Parallel Code Generation of Synchronous Programs for a Many-core Architecture

Amaury Graillat

Supervisors: Pascal Raymond (Verimag), Matthieu Moy (LIP)
Benoît Dupont de Dinechin (Kalray)

Reviewers: Jan Reineke (Universität des Saarlandes)
Robert de Simone (INRIA Sophia-Antipolis, Aoste)

Examiners: Anne Bouillard (ENS Paris)
Alain Girault (INRIA Grenoble)
Christine Rochange (IRIT)

Thesis defense, November 16th, 2018
**SPACE, WEIGHT, AND POWER**

**Single-Core**
Available since 1971
Aircraft: since 1980s for digital fly-by-wire system

**Multi-Core**
Available since 2001

**Many-Core**
Available since 2007

- End of production
  - No longer sufficient
- More functions in one chip
  - More energy efficient
SPACE, WEIGHT, AND POWER

Single-Core
Available since 1971
Aircraft: since 1980s for digital fly-by-wire system
Well-proven for real-time
End of production
No longer sufficient

Multi-Core
Available since 2001

Many-Core
Available since 2007
More functions in one chip
More energy efficient
SAFETY CRITICAL SYSTEMS

Example of aircraft flight controller (control-command)

- Time-critical: latency constraints are part of the specification (e.g. < some 10ms)
- Designed with eg., Synchronous Languages
THE SYNCHRONOUS DATA-FLOW LANGUAGES (1/2)

- Lustre (academic), Scade (industrial)
- Network of nodes
**The Synchronous Data-Flow Languages (2/2)**

**Logical**

- $i_{n-1} = 1$
- $i_n = 3$
- $i_{n+1} = 7$

- $o_{n-1} = 2$
- $o_n = 6$
- $o_{n+1} = 14$

- One execution is called a logical instant
THE SYNCHRONOUS DATA-FLOW LANGUAGES (2/2)

Logical

\[ i_{n-1} = 1 \quad i_n = 3 \quad i_{n+1} = 7 \]

\[ o_{n-1} = 2 \quad o_n = 6 \quad o_{n+1} = 14 \]

▶ One execution is called a logical instant

Physical

\[ i_{n-1} \quad i_n \quad i_{n+1} \]

\[ o_{n-1} \quad o_n \quad o_{n+1} \]

▶ Requirement: \( o_n \) before \( i_{n+1} \).
▶ Worst-Case Execution Time (WCET)
LUSTRE/SCADE Delay Operator

- previous operator
- Logical/functional delay
CODE GENERATION FOR SINGLE-CORE

Node 1 -> Node 2 -> Node 3

Formal semantics
Determinism

Node 1
Node 2
Node 3
Single-core processor

Code Generation
Compilation

for each period {
    i = sensors();
    o = main_step(i);
    actuators(o);
}

void main_step(i) {
    o1 = N1_step(i);
    o2 = N2_step(i);
    returns N3_step(o1, o2);
}

Static schedule

Synchronous languages
- Lustre, Heptagon, SCADE
- Industrial use: SCADE in Airbus A380

WCET

OK for sequential execution. What about parallel?
**CODE GENERATION FOR SINGLE-CORE**

![Diagram showing the flow of code generation for single-core processors](image)

**Typical Code Flow for a Single-Core Processor**

```c
for each period {
    i = sensors();
    o = main_step(i);
    actuators(o);
}
```

**Static schedule**

```c
void main_step(i) {
    o1 = N1_step(i);
    o2 = N2_step(i);
    returns N3_step(o1, o2);
}
```

**Code Generation**

- Node 1
- Node 2
- Node 3

**Formal semantics**

- Determinism

**Conclusion**

OK for sequential execution. What about parallel?
Properties of the Kalray MPPA2:

- **Cores**
  - No complex branch prediction
  - Only LRU caches

- **Cluster**
  - Banked Shared-Memory (16*128ko)
  - Independent arbiter for each memory bank.

