

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



Amaury Grailat

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

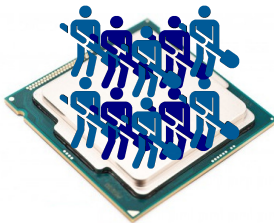
Aircraft: since 1980s
for digital fly-by-wire system

End of production
No longer sufficient



Multi-Core

Available since 2001



Many-Core

Available since 2007

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

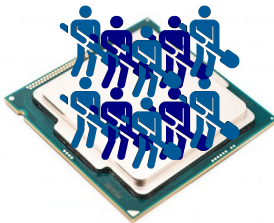
Well-proven for real-time

End of production
No longer sufficient



Multi-Core

Available since 2001



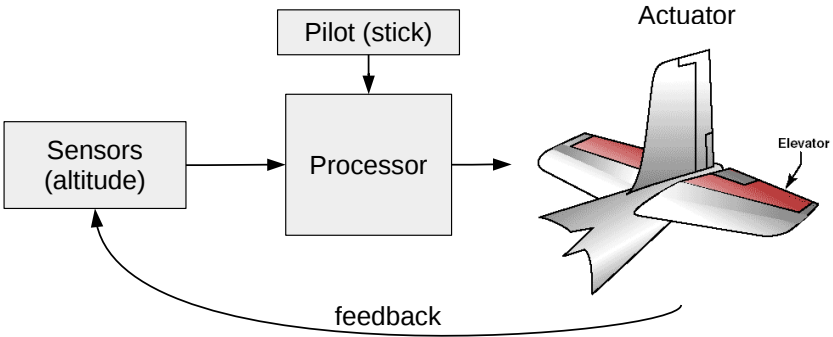
Many-Core

Available since 2007

More functions in one chip
More energy efficient

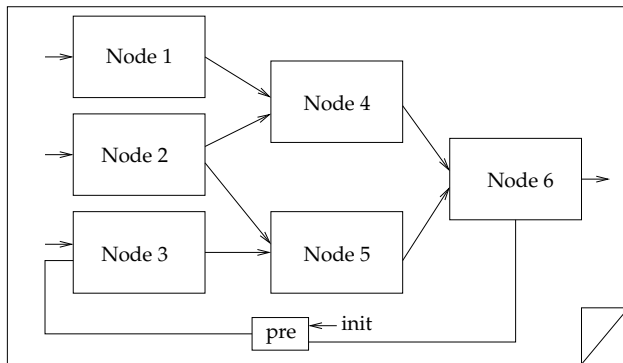
SAFETY CRITICAL SYSTEMS

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

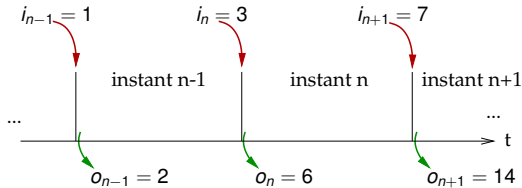
THE SYNCHRONOUS DATA-FLOW LANGUAGES (1/2)



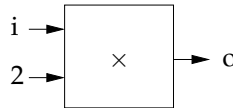
- ▶ Lustre (academic), Scade (industrial)
- ▶ Network of nodes

THE SYNCHRONOUS DATA-FLOW LANGUAGES (2/2)

Logical

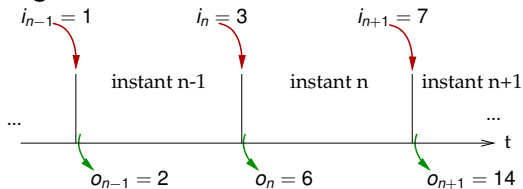


► One execution is called a logical instant



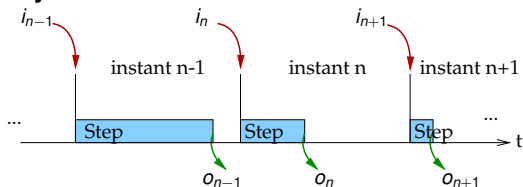
THE SYNCHRONOUS DATA-FLOW LANGUAGES (2/2)

Logical



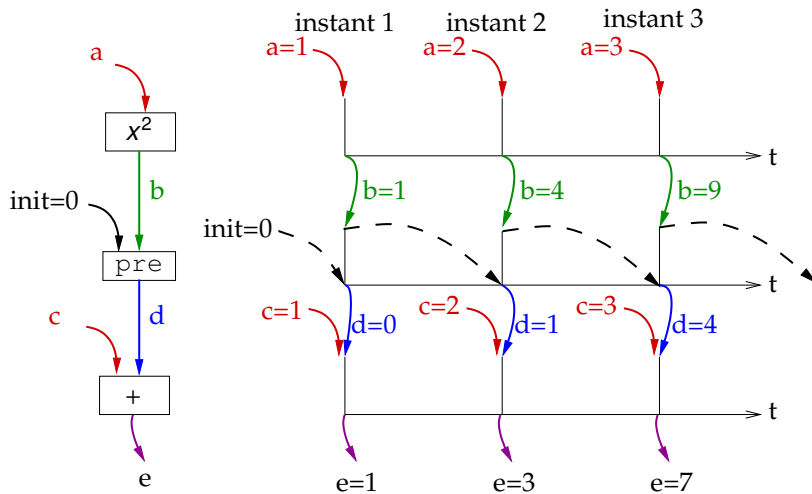
- One execution is called a logical instant

Physical



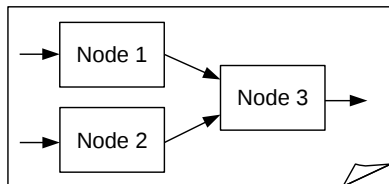
- Requirement: o_n before i_{n+1} .
- Worst-Case Execution Time (WCET)

LUSTRE/SCADE DELAY OPERATOR



- **previous operator**
- Logical/functional delay

CODE GENERATION FOR SINGLE-CORE

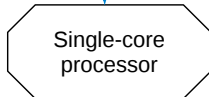


Formal semantics
Determinism

Code Generation



Compilation



```

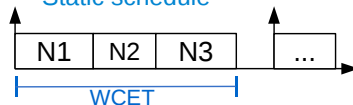
for each period {
  i = sensors();
  o = main_step(i);
  actuators(o);
}
  
```

Static schedule

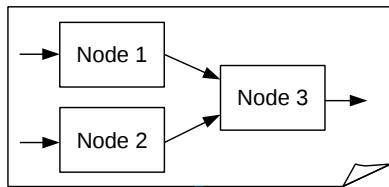
```

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

Static schedule



CODE GENERATION FOR SINGLE-CORE

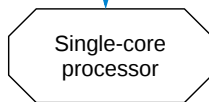


Formal semantics
Determinism

Code Generation



Compilation



```

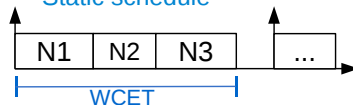
for each period {
  i = sensors();
  o = main_step(i);
  actuators(o);
}
  
```

Static schedule

