A study of predictable execution models implementation for industrial data-flow applications on a multi-core platform with shared banked memory

M. Schuh\textsuperscript{1,3}, C. Maiza\textsuperscript{1}, J. Goossens\textsuperscript{2}, P. Raymond\textsuperscript{1}, B. D. Dinechin\textsuperscript{3}

\textsuperscript{1}Univ. Grenoble Alpes – Verimag

\textsuperscript{2}Univ. Libre Bruxelles – Faculté des Sciences

\textsuperscript{3}Kalray S.A.

Real-Time Systems Symposium, December 1 – 4, 2020
Context

Overview

• Correct by construction time-predictable programs
• Data-flow applications, e.g. SCADE
• On multi-core platforms ⇒ interference

Our approach

• Global static scheduling
• Time-triggered
• Non-preemptive execution
• Kalray MPPA2 processor
Contributions

1. Technique to implement SCADE applications inspired by PREMs
   - Several variants: number of phases and memory partitioning
   - 3 new global scheduling algorithms for isolation

2. Comparison of execution models
   - **2-Phased** behaves the best: 32% decrease
   - Bounding interference is better than isolation: 26% decrease

3. Guidelines for predictable implementation of data-flow applications
Data-flow application

- Programs are oriented graphs
  - Nodes are computational units
  - Directed wires represent communication and dependencies
- High-level languages for critical real-time systems
  - Majority of avionics control code
  - ISO 26262 ASIL C and DO-178C certification

Parallel code generation

- SCADE Multi-Core Code Generator (MCG)
- Nodes are compiled into classical sequential tasks
- Communication channels with intrinsic REW model
Kalray MPPA2 processor

- Independently arbitrated memory banks
- Timing compositional architecture
- Timers and performance counters
## Context

**Introduction and State of the Art — Predictable Execution Models**

### Execution Partitioning

- 2-Phased: Execute-Write (EW) model
- 3-Phased: Read-Execute-Write (REW) model
- Memory-Centric 3-Phased: REW model with a manager core

### Memory Partitioning

- No interference
- Analyzed interference
Implementation reference application

Simple Data-Flow

- 5 nodes, 2 cores
- Node ≈ Task
- Each task 2 or 3 phases
Studied models

Execution Partitioning

2-phased: Local Execute, Remote Write

3-phased: Local Execute, Shared Remote Read/Write

Memory Centric 3-phased: Local Execute, Shared Master Read/Write
2-phased EW model

Memory Partitioning

Interference

Isolation

Schuh, Maiza, Goossens, Raymond, Dinechin

Study of PREMs on multi-core with banked memory

RTSS 2020
3-phased REW model with Shared Bank

Memory Partitioning

Interference

Isolation
Memory Centric 3-phased with Master Core
Memory Partitioning

Schuh, Maiza, Goossens, Raymond, Dinechin
Study of PREMs on multi-core with banked memory
RTSS 2020 11/22
Multi-Core Interference Analysis (MIA)

https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/synchrone/mia

- Free and Open Source tool that performs interference analysis
- Model of the target platform memory system: MPPA2
Multi-Core Interference Analysis (MIA)

https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/synchrone/mia

- Free and Open Source tool that performs interference analysis
- Model of the target platform memory system: MPPA2

**Input**

- Worst-Case Execution Time (WCET)
- Worst-Case Number of Accesses (WCA)
Multi-Core Interference Analysis (MIA)

https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/synchrone/mia

- Free and Open Source tool that performs interference analysis
- Model of the target platform memory system: MPPA2

**Input**

- Worst-Case Execution Time (WCET)
- Worst-Case Number of Accesses (WCA)

**Output**

- Worst-Case Response Time (WCRT)
- Release dates

Schuh, Maiza, Goossens, Raymond, Dinechin

Study of PREMs on multi-core with banked memory

RTSS 2020 12 / 22
Multi-Core Interference Analysis (MIA)

https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/synchrone/mia

- Free and Open Source tool that performs interference analysis
- Model of the target platform memory system: MPPA2

**Input**
- Worst-Case Execution Time (WCET)
- Worst-Case Number of Accesses (WCA)

**Output**
- Worst-Case Response Time (WCRT)
- Release dates

**Detail**
- Isolation is enforced by additional dependencies
Our implementation

Workflow

**Step 1: Memory Phases Generation**
- Code
- Memory Phases
- Initial DFG
- Memory Transaction Definition
- Mapping + Scheduling
- MCA

**Step 2: Memory Mapping and Global Scheduling (isolation)**
- Code
- Memory Phases
- Timing Analyser
- WCET
- WCA
- Memory Partitioning Model

**Step 3: Orchestration Code Generation**
- Annotated DFG
- Memory Mapping
- Global Scheduling
- MIA
- Release dates

- Our tool
- External tool
- Execution model
- Intermediate data
- Code
- Used by
- Produces
Global Scheduling Algorithms

**Algorithm 1: 3-Phased Contiguous**

- Memory phases are considered a contiguous entity
- Immediately schedules the remainder of a write phase
- If possible, schedules the mirror read transaction
Algorithm 1: 3-Phased Contiguous

- Memory phases are considered a contiguous entity
- Immediately schedules the remainder of a write phase
- If possible, schedules the mirror read transaction
Global Scheduling Algorithms

Algorithm 1: 3-Phased Contiguous

- Memory phases are considered a contiguous entity
- Immediately schedules the remainder of a write phase
- If possible, schedules the mirror read transaction