- **Network-on-Chip (NoC) between clusters**
  - Bandwidth limiter (Network calculus possible)

Performance + Predictability
OVERVIEW

Lustre/Scade

Kalray MPPA2 Bostan

Our work

Automatic code generation

Semantics Preservation
Traceability

Temporal Guaranties
Part I: Semantics Preserving Parallelization

- Extraction of Parallelism
- Mapping / Scheduling
- Code Generation
- Communication Channels Implementation

Part II: Real-Time Guarantees

Part III: Evaluation and Conclusion
**EXTRACTION OF PARALLELISM**

Method 1:
- In the top-level node:
  - 1 node = 1 task (Runnable).
- Generation of sequential code for each node
- Similar to Architecture Description Language (Prelude, Giotto)

Method 2: based on fork-join:
- see manuscript
EXTRACTION OF PARALLELISM

Method 1:
- In the top-level node:
  - 1 node = 1 task (runnable).
- Generation of sequential code for each node
- Similar to Architecture Description Language (Prelude, Giotto)

Method 2: based on fork-join:
- see manuscript
MAPPING / SCHEDULING

- Need non-preemptive static schedule
- External scheduling tool
- We can easily check the schedule
**Mapping/Scheduling: Example**

- **Core 0**: N1 ; N4 ; N6
- **Core 1**: N2 ; N5
- **Core 2**: N3

Schedule checked using the dependency graph
CODE GENERATION

Parallelism Extraction

Mapping/Scheduling

Communication

Communication and Dependency graph

Code Generation

System + Communication

NoC Routing

Executable for Kalray
Sketch of code for core 0.

```cpp
for each period{
    wait_inputs_N1(); // N1
    N1_step();
    write_outputs_N1();

    wait_inputs_N4(); // N4
    N4_step();
    write_outputs_N4();

    wait_inputs_N6(); // N6
    N6_step();
    write_outputs_N6();
}
```
COMMUNICATION CHANNELS

- Contribution
- External tool

Two kinds of communications:
- **instantaneous** (→)
- **delayed** (→)
INSTANTANEOUS COMMUNICATION

void write_outputs_N2() {
    copy(N4.in2, N2.out);
    copy(N5.in1, N2.out);
}

void wait_for_inputs_N4() {

    /* Efficient hardware synchronization */
    /* Software-based cache coherency */
}
INSTANTANEOUS COMMUNICATION

```
void write_outputs_N2() {
    copy(N4.in2, N2.out);
    channel_N2_N4 = true;
    // + cache management
    notify(core_N4);
    copy(N5.in1, N2.out);
}
```

```
void wait_for_inputs_N4() {
    while(! channel_N2_N4) {
        wait();
    }
    channel_N2_N4 = false;
    while(! channel_N1_N4) {
        wait();
    }
    channel_N1_N4 = false;
    // + cache management
}
```

- Efficient hardware synchronization
- Software-based cache coherency
COMMUNICATION CHANNELS

Two kinds of communications:
- instantaneous (→)
- delayed (→)
**Delayed Communication**

Transformation into a **SWAP** + scheduling constraints.

**Scenario 1**

- **Constraint:** $B \rightarrow S$

**Scenario 2**

- **Constraint:** $A \rightarrow S$
**DELAYED COMMUNICATION**

Transformation into a **SWAP** + scheduling constraints.

**Scenario 1**
- Constraint: $B \rightarrow S$

**Scenario 2**
- Constraint: $A \rightarrow S$
**DELAYED COMMUNICATION**

Transformation into a SWAP + scheduling constraints.

**Scenario 1**

```
A.in ----------> S.in
```

Constraint: B → S

**Scenario 2**

```
A.in ----------> S.in
```

Constraint: A → S
CONCLUSION OF PART I

- Semantics preserved
- Next step: Temporal Guaranties
Part I: Semantics Preserving Parallelization

Part II: Real-Time Guarantees

► Shared Memory Interference
► Time-Triggered Execution Model
► Real-Time Guarantees with Network-on-Chip

Part III: Evaluation and Conclusion
Single-core: WCET is sufficient
Single-core: WCET is sufficient

Multi-core: WCET + interference on shared resources = Worst-Case Response Time (WCRT).

\[\text{WCET + interference too pessimistic in the general case} \Rightarrow \text{exploit the execution model in the analysis}\]
Private arbiter for each bank
Available in the Kalray MPPA2 (silicon)
**Banked Memory in Kalray MPPA2**

- Node 1
- Node 2
- Node 3
- Node 4
- Node 5
- Node 6

**Core 0**

- Bank 0 for core 0
  - N1
  - N4
  - N6

**Core 1**

- Bank 1 for Core 1
  - N2
  - N5

- Round Robin

▶ Choice: Code, input buffer, local variables are mapped in core’s bank of the core
▶ Interference on communication only
**Banked Memory in Kalray MPPA2**