```

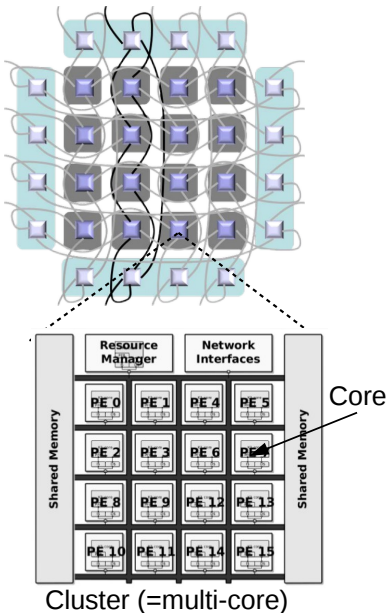
void main_step(i) {
  o1 = N1_step(i);
  o2 = N2_step(i);
  returns N3_step(o1, o2);
}
  
```

Static schedule



OK for sequential execution. What about parallel?

THE KALRAY MPPA2

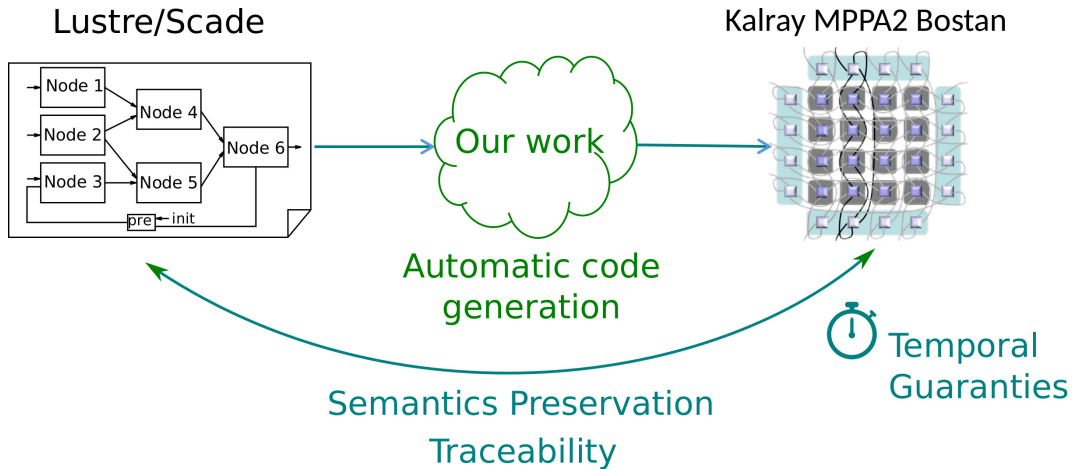


Properties of the Kalray MPPA2:

- ▶ **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



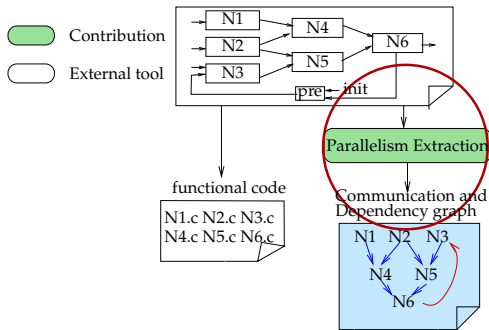
Part I: Semantics Preserving Parallelization

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

Part II: Real-Time Guarantees

Part III: Evaluation and Conclusion

EXTRACTION OF PARALLELISM



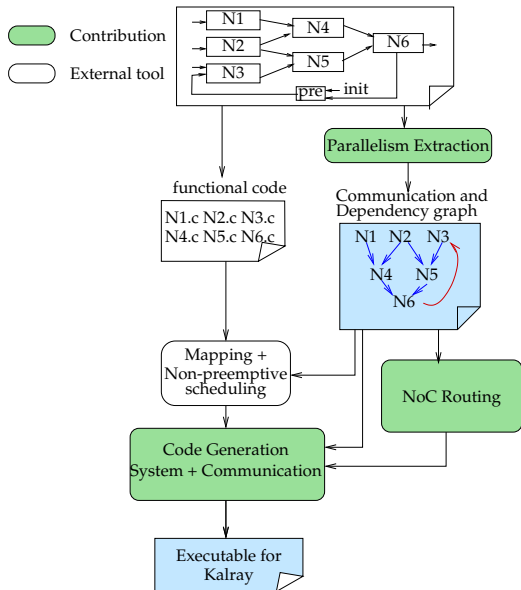
Method 1:

- 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



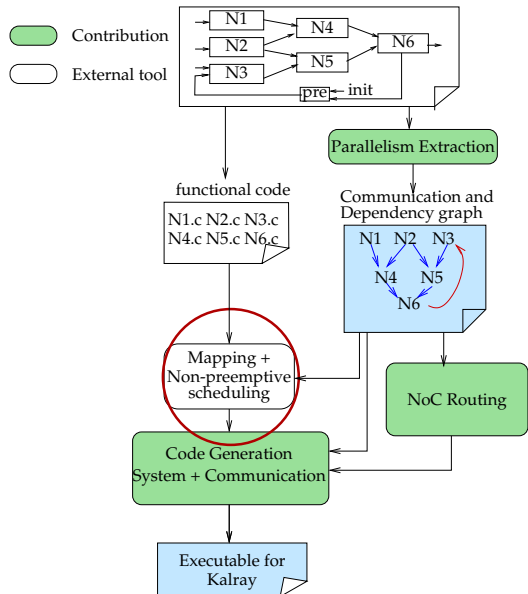
Method 1:

- 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:

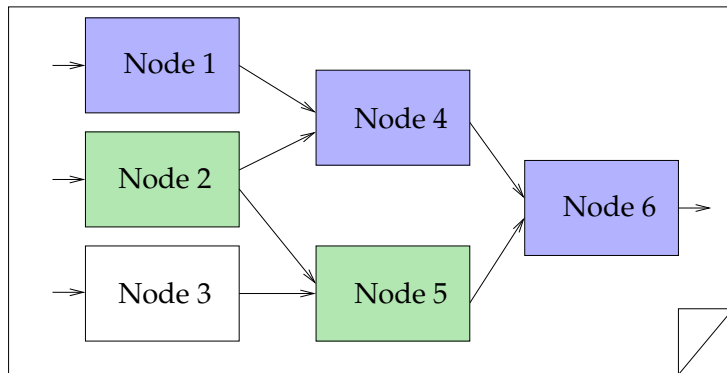
- see manuscript

MAPPING / SCHEDULING



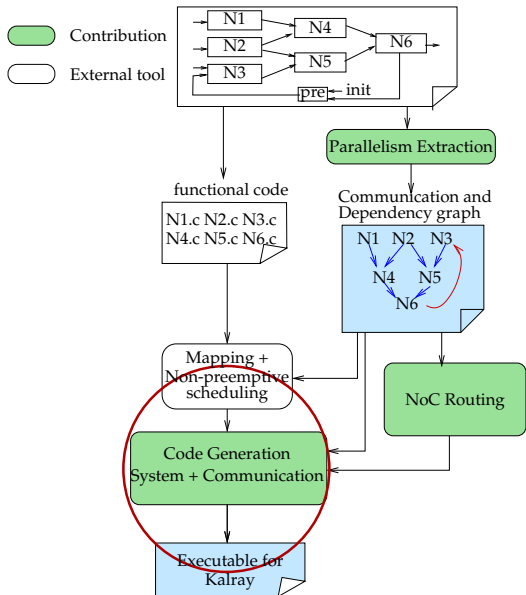
- Need non-preemptive static schedule
- External scheduling tool
- We can easily check the schedule

MAPPING/SCHEDULING: EXAMPLE

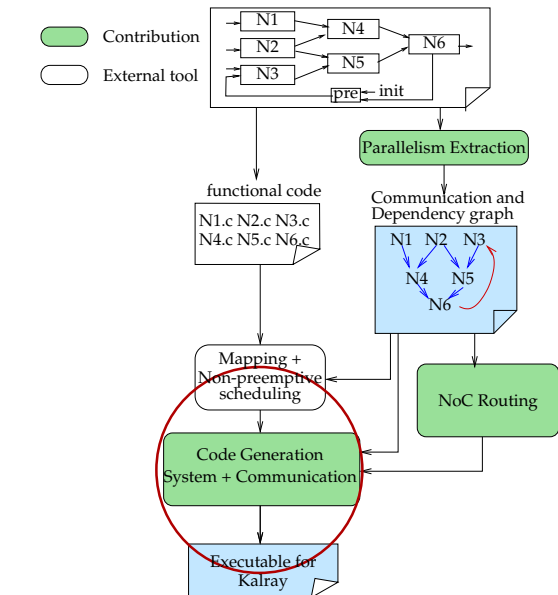


- Core 0: N1 ; N4 ; N6
Core 1: N2 ; N5
Core 2: N3
- Schedule checked using the dependency graph

CODE GENERATION



CODE GENERATION



- Dependencies compiled into “wait for input”.

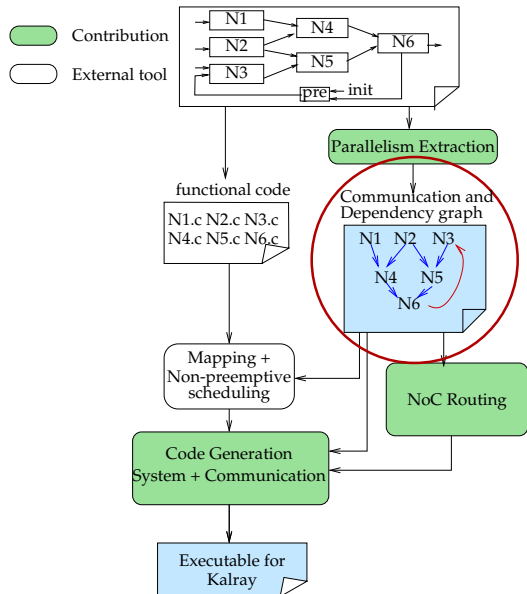
Sketch of code for core 0.

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

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

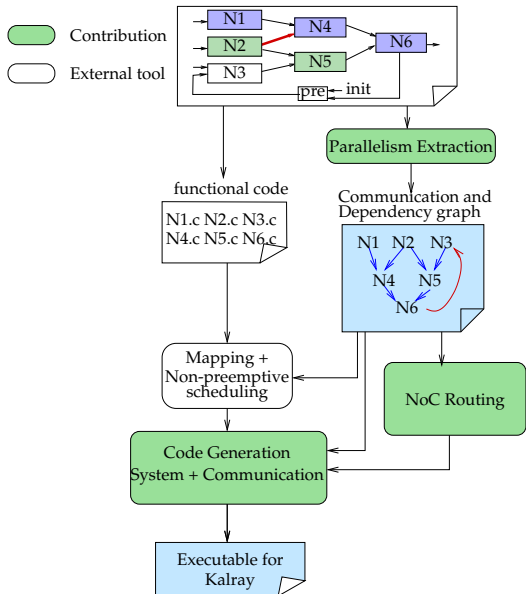
    wait_inputs_N6(); // N6
    N6_step();
    write_outputs_N6();
}
```

