekeeping</td>
<td>Data Handling</td>
<td>68</td>
<td>Periodic</td>
<td>1</td>
<td>1000</td>
<td>400</td>
<td>40.40</td>
<td>0.850</td>
<td>31.820</td>
<td>4.04%</td>
<td>73.07</td>
<td>81.73%</td>
</tr>
<tr>
<td>MTL</td>
<td>Data Handling</td>
<td>91</td>
<td>Sporadic</td>
<td>1</td>
<td>1000</td>
<td>400</td>
<td>24.30</td>
<td>0.160</td>
<td>271.120</td>
<td>2.43%</td>
<td>295.58</td>
<td>26.11%</td>
</tr>
<tr>
<td>System Log</td>
<td>Data Handling</td>
<td>97</td>
<td>Sporadic</td>
<td>1</td>
<td>1000</td>
<td>1000</td>
<td>126.90</td>
<td>0.160</td>
<td>569.020</td>
<td>12.69%</td>
<td>696.08</td>
<td>30.39%</td>
</tr>
<tr>
<td>Scrubbing</td>
<td>System Control</td>
<td>98</td>
<td>Periodic</td>
<td>1</td>
<td>1000</td>
<td>1000</td>
<td>114.00</td>
<td>0.000</td>
<td>744.000</td>
<td>11.40%</td>
<td>858.00</td>
<td>14.20%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>69.73%</td>
</tr>
</tbody>
</table>
CURRENT CHALLENGES

- **Multiple sources of pessimism in schedulability analysis**
  - Requirements (e.g. telecommand upload rate)
  - Architectural and detailed design (lack of detail)
    - All tasks are considered hard real-time
  - Analysis method (for preemptive systems)
    - Combining all worst-case scenarios in a single scenario with probability to occur close or equal to zero
    - Example: Period of sporadic task is set as minimum of its inter-arrival time
    - Example: Blocking time on a set of semaphores assumes that they are all acquired
      - Not if priority ceiling is used
      - Not if more subtle analysis of resource usage is performed

- **Typical requirement is 25% margin on timing**
  - However we often see 20-25% nominal CPU load on satellites
FURTHER CHALLENGES

- **Observation**
  - Preemptive scheduling allows you not to concern about timing when designing your flight software
  - This is not possible with cyclic scheduling (where timing is defined for your task from the beginning)
  - This is fine, as long as you end up passing the schedulability test ... but what happens if you don’t?

- **Hardware not so fast comparing to other domains**
  - ERC32 (SPARC v7) clocked 20 MHz
  - LEON2 (SPARC v8) clocked 80-120 MHz
  - LEON2 and LEON3 with cache

- **Tools for schedulability analysis and WCET measurements**
  - No unified set of tools for different ESA projects
  - Often spreadsheet based approach
CHALLENGES IN THE NEAR FUTURE
Cache is good to speed-up in average, at the cost of more variable execution time

- Less predictability
- Difficult to analyse
- Aggravated worst-case execution path at the level if microinstructions (not in LEON2/SPARC-V8 RISC architecture)

Impact of architecture & design on the use of cache

- Small changes to memory map could have impact on cache miss ratio and consequently on timing

Possible solutions

- Cache-aware schedulability
- Cache locking
- Cache partitioning
- Cache-aware coding style
- Cache-aware linker
Multi-cores offer better performance per watt than single-core processors
- Expected technology trend also in time-critical systems

Classical approach is not efficient for multi-core (WCET per task, fixed priorities)
- Multiple tasks execute at the same time (one per core)
- WCET harder to analyse due to inter-task interferences accessing shared resources
  - Arbitration mechanism
  - WCET depends on workload!
- For two or more processors, no deadline scheduling algorithm can be optimal without complete a priori knowledge of deadlines, computation times and process start times

Dynamic priority scheduling theory is regarded as having potential advantages
- Higher CPU utilisation
- Separation between truly hard real-time tasks (missing a deadline is not acceptable) and soft real-time tasks
INTEGRATED MODULAR AVIONICS

- Logical partitions with strong spatial and temporal isolation
- Inter-partition Communication (IPC) mechanism respects space partitioning and real-time determinism
  - Static scheduling of communication partition
- Reduced integration effort
  - Modular verification
- Co-hosting applications with different criticality levels
- Partitioned design is a good way to migrate to a multi-core system
  - Task parallelism better suited than data parallelism for data processing on multicore
- Scheduling policy could be chosen per core
  - Fixed-priority preemptive scheduling for hard-real-time
  - Dynamic scheduling for soft-real-time and event-driven tasks
    - Example: IRQ handlers execution, as I/O is mostly non-deterministic
    - Example: FDIR mechanisms
  - E.g. 3 cores using fixed cyclic scheduling, 1 core using dynamic scheduling
Ensuring Schedulability of Spacecraft Flight Software | 7-9 November 2012 | Slide 16

MODEL-DRIVEN SOFTWARE ENGINEERING

- Schedulability analysis should be embedded in the model-based development
- New programming model: Non-functional properties used to define timing and concurrency control
  - Non-functional properties related to timing (control flow, timing, deadlines, communication budgets, etc.) and concurrency (reentrancy) must become an integral part of a software component description

- ASSERT/TASTE
  - TASTE is a set of tools dedicated to the development of embedded, real-time systems, developed by ESA
  - Allows to easily integrate heterogeneous pieces of code produced either manually or automatically by external modeling tools
  - Provide facilities for automatic schedulability analysis (connection with CHEDDAR schedulability analysis tool)

- Challenges
  - No experience from real projects
  - Programming model is restricted by a tool
  - Difficult to apply legacy systems
CONCLUSIONS

- **ESA approach to schedulability analysis**
  - Overview of FSW development lifecycle
  - Current challenges

- **(Near) future challenges**
  - Cache, multi-core, IMA
  - Model-based FSW engineering

- **Scheduling on multi-cores is still a research topic**
  - IMA could make it easier to control

- **Model-driven engineering**
  - From schedulability point of view still an open issue

- **ESA is trying to unify the current approach as well as look into the future**
  - Research studies
    - WCET for LEON with cache
    - Schedulability for Integrated Modular Avionics
    - Schedulability for multi-core processors
    - Model-based engineering
THANK YOU

Marek.Prochazka@esa.int
Jorge.Lopez.Trescastro@esa.int
European Space Agency