- Choice: Code, input buffer, local variables are mapped in core’s bank of the core
- Interference on communication only
**Banked Memory in Kalray MPPA2**

- **Choice:** Code, input buffer, local variables are mapped in core’s bank of the core
- **Interference on communication only**
Banked Memory in Kalray MPPA2

- Node 1
- Node 2
- Node 3
- Node 4
- Node 5
- Node 6

- Core 0
  - Communication
  - Bank 0 for core 0
    - N1
    - N4
    - N6

- Core 1
  - Round Robin
  - Bank 1 for Core 1
    - N2
    - N5

Choice: Code, input buffer, local variables are mapped in core’s bank of the core

Interference on communication only

How to activate tasks?
**Problem with ASAP Execution**

Do tasks B and C interfere?

Faster execution of A $\Rightarrow$ interference between B and C.

Timing Anomaly

Solution: release dates for tasks (time-triggered)
PROBLEM WITH ASAP EXECUTION

Faster execution of A ⇒ interference between B and C.

Timing Anomaly
**Problem With ASAP Execution**

Faster execution of A ⇒ interference between B and C.

**Timing Anomaly**

Solution: release dates for tasks (time-triggered)
TIME-TRIGGERED EXECUTION MODEL

**Principle**

Compute a static release date when data is guaranteed to be available
Time-Triggered execution prevents tasks from starting earlier

**Implementation**

```c
void wait_inputs_N1() {
    while(time() < t_period + release_date_N1) {
        // wait
    }
}
```
**MULTI-CORE INTERFERENCE ANALYSIS**

Multi-Core Interference Analysis (MIA) Tool [Hamza Rihani, RTNS 2016]

www-verimag.imag.fr/Multi-core-interference-Analysis.html
Multi-Core Interference Analysis (MIA) Tool [Hamza Rihani, RTNS 2016]
www-verimag.imag.fr/Multi-core-interference-Analysis.html
FWAMEWORK

Parallelism Extraction

Communication and Dependency graph

Communication

Mapping + Non-preemptive scheduling

Code Generation

System + Communication

NoC Routing + Rate Attribution

Network Calculus (WCTT)

Executable for Kalray

WCET Analysis

release dates

MIA

▸ WCET Analysis (Otawa, AiT)

▸ Computation of the release dates (MIA tool)

▸ Insertion in the executable.

OUR EXECUTION MODEL

► Static scheduling
► Bare metal, no interrupt, no preemption
► Banked memory mapping (1 core → 1 bank)
► Time-triggered

Deterministic and WCRT guarantee
Overview

- Contribution
- External tool

Parallelism Extraction

Communication and Dependency graph

Mapping + Non-preemptive scheduling

Code Generation

System + Communication

Executable for Kalray

WCET Analysis

MIA

NoC Routing + Rate Attribution

Network Calculus (WCTT)

Release dates

WCET Analysis
The Kalray MPPA2’s NoC
THE Kalray MPPA2’s NoC

Cluster A
- Core 0
- MEM
- East, North, East, Local
- Rate limiter
- TX buffer

Cluster B
- Core 0
- MEM
- RX buffer
- DMA
The Kalray MPPA2’s NoC
THE KALRAY MPPA2’S NoC

Cluster A

Cluster B

EAST, NORTH, EAST, LOCAL

RATe LIMITER

HEAD ROUTEDATA

TX BUFFER

MEM

CORE 0

NORTH

EAST

LOCAL

FLOW 1

FLOW 2

RATE = $\frac{1}{2}$ B

RATE = $\frac{1}{2}$ B

HEAD ROUTE DATA

DMA

RX BUFFER

MEM

CORE 0
REAL-TIME GUARANTEES WITH NETWORK-ON-CHIP

1. Routing
Route = sequence of directions computed by sender

[Dupont de Dinechin B., Graillat A., NoCArc 2017]
REAL-TIME GUARANTEES WITH NETWORK-ON-CHIP

1. **Routing**
   Route = sequence of directions computed by sender

![Diagram of network-on-chip with clusters and routers]

[Dupont de Dinechin B., Graillat A., *NoCArc 2017*]
REAL-TIME GUARANTEES WITH NETWORK-ON-CHIP

1. **Routing**
   Route = sequence of directions computed by sender
   Deadlock-free algorithms (XY, Hamiltonian)