COMMUNICATION CHANNELS



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

INSTANTANEOUS COMMUNICATION



```

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

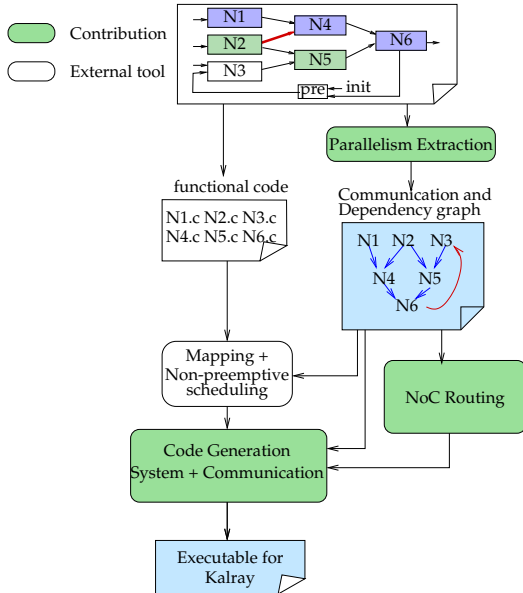
    copy(N5.in1, N2.out);
}

void wait_for_inputs_N4() {

```

- Efficient hardware synchronization
- Software-based cache coherency

INSTANTANEOUS COMMUNICATION



```

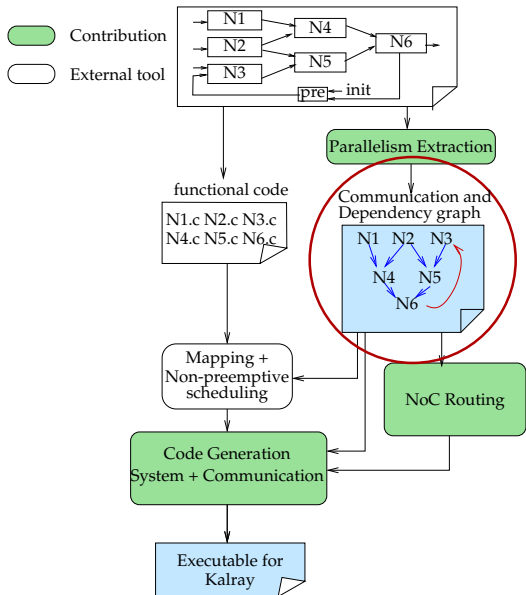
void write_outputs_N2() {
    copy(N4.in2, N2.out);
    channel_N2_N4 = true;
    // + cache management
    notify(core_N4);
    copy(N5.in1, N2.out);
}
  
```

```

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
}
  
```

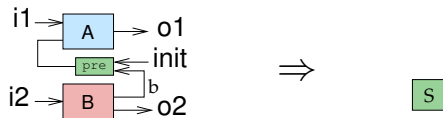
- Efficient hardware synchronization
- Software-based cache coherency

COMMUNICATION CHANNELS

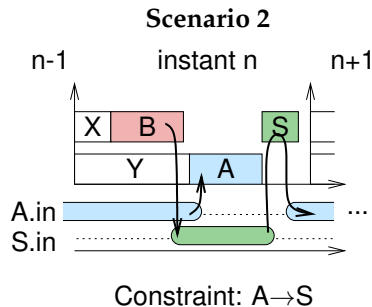
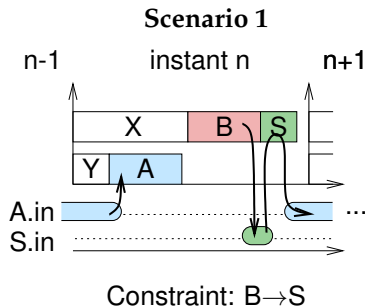


- Two kinds of communications:
- instantaneous (→)
 - **delayed** (→)

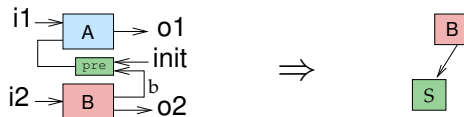
DELAYED COMMUNICATION



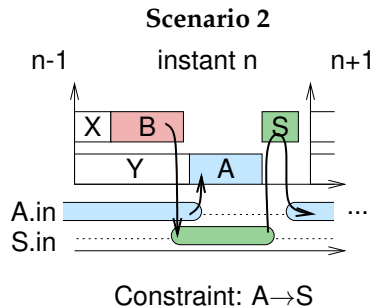
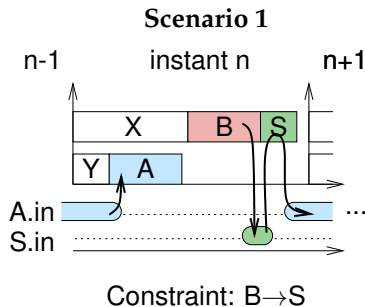
Transformation into a *SWAP* + scheduling constraints.