Core 0
N0
N1
N2
N3
N4
Core 1

Core 0
N0_execute
N0_write_N1
Core 1

N2_read_N0
N0_write_N2
N1_read_N0
Algo 1
Global Scheduling Algorithms

Algorithm 1: 3-Phased Contiguous

- Memory phases are considered a contiguous entity
- Immediately schedules the remainder of a write phase
- If possible, schedules the mirror read transaction

```
N0
Core 0

N1

N2

N3

Core 1

N4

Core 1

N0_execute

N0_write_N1

N0_write_N2

N2_read_N0

N1_read_N0

Algo 1

Core 0

Core 1
```
Global Scheduling Algorithms

Algorithm 1: 3-Phased Contiguous

- Memory phases are considered a contiguous entity
- Immediately schedules the remainder of a write phase
- If possible, schedules the mirror read transaction

![Diagram showing the scheduling process](image_url)
Global Scheduling Algorithms

*Algorithm 2: 3-Phased Optimized*

- Removes the contiguous schedule constraint
- Possibility of idle time between transactions of same task
- Unblocks execute phases earlier
Global Scheduling Algorithms

Algorithm 2: 3-Phased Optimized

- Removes the contiguous schedule constraint
- Possibility of idle time between transactions of same task
- Unblocks execute phases earlier
Global Scheduling Algorithms

Algorithm 2: 3-Phased Optimized

- Removes the contiguous schedule constraint
- Possibility of idle time between transactions of same task
- Unblocks execute phases earlier
Global Scheduling Algorithms

Algorithm 2: 3-Phased Optimized

- Removes the contiguous schedule constraint
- Possibility of idle time between transactions of same task
- Unblock execute phases earlier
Global Scheduling Algorithms

Algorithm 2: 3-Phased Optimized

- Removes the contiguous schedule constraint
- Possibility of idle time between transactions of same task
- Unblock execute phases earlier
Global Scheduling Algorithms

**Algorithm 3: Memory-Centric**

- Memory transactions are mapped to a different core
- Thus they can be arranged in any order
- As long as the dependencies are satisfied
Global Scheduling Algorithms

Algorithm 3: Memory-Centric

- Memory transactions are mapped to a different core
- Thus they can be arranged in any order
- As long as the dependencies are satisfied
Global Scheduling Algorithms

Algorithm 3: Memory-Centric

- Memory transactions are mapped to a different core
- Thus they can be arranged in any order
- As long as the dependencies are satisfied
Global Scheduling Algorithms

Algorithm 3: Memory-Centric

- Memory transactions are mapped to a different core
- Thus they can be arranged in any order
- As long as the dependencies are satisfied
Global Scheduling Algorithms

**Algorithm 3: Memory-Centric**

- Memory transactions are mapped to a different core
- Thus they can be arranged in any order
- As long as the dependencies are satisfied
Global Scheduling Algorithms

**Algorithm 3: Memory-Centric**

- Memory transactions are mapped to a different core
- Thus they can be arranged in any order
- As long as the dependencies are satisfied
Results
Case-Studies

Simple Data-Flow
• Example application: 5 nodes, mapped to 2 cores
• Computation-Communication Ratio (CCR): 1732 – 3041 cycles

ROSACE
• Avionics ONERA case-study: 10 nodes, mapped to 8 cores
• CCR: 260214753 – 5972 cycles

Automotive ECU
• Industrial automotive program: 9 nodes, mapped to 6 cores
• CCR: 12548 – 5416 cycles
Results

Simple Data-Flow Results

Interference

Isolation

10^3 Processor Clock Cycles

<table>
<thead>
<tr>
<th></th>
<th>Measured</th>
<th>WCRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-Phased</td>
<td>2484</td>
<td>2608</td>
</tr>
<tr>
<td>3-Phased</td>
<td>3566</td>
<td>3729</td>
</tr>
<tr>
<td>2-Phased</td>
<td>3144</td>
<td>3264</td>
</tr>
<tr>
<td>3-Phased (Cont.)</td>
<td>4919</td>
<td>5057</td>
</tr>
<tr>
<td>3-Phased (Opt.)</td>
<td>3962</td>
<td>4244</td>
</tr>
<tr>
<td>Memory Centric</td>
<td>4106</td>
<td>4229</td>
</tr>
</tbody>
</table>

Schuh, Maiza, Goossens, Raymond, Dinechin
Study of PREMs on multi-core with banked memory

RTSS 2020 18/22
### Results

**ROSACE and Automotive Results**

- Same tendency on other applications
- 3-Phased Opt and Memory-Centric are incomparable
  - Depends on the application

#### ROSACE Results

<table>
<thead>
<tr>
<th>Interference</th>
<th>Isolation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>2-Phased</strong></td>
<td>3-Phased (Cont.)</td>
</tr>
<tr>
<td>Measured</td>
<td>Measured</td>
</tr>
<tr>
<td>10^6 Processor Clock Cycles</td>
<td></td>
</tr>
</tbody>
</table>

#### Automotive Results

<table>
<thead>
<tr>
<th>Interference</th>
<th>Isolation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>2-Phased</strong></td>
<td>3-Phased (Cont.)</td>
</tr>
<tr>
<td>Measured</td>
<td>Measured</td>
</tr>
<tr>
<td>10^3 Processor Clock Cycles</td>
<td></td>
</tr>
</tbody>
</table>
Discussion
Open questions

• Methodology based on *timing compositionality*

• *3-Phased* model *commonly* used in real-time community

• *2-Phased* model is more *efficient* and easy to implement in multi-banked shared memory architectures

• *Low* potential of *interference* in our case-studies
Discussion

Key takeaways

1. Taking interference into account leads to shorter execution time

2. 2-Phased implementation is always more efficient than 3-Phased

3. Our optimized algorithm yields the best results in most cases
Thanks for your attention!

Questions?

You can find me at
matheus.schuh@univ-grenoble-alpes.fr
http://www-verimag.imag.fr/~schuhm/