A Dedicated Micro-Kernel to Combine Real-Time and Stream Applications on Embedded Manycores

P. Dubrulle, E. Ohayon
CEA LIST
Increasing number of cores in Chip Multi-Processors (CMP)
- Hundreds of cores in emerging architectures (from *multicores* to *manycores*)
- Massive parallelism required in applications

Including manycores in design of safety critical embedded systems
- Using adapted programming models to handle
  - massive parallelism
  - real-time constraints
  - execution determinism
- Mixing independent legacy applications
Models for parallel programs
- Frameworks (e.g. MPI, OpenMP)
- Dataflow (e.g. OpenCL)
  - Stream programming (e.g. StreamIt, ΣC)

Stream programming answers programmability issues of multicores and manycores
- No race condition
- Offline detection of deadlocks
- Amenable to bounded-memory execution
Models for real-time programs

- Event-Triggered
  - Highest reactivity + Intuitive programming model

- Time-Triggered
  - Determinism, Testability & Extensibility
  - Well suited for safety-critical applications
  - Example: Time Constrained Automata – implementation language = ΨC

Time Constrained Automata gives possibility to express multi-task real-time programs and provides

- Hard Real-Time consistency
  - By design: no deadlock or race condition

- Offline Schedulability Analysis

- Global communication determinism → Reproducibility

- Offline Sizing of communication buffers
What execution support for a mix of hard real time and massively parallel applications on a manycore?

- Safety
- Performance
- Power efficiency
- Good integration
State of the art

- Allocation of a subset of cores to critical tasks (shielding)
  - Decreases the response time for critical tasks
  - Wastes execution resources

- Compositional/hierarchical scheduling
  - Allows using all resources whenever available
  - Has a significant system overhead
  - Forces a strict separation

- Virtualization with hypervisor
  - Increases safety by strict separation
  - Wastes execution resources
  - Has low performance efficiency
Proposed approach

- A single dedicated micro-kernel with flexible shielding for two models
  - Best-effort, based on stream programming using $\Sigma C$
  - Real-time, based on time constrained automata using $\Psi C$

- Bare metal execution support provides
  - complete control over hardware for safety (interrupts, timers)
  - computing performance
  - accurate power management

- Shared memory targets allow for dynamic scheduling
  - preemption
  - migration
Real-time task ($\Psi$)

- Time triggered
- Performs jobs within a time window
  - Earliest start date
  - Latest end date (i.e. deadline)
- Has a priority $p$ equal to the deadline, the less $p$ is, the higher the priority

Best effort task ($\Sigma$)

- Data driven
- Performs jobs on a dataflow
  - Input tokens are present in input channels
  - Output channels have enough space to hold output tokens
- Has an infinite priority $p$
Partitioning execution cores

1 Supervision Element (SE)
- Task scheduling
- Hardware supervision

N Processing Elements (PE)
- Task scheduling (minimal)
- Basic IPC
- User tasks

Flexible shielding
- PE are partitioned in two sets
  - RT: can run any task ($\Psi$ or $\Sigma$), no binding
  - NRT: can only run best effort ($\Sigma$)
- Keeps good response time without wasting resources
Scheduling

SE

ψ

Σ

Scheduling

PE

Scheduling (RT)

IPC(Σ)

IPC(Ψ)

PE

Scheduling (NRT)

IPC(Σ)

Task

Preemption interrupt

ready

ended
About task allocation

- The SE is in charge of task allocation
  - Wake idle NRT PE to execute a $\Sigma$ task
  - Wake idle RT PE to execute either $\Psi$ or $\Sigma$ task (according to priority)
  - Preempt RT PE to execute $\Psi$ task (according to priority)

- Weak synchronization
  - Fast operations and avoids contention
  - May cause useless wake up/preemption
    - For wake up, wastes a little energy
    - For preemption, wastes a few cycles
Early benchmarking on a first prototype

- Platform: dual processor Intel Xeon 2.53GHz quad core (i.e. 8 SMP cores)
- Real-time application: PID autopilot
  - 6 communicating tasks
    - 4 tasks with 5 ms period (I&C loop for autopilot)
    - 2 tasks with 10 ms and 50 ms period (I/O for display and manual command)
- Best-effort application: edge detection
  - 16 stream tasks
- Partitioning: 1 SE, from 1 up to 7 RT-PE, 0 NRT-PE
Evaluation

Average CPU load on a 5 min period, depending on count of active PE

<table>
<thead>
<tr>
<th>Active PE</th>
<th>Average PE load</th>
<th>SE load</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>70</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>60</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>40</td>
<td>0</td>
</tr>
</tbody>
</table>
Evaluation

<table>
<thead>
<tr>
<th>Active PE</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. latency (µs)</td>
<td>1.8</td>
<td>1.8</td>
<td>1.4</td>
<td>1.4</td>
<td>1.4</td>
<td>1.7</td>
<td>1.4</td>
</tr>
<tr>
<td>Avg. latency (µs)</td>
<td>1.8</td>
<td>1.9</td>
<td>2.0</td>
<td>2.3</td>
<td>2.4</td>
<td>2.5</td>
<td>2.6</td>
</tr>
<tr>
<td>Max. latency (µs)</td>
<td>4.0</td>
<td>4.7</td>
<td>4.9</td>
<td>5.3</td>
<td>5.8</td>
<td>6.0</td>
<td>6.9</td>
</tr>
</tbody>
</table>

Real time latency on the SE (time in µs between acknowledging timer interrupt and releasing the task in the ready list – 10000 samples)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inter-processor interrupt</td>
<td>0.9</td>
</tr>
<tr>
<td>Send instruction on SE – first handler instruction on PE</td>
<td></td>
</tr>
<tr>
<td>Full context switch</td>
<td>1.1</td>
</tr>
<tr>
<td>Stack, ABI, no FPU</td>
<td></td>
</tr>
<tr>
<td>RT latency (wake up)</td>
<td>3.2</td>
</tr>
<tr>
<td>Timer interrupt on SE – first user instruction on PE</td>
<td></td>
</tr>
<tr>
<td>RT latency (preempt)</td>
<td>4.0</td>
</tr>
<tr>
<td>Timer interrupt on SE – first user instruction on PE</td>
<td></td>
</tr>
<tr>
<td>End of job syscall</td>
<td>1.9</td>
</tr>
<tr>
<td>Syscall instruction – after picking next task if any</td>
<td></td>
</tr>
</tbody>
</table>

Average duration of micro-kernel operations (7 PE, 10000 samples)
Conclusion and perspectives

- A single micro-kernel as execution support to mix safety critical real-time tasks and compute intensive applications on a manycore
  - Using appropriate execution models, the system has essential safety properties
  - Using a flexible shielding technique, the system shows good performance (low latencies)
  - Using hardware mechanism for power saving allows to put idle cores to sleep whenever needed

- Future work
  - Benchmark on an embedded manycore (Kalray MPPA), use WCET analysis to infer latency properties
  - Implement automatic memory protection to improve safety
  - Further enhance cohabitation by providing communication between tasks from different models