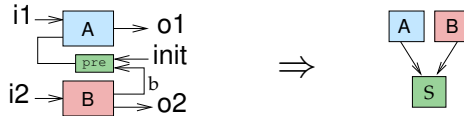
DELAYED COMMUNICATION



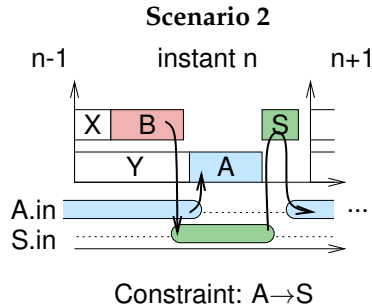
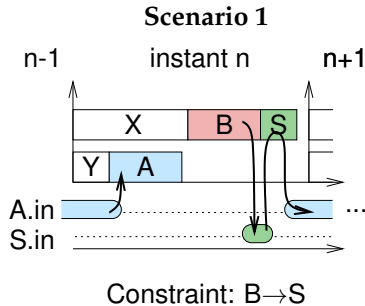
Transformation into a *SWAP* + scheduling constraints.



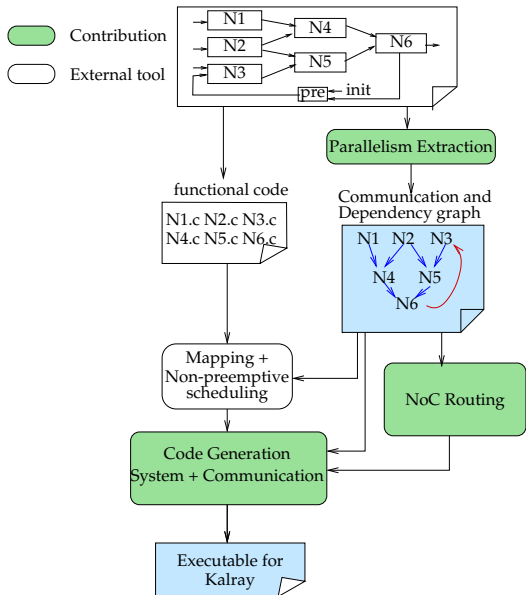
DELAYED COMMUNICATION



Transformation into a *SWAP* + scheduling constraints.



CONCLUSION OF PART I



- Semantics preserved
- Next step: Temporal Guaranties

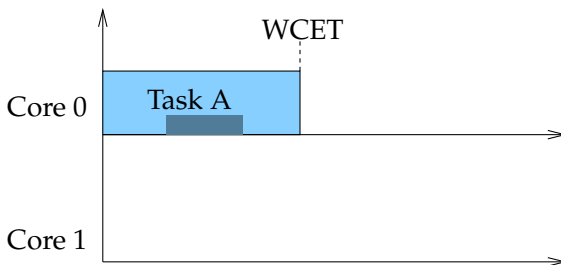
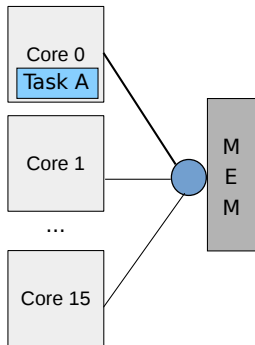
Part I: Semantics Preserving Parallelization

Part II: Real-Time Guarantees

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

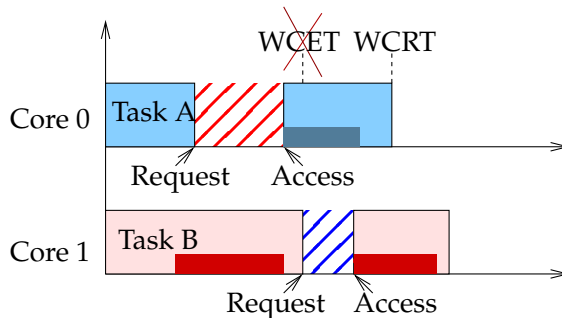
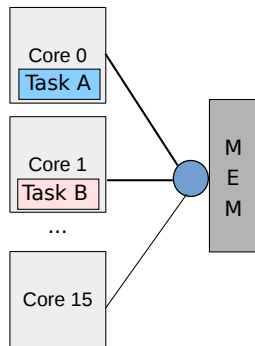
Part III: Evaluation and Conclusion

INTERFERENCE AND REACTION TIME



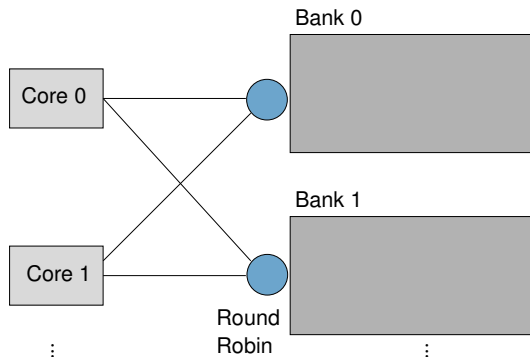
- Single-core: WCET is sufficient

INTERFERENCE AND REACTION TIME



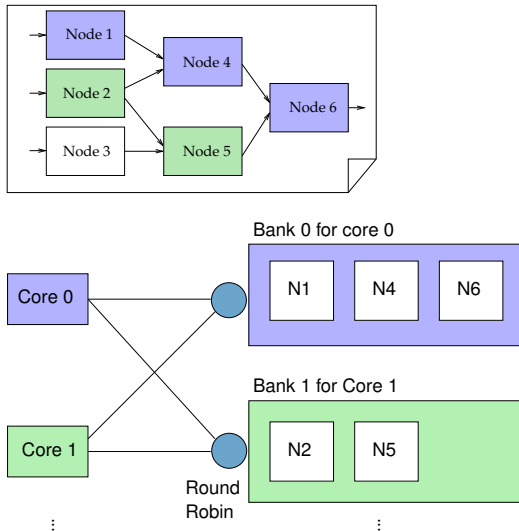
- ▶ Single-core: WCET is sufficient
- ▶ Multi-core: WCET+interference on shared resources = Worst-Case Response Time (WCRT).
- ▶ **WCET+interference too pessimistic in the general case**
 ⇒ **exploit the execution model in the analysis**

BANKED MEMORY



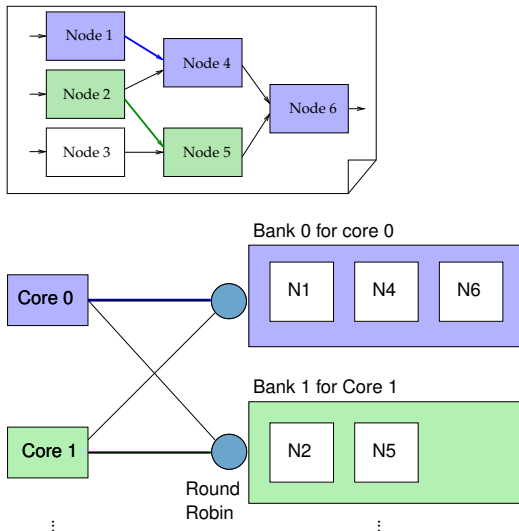
- Private arbiter for each bank
- Available in the Kalray MPPA2 (silicon)