[Dupont de Dinechin B., Graillat A., NoCArc 2017]
1. **Routing**
   - Route = sequence of directions computed by sender
   - Deadlock-free algorithms (XY, Hamiltonian)
   - Algorithm with path diversity (Hamiltonian Odd Even (HOE))

[Dupont de Dinechin B., Graillat A., *NoCArc 2017*]
REAL-TIME GUARANTEES WITH NETWORK-ON-CHIP

1. Routing
   Route = sequence of directions computed by sender
   Deadlock-free algorithms (XY, Hamiltonian)
   Algorithm with path diversity (Hamiltonian Odd Even (HOE))

2. Route Selection
   Choose one static route per flow
   Optimize performance and fairness

[Dupont de Dinechin B., Graillat A., NoCArc 2017]
REAL-TIME GUARANTEES WITH NETWORK-ON-CHIP

1. Routing
   Route = sequence of directions computed by sender
   Deadlock-free algorithms (XY, Hamiltonian)
   Algorithm with path diversity (Hamiltonian Odd Even (HOE))

2. Route Selection
   Choose one static route per flow
   Optimize performance and fairness
   e.g. \{f1.1, f2.2\},

[Dupont de Dinechin B., Graillat A., NoCArc 2017]
REAL-TIME GUARANTEES WITH NETWORK-ON-CHIP

1. **Routing**
   - Route = sequence of directions computed by sender
   - Deadlock-free algorithms (XY, Hamiltonian)
   - Algorithm with path diversity (Hamiltonian Odd Even (HOE))

2. **Route Selection**
   - Choose one static route per flow
   - Optimize performance and fairness
   - e.g. \{f1.1, f2.2\}, \{f1.2, f2.2\}...

[Dupont de Dinechin B., Graillat A., NoCArc 2017]
REAL-TIME GUARANTEES WITH NETWORK-ON-CHIP

1. Routing
Route = sequence of directions computed by sender
Deadlock-free algorithms (XY, Hamiltonian)
Algorithm with path diversity (Hamiltonian Odd Even (HOE))

2. Route Selection
Choose one static route per flow
Optimize performance and fairness
e.g. \{f1.1, f2.2\}, \{f1.2, f2.2\}...

3. Rate attribution
Fair attribution
e.g. \{0.5, 0.5\},

[Dupont de Dinechin B., Graillat A., NoCArc 2017]
**REAL-TIME GUARANTEES WITH NETWORK-ON-CHIP**

1. **Routing**
   Route = sequence of directions computed by sender
   Deadlock-free algorithms (XY, Hamiltonian)
   Algorithm with path diversity (Hamiltonian Odd Even (HOE))

2. **Route Selection**
   Choose one static route per flow
   Optimize performance and fairness
   e.g. \{f1.1, f2.2\}, \{f1.2, f2.2\}...

3. **Rate attribution**
   Fair attribution
   e.g. \{0.5, 0.5\}, \{1.0, 1.0\}...

[Dupont de Dinechin B., Graillat A., *NoCArc 2017*]
REAL-TIME GUARANTEES WITH NETWORK-ON-CHIP

1. **Routing**
   Route = sequence of directions computed by sender
   Deadlock-free algorithms (XY, Hamiltonian)
   Algorithm with path diversity (Hamiltonian Odd Even (HOE))

2. **Route Selection**
   Choose one static route per flow
   Optimize performance and fairness
   e.g. \{f1.1, f2.2\}, \{f1.2, f2.2\}...

3. **Rate attribution**
   Fair attribution
   e.g. \{0.5, 0.5\}, \{1.0, 1.0\}...

4. **Network Calculus**
   Compute latency and buffer level

[Dupont de Dinechin B., Graillat A., *NoCArc 2017*]
1. **UNICAST ROUTING**

Hamiltonian Routing

- **Deadlock-free**
- **Path-Diversity**
1. **Multicast Routing Problem**

- Deadlock-free heuristic to “travelling salesman problem”:
  - One route? Tree is no possible with Kalray architecture
  - Hamiltonian: minimum of two routes

![Multicast Routing Diagram]

- Source
- Destination
1. **Multicast Routing Problem**

- Deadlock-free heuristic to “travelling salesman problem”:
- One route? Tree is no possible with Kalray architecture
- Hamiltonian: minimum of two routes

![Diagram of multicast routing problem with nodes and routes highlighted.

