CYCLIC EXECUTIVE FOR SAFETY-CRITICAL JAVA ON CHIP-MULTIPROCESSORS

Anders P. Ravn (AAU)
Martin Schoeberl (DTU)
Cyclic executive

Advantages and disadvantages

Cyclic executive on a CMP

Schedule generation

Implementation for SCJ

Summary
Cyclic Executive

- Static schedule of tasks
- No preemption
- Organized in minor and major cycles
- Used in safety-critical applications
Advantages

- Deterministic schedule
- Easy communication & precedents constraints
- Simple implementation
- Simple context switch
- Fewer context switches
- WCET friendly
Disadvantages

- Constraints on task periods
  - Long running tasks need to be split
- Deadline miss influences all tasks
WCET ANALYSIS

- Considers individual tasks
- Scheduling effects are usually ignored
  - Cost of preemption and dispatch
  - Cost of scheduling
  - Cache trashing due to a task switch
- Analysis works well for CE
Three levels:

- L0 cyclic executive
- L1 preemptive, static schedule, single mission
- L2 nested missions
- CMP considered only for L1 and L2
Keep the CE advantages and relax constraints

- Long running tasks can have their own CPU
- Schedules are synchronous on the cores
- Tasks are allowed to migrate
- Cheap on chip-multiprocessors
Two processors

Three tasks

Scheduleable only with migration of task A

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task A</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Task B</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Task C</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>B</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core 2</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>A</td>
</tr>
</tbody>
</table>
Easy on uniprocessor CE

Options for CMP CE
- Locks with blocking (spin lock)
- Precedence constraints in the schedule
- Non-blocking queues between tasks
**Precedence Constr.**

- Tasks that share a resource
  - One writes, another reads or writes
  - Not allowed to run in parallel
- Part of the schedule generation
- More schedule flexibility with simple task model: read - execute - write
Schedule generation is NP-complete

- We use model checking (UppAal)
- No restrictions on minor frames
- Each task represented by one automaton
- Check the tasks until global $t > SCM(T_i)$
- Results in one possible schedule or failure
One automaton per task

Parameterized with

- T, C, and D

Number of processors: p

Local time: t

Local execution time: r
IMPLEMENTATION

- Prerequisites
  - Common passive fine grain clock
  - No interrupts needed
- Schedules are synchronous
- Schedule is a simple table of slot times and Runnables
Implementation

- Use a Java CMP (JOP)
- Each core has its own clock
- Scheduler in Java - just a few lines of code
- Deadline overrun can be queried
DEPARTURE FROM SCJ

• CMP at Level 0 ;-) 
• Runnable instead of BAEH 
• Task migration is allowed 
• One Runnable per Frame 
• Overrun detection (late) 
• getCurrentProcessor()
SUMMARY

- Cyclic executive on a CMP combines
  - deterministic schedule with
  - more processing power
- Model checking used for schedule generation
- Runtime implementation is very simple