BANKED MEMORY IN KALRAY MPPA2



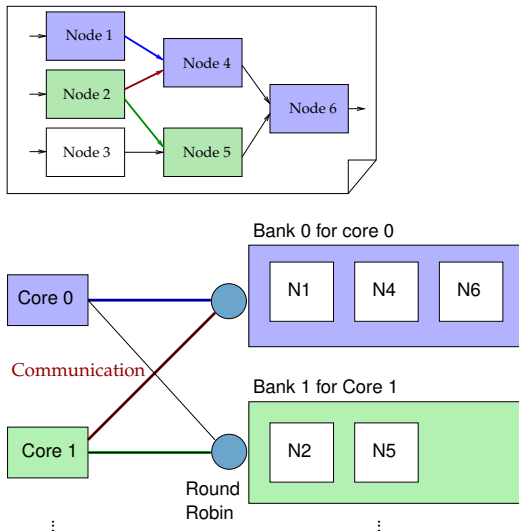
- Choice: Code, input buffer, local variables are mapped in core's bank of the core
- Interference on communication only

BANKED MEMORY IN KALRAY MPPA2



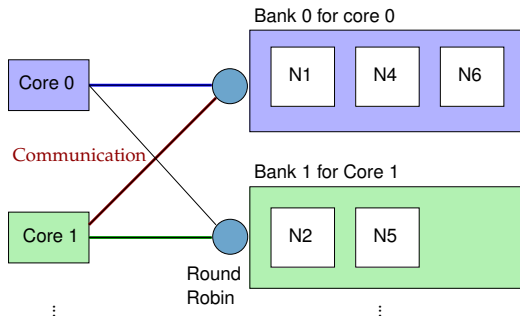
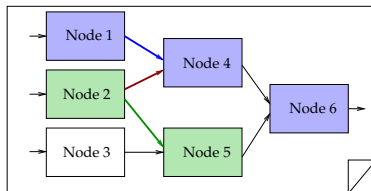
- Choice: Code, input buffer, local variables are mapped in core's bank of the core
- Interference on communication only

BANKED MEMORY IN KALRAY MPPA2



- Choice: Code, input buffer, local variables are mapped in core's bank of the core
- Interference on communication only

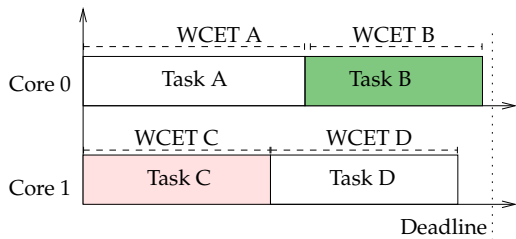
BANKED MEMORY IN KALRAY MPPA2



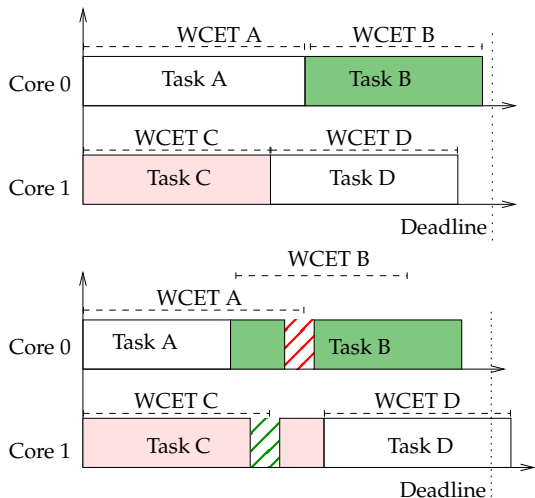
- Choice: Code, input buffer, local variables are mapped in core's bank of the core
- Interference on communication only

How to activate tasks?

PROBLEM WITH ASAP EXECUTION



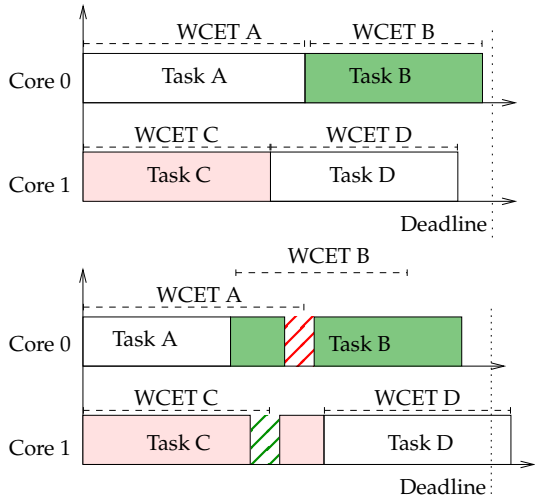
PROBLEM WITH ASAP EXECUTION



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

Timing Anomaly

PROBLEM WITH ASAP EXECUTION



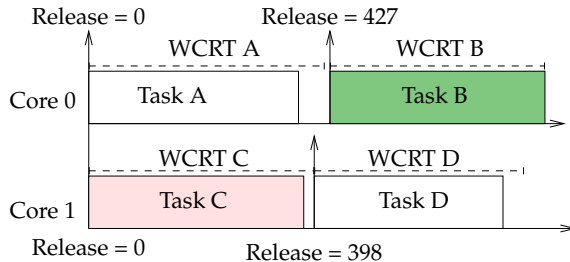
Faster execution of A \Rightarrow interference between B and C.

Timing Anomaly

Solution: release dates for tasks (time-triggered)

TIME-TRIGGERED EXECUTION MODEL

Principle

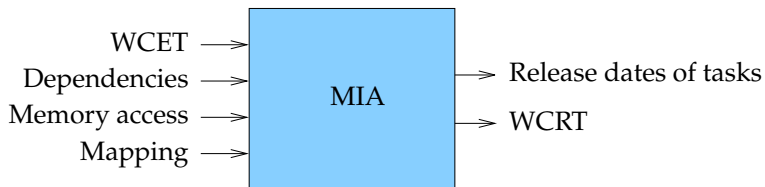


- Compute a static release date when data is guaranteed to be available
- Time-Triggered execution prevents tasks from starting earlier

Implementation

```
void wait_inputs_N1() {
    while(time() < t_period + release_date_N1) {
        // wait
    }
}
```

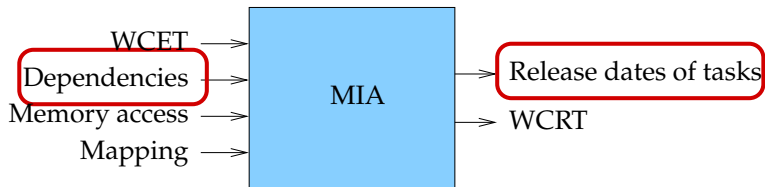
MULTI-CORE INTERFERENCE ANALYSIS



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

www-verimag.imag.fr/Multi-core-interference-Analysis.html

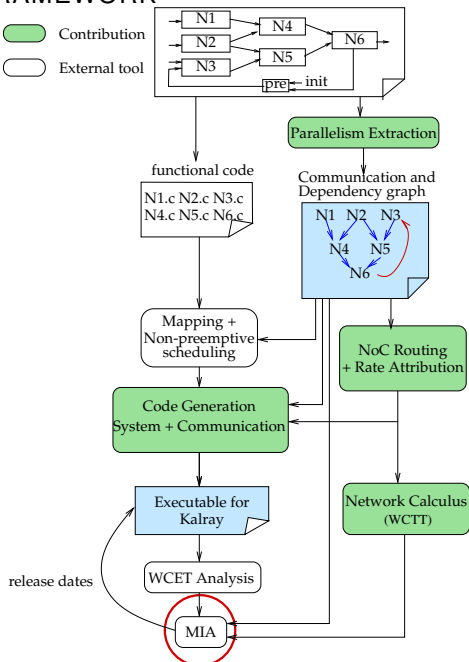
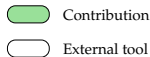
MULTI-CORE INTERFERENCE ANALYSIS



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

www-verimag.imag.fr/Multi-core-interference-Analysis.html

FRAMEWORK



- WCET Analysis (Ottawa, AiT)
- Computation of the release dates (MIA tool)
- Insertion in the executable.