Source
Destination]
1. **Multicast Routing: Hamiltonian**

- Dual-Path + Hamiltonian without path diversity (Lin et al., 1992)
- Small path diversity, we can do better.
1. **Multicast Routing: HOE**

- Hamiltonian Odd Even (Bahrebar et al.) routing: allows some non-Hamiltonian paths

- Our choice: Dual-Path + HOE
EVALUATION OF DEADLOCK-FREE MULTICAST ROUTING

Minimum rate vs. Network load
(Higher is better)

- Dual Path+Hamiltonian
- Dual Path+HOE
Minimal rate increased of up to 19% with Dual Path HOE compared to Dual Path Hamiltonian.
2. ROUTE SELECTION

- Input: set of route for each flow
- Output: one route per flow
- Maximize fairness

- **Naive exploration**: Enumeration all combinations, run step 3 on each solution, keep the best.

<table>
<thead>
<tr>
<th></th>
<th>f2.1</th>
<th>f2.2</th>
<th>f2.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1.1</td>
<td>0.5, 0.5</td>
<td>0.5, 0.5</td>
<td>0.5, 0.5</td>
</tr>
<tr>
<td>f1.2</td>
<td>0.5, 0.5</td>
<td>1.0, 1.0</td>
<td>1.0, 1.0</td>
</tr>
<tr>
<td>f1.3</td>
<td>1.0, 1.0</td>
<td>0.5, 0.5</td>
<td>1.0, 1.0</td>
</tr>
</tbody>
</table>

Enumeration of 9 combinations of possible routes
2. **ROUTE SELECTION**

- **Input:** set of route for each flow
- **Output:** one route per flow
- **Maximize fairness**

**Naive exploration:** Enumeration all combinations (9 combinations)

**Exploration with pruning:** Ignore combinations with non-minimal bottleneck

---

![Routing Example](image)

- □ cluster
- ○ router

2. Route Selection

- Input: set of route for each flow
- Output: one route per flow
- Maximize fairness

- **Naive exploration**: Enumeration all combinations (9 combinations)
- **Exploration with pruning**: Ignore combinations with non-minimal bottleneck
  - Apply rate attribution (step 3)

<table>
<thead>
<tr>
<th></th>
<th>f2.1</th>
<th>f2.2</th>
<th>f2.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f1.2</td>
<td>1.0, 1.0</td>
<td>1.0, 1.0</td>
<td></td>
</tr>
<tr>
<td>f1.3</td>
<td>1.0, 1.0</td>
<td></td>
<td>1.0, 1.0</td>
</tr>
</tbody>
</table>

Enumeration of 4 combinations

Exploration vs. Our Exploration with Pruning

Combinations

Use cases

Pruned combinations
Remaining combinations

Exploration with pruning (EURS)
Works well for 11/18 test cases.
Exploration vs. Our Exploration with Pruning

Combination: $1.00 \times 10^{14}$

Use cases:
- BC-HOE
- BC-SCB
- BC-TP
- BR-HOE
- BR-SCB
- BR-TP
- E1-HOE
- E1-SCB
- E1-TP
- E2-HOE
- E2-SCB
- E2-TP
- S-HOE
- S-SCB
- S-TP
- THOE
- TSCB
- TP

- Pruned combinations
- Remaining combinations
2. Route Selection

- Input: set of routes for each flow
- Output: one route per flow
- Maximize fairness

- Naive exploration: Enumeration all combinations (9 combinations)
- Our Exploration with pruning: Ignore non-minimal bottleneck (4 combinations)
- Our LP-based Heuristic:
  1. Consider all alternative routes at once

2. **ROUTE SELECTION**

- **Input**: set of route for each flow
- **Output**: one route per flow
- **Maximize fairness**

- **Naive exploration**: Enumeration all combinations (9 combinations)
- **Our Exploration with pruning**: Ignore non-minimal bottleneck (4 combinations)
- **Our LP-based Heuristic**:
  - Consider all alternative routes at once
    1. Attribute fair rates

2. ROUTE SELECTION

- Input: set of route for each flow
- Output: one route per flow
- Maximize fairness

- **Naive exploration:** Enumeration all combinations (9 combinations)
- **Our Exploration with pruning:** Ignore non-minimal bottleneck (4 combinations)
- **Our LP-based Heuristic:**
  - Consider all alternative routes at once
    1. Attribute fair rates
    2. Remove smallest alternative routes