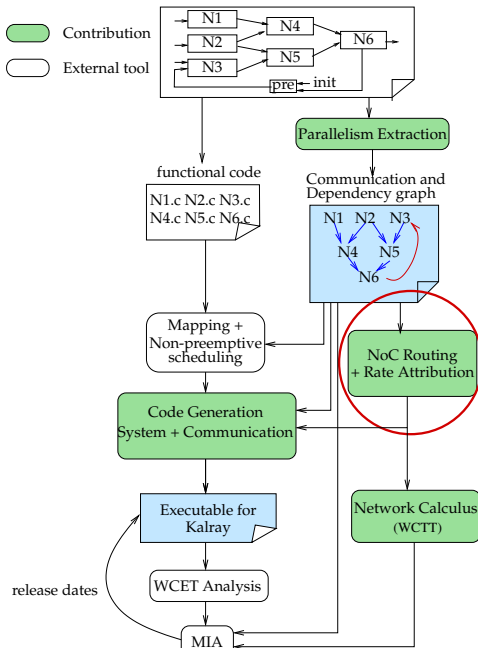
[Submitted: Graillat A., Rihani H., Maiza C., Moy M., Raymond P., Dupont de Dinechin B., *Real-Time Systems Journal*]

OUR EXECUTION MODEL

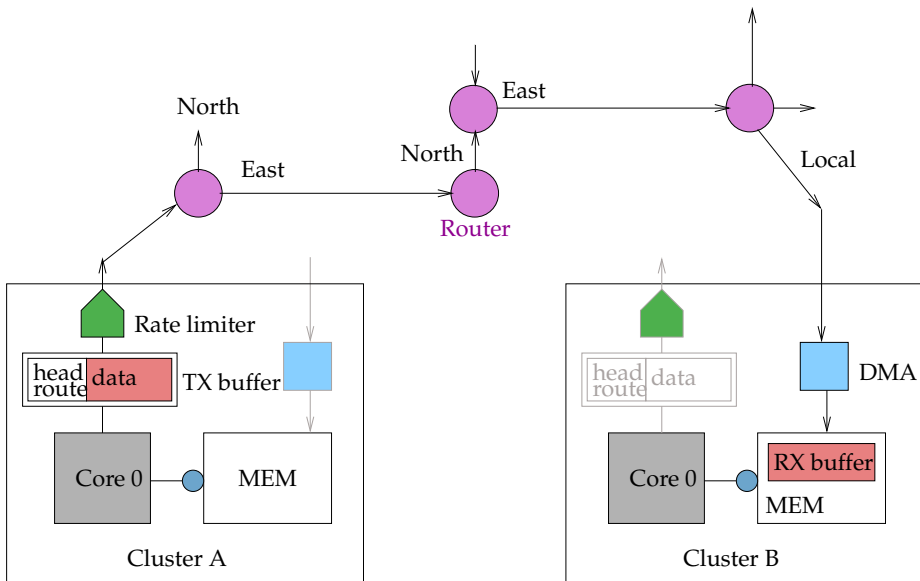
- ▶ Static scheduling
- ▶ Bare metal, no interrupt, no preemption
- ▶ Banked memory mapping (1 core \rightarrow 1 bank)
- ▶ Time-triggered

Deterministic and WCRT guarantee

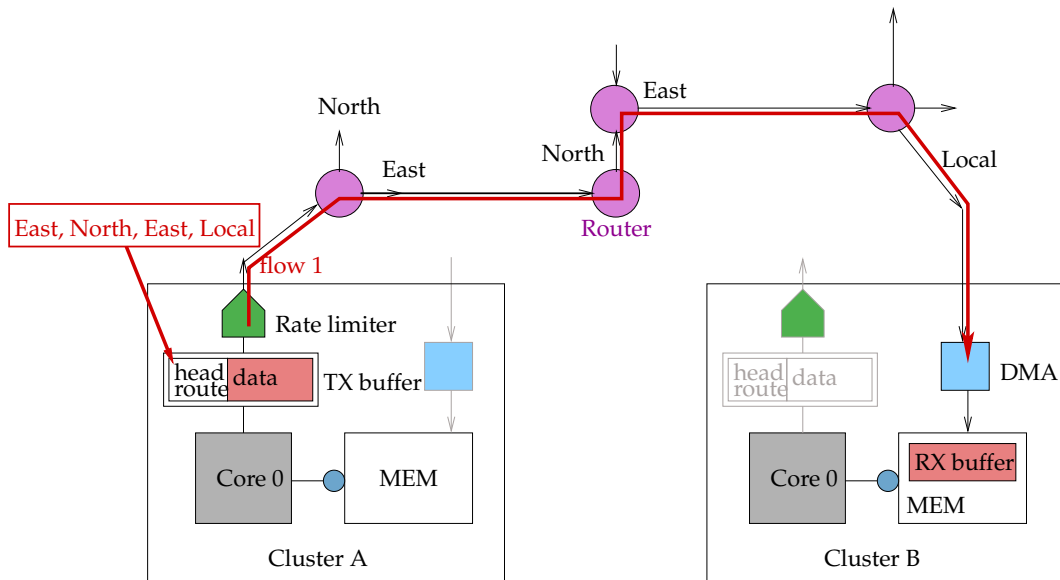
OVERVIEW



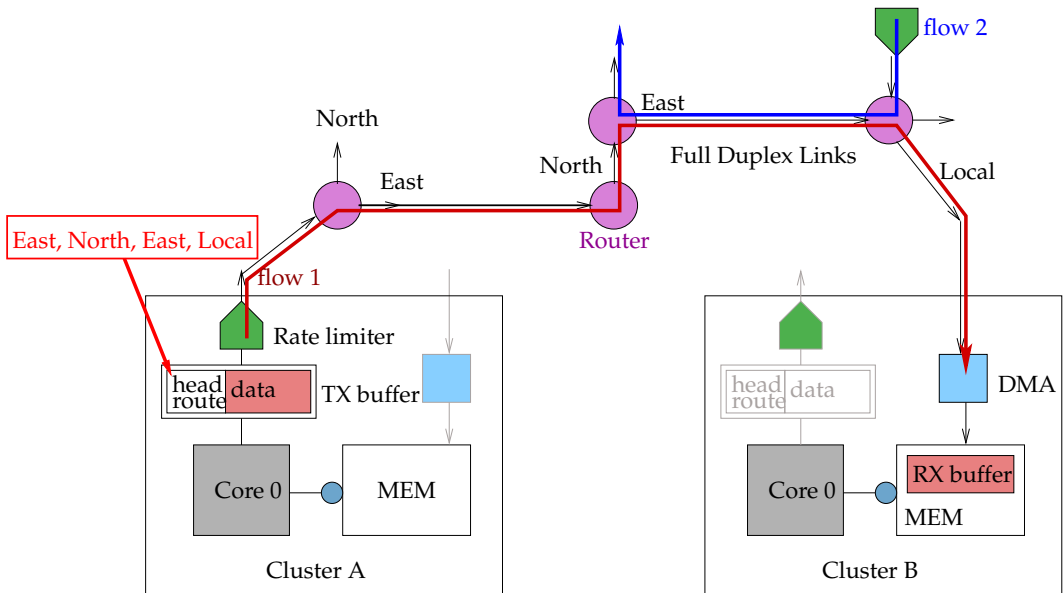
THE KALRAY MPPA2's NoC



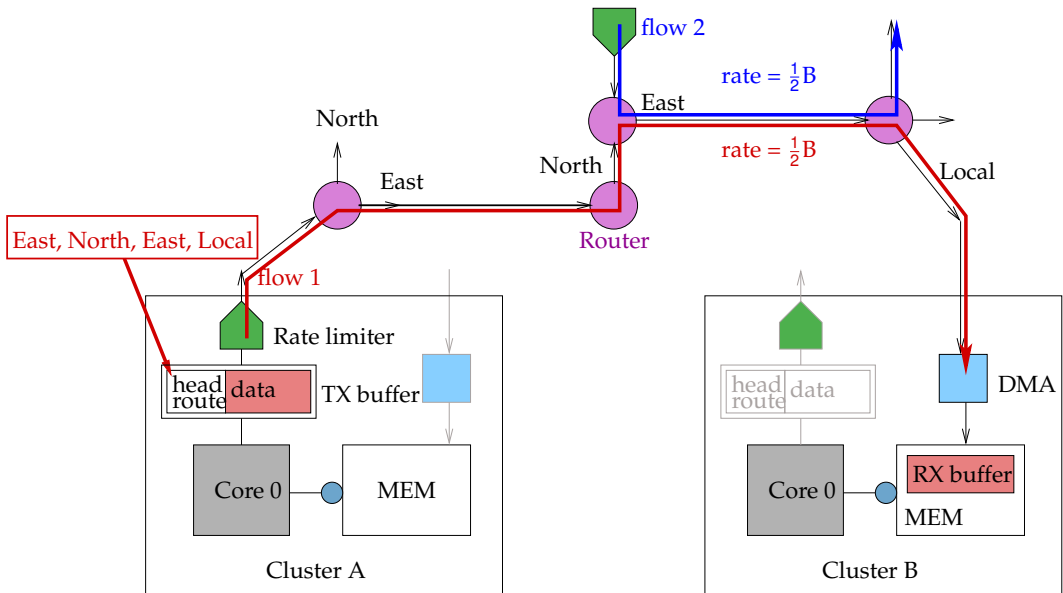
THE KALRAY MPPA2's NoC



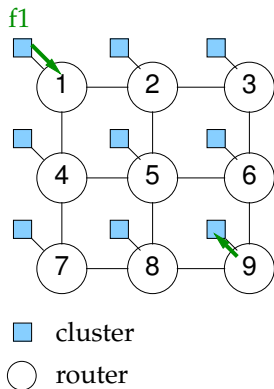
THE KALRAY MPPA2's NoC



THE KALRAY MPPA2's NoC



REAL-TIME GUARANTEES WITH NETWORK-ON-CHIP

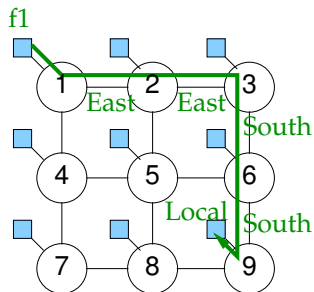


1. Routing

Route = sequence of directions computed by sender

[Dupont de Dinechin B., Graillat A.,
NoCArc 2017]

REAL-TIME GUARANTEES WITH NETWORK-ON-CHIP



■ cluster

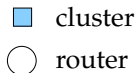
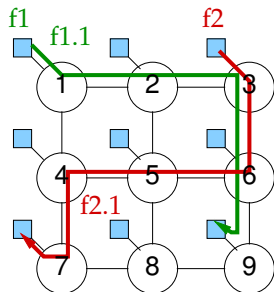
○ router

1. Routing

Route = sequence of directions computed by sender

[Dupont de Dinechin B., Graillat A.,
NoCArc 2017]

REAL-TIME GUARANTEES WITH NETWORK-ON-CHIP



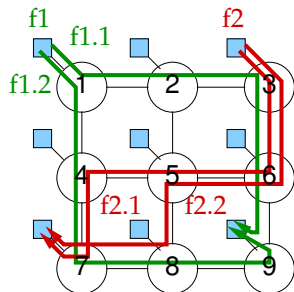
1. Routing

Route = sequence of directions computed by sender

Deadlock-free algorithms (XY, Hamiltonian)

[Dupont de Dinechin B., Graillat A.,
NoCArc 2017]

REAL-TIME GUARANTEES WITH NETWORK-ON-CHIP



■ cluster

○ router

1. Routing

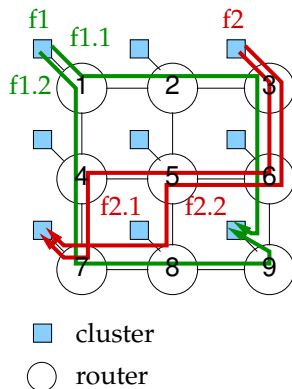
Route = sequence of directions computed by sender

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))

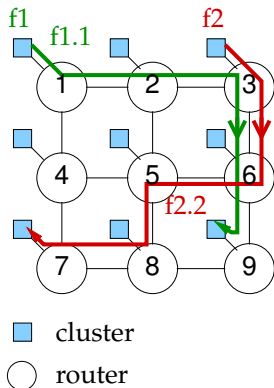
2. Route Selection

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

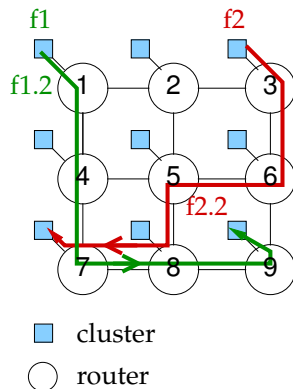
Choose one static route per flow

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

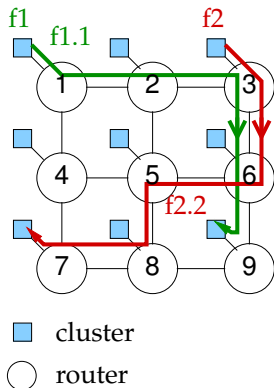
Choose one static route per flow

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}...

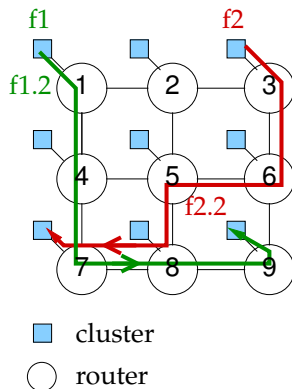
3. Rate attribution

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}...

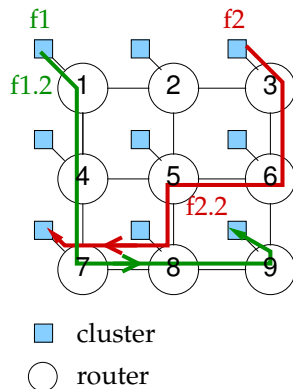
3. Rate attribution

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



[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))

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

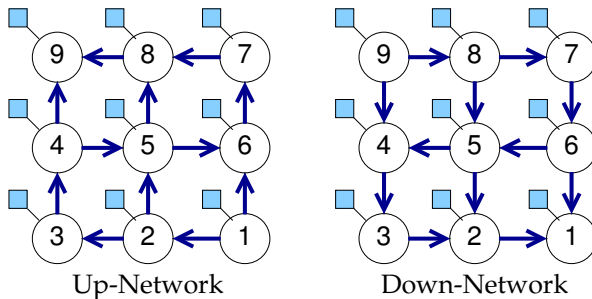
e.g. {0.5, 0.5}, {1.0, 1.0}...