2. ROUTE SELECTION

- Input: set of route for each flow
- Output: one route per flow
- Maximize fairness

- **Naive exploration**: Enumeration all combinations (9 combinations)
- **Our Exploration with pruning**: Ignore non-minimal bottleneck (4 combinations)
- **Our LP-based Heuristic**:
  - Consider all alternative routes at once
    1. Attribute fair rates
    2. Remove smallest alternative routes

2. ROUTE SELECTION

- Input: set of route for each flow
- Output: one route per flow
- Maximize fairness

- Naiive exploration: Enumeration all combinations (9 combinations)
- Our Exploration with pruning: Ignore non-minimal bottleneck (4 combinations)
- Our LP-based Heuristic:
  - Consider all alternative routes at once
    1. Attribute fair rates
    2. Remove smallest alternative routes
    3. Enumerate the 3 combinations

Evaluation of Our LP-based Heuristic
Evaluation of Our LP-based Heuristic

Combinations

25

[Bar chart showing various categories and their corresponding values]
LP-BASED HEURISTIC VS. OPTIMAL EXPLORATION

▶ Optimal algorithms (100%): naive exploration, exploration with pruning
▶ Minimal rate as indicator of fairness
Part I: Semantics Preserving Parallelization

Part II: Real-Time Guarantees

Part III: Evaluation and Conclusion

- Use Cases and Evaluation
- Conclusion
THE ROSACE CASE STUDY

- Altitude-only flight controller
- Open source (Simulink, Lustre, Giotto) [Pagetti, Soussie, RTAS’14]

![Graph showing execution time comparison between different scenarios]

- +100: each task augmented with 100 cycles
- +200: each task augmented with 200 cycles
- Best effort (ASAP): no real-time guarantees
- Time-Triggered (TT): real-time guarantees
THE ROSACE CASE STUDY

- Altitude-only flight controller
- Open source (Simulink, Lustre, Giotto) [Pagetti, Soussié, RTAS’14]

![Graph showing performance comparisons]

- +100: each task augmented with 100 cycles
- +200: each task augmented with 200 cycles
- Best effort (ASAP): no real-time guarantees
- Time-Triggered (TT): real-time guarantees

[Graph showing performance improvements with +100 cycles and +200 cycles, comparing Sequential, Best effort, and TT approaches]
SYNTHETIC BENCHMARK ON 64 CORES

- 3 phases: Dispatch, Compute, Gather
- 20 Bytes per flow, high network congestion for gather phase

![Network Diagram]

- Dispatch
- Compute
- Gather
SYNTHETIC BENCHMARK ON 64 CORES

- 3 phases: Dispatch, Compute, Gather
- 20 Bytes per flow, high network congestion for gather phase

- 54% of WCRT for functional code
- 46% of WCRT for communication and system code
CONCLUSION

Parallelism Extraction

Mapping/Scheduling

Code Generation

Communication

Interference

Real-Time NoC

Evaluation

Contribution

External tool

N1 N2 N3 N4 N5 N6

Dependency graph

functional code

N1.c N2.c N3.c N4.c N5.c N6.c

Mapping + Non-preemptive scheduling

Code Generation

System + Communication

NoC Routing + Rate Attribution

Network Calculus (WCTT)

Executable for Kalray

WCET Analysis

release dates

MIA

Conclusion

Publications

References
CONCLUSION

- Contributions
- External contributions

Semantics

Preserving

Code

Generation

Parallelism Extraction

Communication and Dependency graph

NoC Routing + Rate Attribution

Executable for Kalray

Network Calculus (WCTT)

WCET Analysis

MIA

release dates
CONCLUSION

Semantics

Preserving

Code

Generation

Time-triggered

Execution

Model

Parallelism Extraction

Communication and Dependency graph

NoC Routing + Rate Attribution

Network Calculus (WCTT)

Contrib

Externa

release dates

CONCLUSION

Communication and

Contrib

Externa

release dates

CONCLUSION

Communication and

Contrib

Externa

release dates

CONCLUSION

Communication and

Contrib

Externa

release dates

CONCLUSION

Communication and

Contrib

Externa

release dates
CONCLUSION

Semantics

Preserving Code Generation

Hard real-time Guarantees

Many-Core Performance

Time-triggered Execution Model

Contrib

External

release dates

Bridge between academic and industry
CONCLUSION

Semantics
Preserving Code Generation
Hard realtime Guarantees
Many-Core Performance
Time-triggered Execution Model

Bridge between academic and industry
**FUTURE WORK**

- Semantics
- Preserving Code Generation
- Time-triggered Execution Model
- Hard real-time Guarantees
- Many-Core Performance

Thank you for your attention. Questions?
FUTURE WORK

- Release date computation and memory interferences analysis with NoC
- Hard real-time guarantees
- Many-core performance
- Time-triggered execution model
- Semantics preserving code generation
FUTURE WORK

- Memory size problem (overlays?)
- Release date computation and memory interferences analysis with NoC

Semantics
Preserving
Code Generation

Hard realtime
Guarantees
Many-Core
Performance

Time-triggered Execution Model
**FUTURE WORK**

- Input/Output Management
- Memory size problem (overlays?)
- Release date computation and memory interferences analysis with NoC
- Semantics
- Preserving
- Code Generation
- Hard realtime Guarantees
- Many-Core Performance
- Time-triggered Execution Model

Thank you for your attention. Questions?
**FUTURE WORK**

- Input/Output Management
- Memory size problem (overlays?)
- Release date computation and memory interferences analysis with NoC

**Semantics**
- Preserving
- Code Generation

**Hard realtime**
- Guarantees
- Many-Core Performance

**Time-triggered**
- Execution Model

What should be the standard real-time multi-core processor?
**FUTURE WORK**

- Input/Output Management
- Memory size problem (overlays?)
- Release date computation and memory interferences analysis with NoC

- Time-triggered Execution Model
- Hard real-time Guarantees
- Many-Core Performance
- Semantics
- Preserving Code Generation

What should be the standard real-time multi-core processor?

Thank you for your attention. Questions?
**Published**
Graillat A., Dupont de Dinechin B., *DATE 2018*
Parallel Code Generation of Synchronous Programs for a Many-core Architecture.

Boyer M., Dupont de Dinechin B., Graillat A., Havet L, *ERTS 2018*
Computing Routes and Delay Bounds for the Network-on-Chip of the Kalray MPPA2 Processor.

Dupont de Dinechin B., Graillat A., *NoCArc 2017*
Feed-Forward Routing for the Wormhole Switching Network-on-Chip of the Kalray MPPA2-256 Processor.

Dupont de Dinechin B., Graillat A., *AISTECS 2017*
Network-on-Chip Service Guarantees on the Kalray MPPA-256 Bostan Processor.

**Submitted**
# REFERENCES
DEADLOCK IN WORMHOLE NETWORKS

- This instance of flows deadlocks
- A wormhole packet is “spread” along the route
- Links 1-4 and 3-2 are shared
- A holds 1-4 but waits for 3-2
- B holds 3-2 but waits for 1-4

- Deadlock-freeness can be ensured at routing time
- Solutions: XY, Hamiltonian Odd-Even, Turn Prohibition, etc
MAX MIN FAIR RATE ATTRIBUTION

- Rate limiter configuration to avoid buffer overflow
- Rate of a flow $f_i$ noted $\rho_i$
- Valid: for each link, $\sum_i \rho_i \leq 1$ flit/cycle
- Fair attribution: max-min fairness [? ]: “cannot increase a rate without decreasing an already smaller (or equal) rate”

![Diagram showing cluster and router nodes with flows f1, f2, f3, f4.](image-url)
**Max Min Fair Rate Attribution**

- Rate limiter configuration to avoid buffer overflow
- Rate of a flow $f_i$ noted $\rho_i$
- Valid: for each link, $\sum_i \rho_i \leq 1$ flit/cycle
- Fair attribution: max-min fairness: “cannot increase a rate without decreasing an already smaller (or equal) rate”

Example 1:

$f_1 = \frac{1}{2}$, $f_2 = \frac{1}{2}$, $f_3 = \frac{1}{4}$, $f_4 = \frac{1}{4}$
**Max Min Fair Rate Attribution**

- Rate limiter configuration to avoid buffer overflow
- Rate of a flow $f_i$ noted $\rho_i$
- Valid: for each link, $\sum_i \rho_i \leq 1$ flit/cycle
- Fair attribution: max-min fairness [?]: “cannot increase a rate without decreasing an already smaller (or equal) rate”

**Example 1:**
- $f_1 = \frac{1}{2}, f_2 = \frac{1}{2}, f_3 = \frac{1}{4}, f_4 = \frac{1}{4}$
- $\Rightarrow$ Valid but **not** max-min fair (since increasing $f_3$ or $f_4$ can be done by reducing the greater $f_2$)

**Diagram:**
- Cluster
- Router

![Diagram showing network with nodes and links labeled with flow rates and corresponding rate limiter configurations to avoid buffer overflow.](diagram.png)
MAX MIN FAIR RATE ATTRIBUTION

- Rate limiter configuration to avoid buffer overflow
- Rate of a flow $f_i$ noted $\rho_i$
- Valid: for each link, $\sum_i \rho_i \leq 1$ flit/cycle
- Fair attribution: max-min fairness [?]: “cannot increase a rate without decreasing an already smaller (or equal) rate" 

Example 1:
$f_1 = \frac{1}{2}, f_2 = \frac{1}{2}, f_3 = \frac{1}{4}, f_4 = \frac{1}{4}$
→ Valid but not max-min fair (since increasing $f_3$ or $f_4$ can be done by reducing the greater $f_2$)

Example 2:
$f_1 = \frac{2}{3}, f_2 = \frac{1}{3}, f_3 = \frac{1}{3}, f_4 = \frac{1}{3}$
Max Min Fair Rate Attribution

- Rate limiter configuration to avoid buffer overflow
- Rate of a flow $f_i$ noted $\rho_i$
- Valid: for each link, $\sum_i \rho_i \leq 1$ flit/cycle
- Fair attribution: max-min fairness [? ]: “cannot increase a rate without decreasing an already smaller (or equal) rate”

Example 1:

- $f_1 = \frac{1}{2}$, $f_2 = \frac{1}{2}$, $f_3 = \frac{1}{4}$, $f_4 = \frac{1}{4}$
- Valid but not max-min fair (since increasing $f_3$ or $f_4$ can be done by reducing the greater $f_2$)

Example 2:

- $f_1 = \frac{2}{3}$, $f_2 = \frac{1}{3}$, $f_3 = \frac{1}{3}$, $f_4 = \frac{1}{3}$
- Valid and max-min fair (increasing $f_2$, $f_3$ or $f_4$ cannot be done without reducing one of them)

Classical solution: the “Water Filling” algorithm [?]
**Deterministic Network Calculus (DNC) Principle**

- **Cluster**
- **Router**

- **Arrival curve** is a maximum traffic entering the network
- **Service curve** is a minimum traffic handled by the network
- **How to compute service curve?**

Arrival Curve: $\gamma_{r,b}(t)$

Service Curve: $\beta_{R,T}(t)$

Convolution: $(\beta_{R,T} \otimes \gamma_{r,b})(t)$
Kalray MPPA2 Network-on-Chip

Kalray MPPA2 Network Elements

- cluster
- router

FIFO

Round-Robin
**Separated Flow Analysis (1/2)**

- Compute the service curve of each network element
- Compute the successive arrival curves at each network element
- Convolution of the element service curves → network service curve
**Separated Flow Analysis (2/2)**

- Service curve offered to $f_2$?
- At routers 1, 2, 3, 7 and 11.
BLIND MULTIPLEXING

- No information about the arbitration: consider f2 is low priority.
- Can we do better?
ROUND ROBIN MULTIPLEXING

- Packets of size $l^{max}$
- Restriction: Rate $\leq R$ (not applicable to $f_1$)
- Blind multiplexing is the conservative solution.

Service Curve for $f_2$

\[ T = (N - 1)l^{max} \]
\[ R = \frac{1}{N} \]
WORST-CASE TRAVERSAL TIME (WCTT): APPLICATION (1/2)

- Service curve offered to f2?
- Router 1: Round Robin multiplexing (N=2)
- Router 2: non active (alone)
- Router 3: Round Robin multiplexing (N=2, f3 and f4 are aggregated with $b_a = \sum_{i \neq 2} b_i$, $r_a = \sum_{i \neq 2} r_i$)
- Router 7 and 11: non active
Worst-Case Traversal Time (WCTT): Application (2/2)

With packets of $l^{\text{max}}=17$ flits.
EVALUATION OF OUR LP-BASED HEURISTIC

Combinations
EVALUATION OF OUR LP-BASED HEURISTIC
**LP-based Heuristic vs. Optimal Exploration**

- Optimal algorithms (100%): naive exploration, exploration with pruning
- Minimal rate as indicator of fairness