4. Network Calculus

Compute latency and buffer level

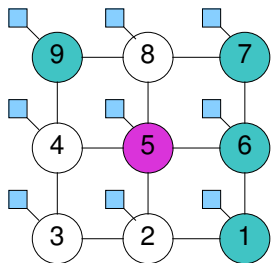
1. UNICAST ROUTING

Hamiltonian Routing



- Deadlock-free
- Path-Diversity

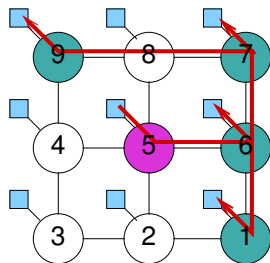
1. MULTICAST ROUTING PROBLEM



 Source
 Destination

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

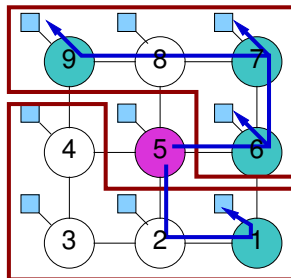
1. MULTICAST ROUTING PROBLEM



 Source
 Destination

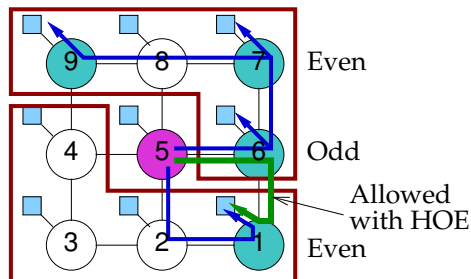
- Deadlock-free heuristic to “travelling salesman problem”:
- One route? Tree is no possible with Kalray architecture
- Hamiltonian: minimum of two routes

1. MULTICAST ROUTING: HAMILTONIAN



- Dual-Path + Hamiltonian without path diversity (Lin et al., 1992)
- Small path diversity, we can do better.

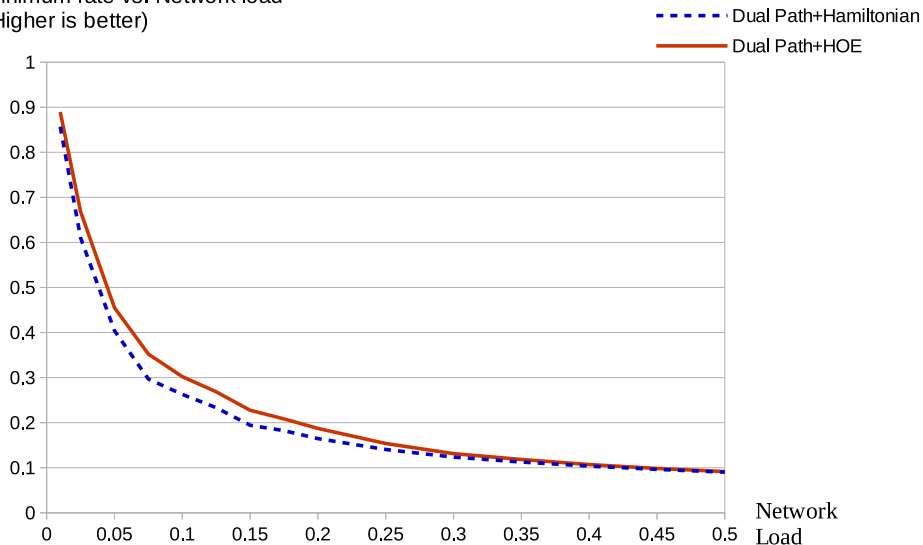
1. MULTICAST ROUTING: HOE



- Hamiltonian Odd Even (Bahrebar et al.) routing: allows some non-Hamiltonian paths
- Our choice: Dual-Path + HOE

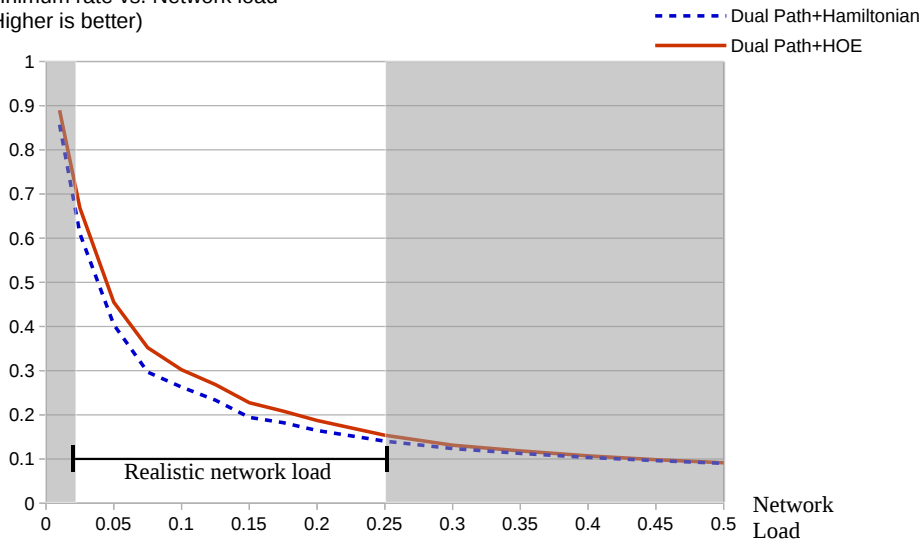
EVALUATION OF DEADLOCK-FREE MULTICAST ROUTING

Minimum rate vs. Network load
(Higher is better)



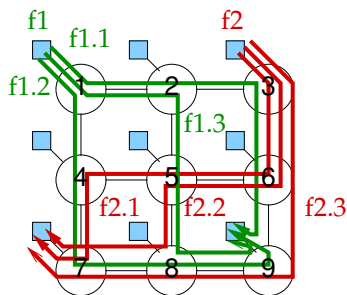
EVALUATION OF DEADLOCK-FREE MULTICAST ROUTING

Minimum rate vs. Network load
(Higher is better)



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

2. ROUTE SELECTION



■ cluster
○ router

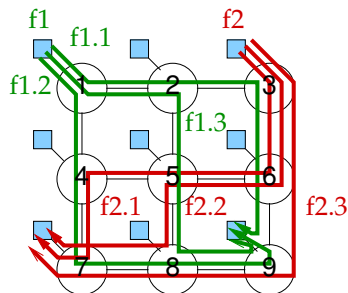
[Boyer M., Dupont de Dinechin B.,
Graillat A., Havet L, *ERTS 2018*]

- 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.

	f2.1	f2.2	f2.3
f1.1	0.5, 0.5	0.5, 0.5	0.5, 0.5
f1.2	0.5, 0.5	1.0, 1.0	1.0, 1.0
f1.3	1.0, 1.0	0.5, 0.5	1.0, 1.0

Enumeration of 9 combinations of possible routes

2. ROUTE SELECTION



■ cluster

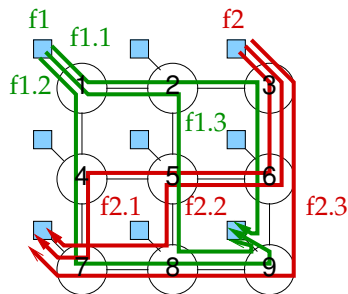
○ router

[Boyer M., Dupont de Dinechin B.,
Graillat A., Havet L, *ERTS 2018*]

- 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

	f2.1	f2.2	f2.3
f1.1			
f1.2			
f1.3			

2. ROUTE SELECTION



■ cluster

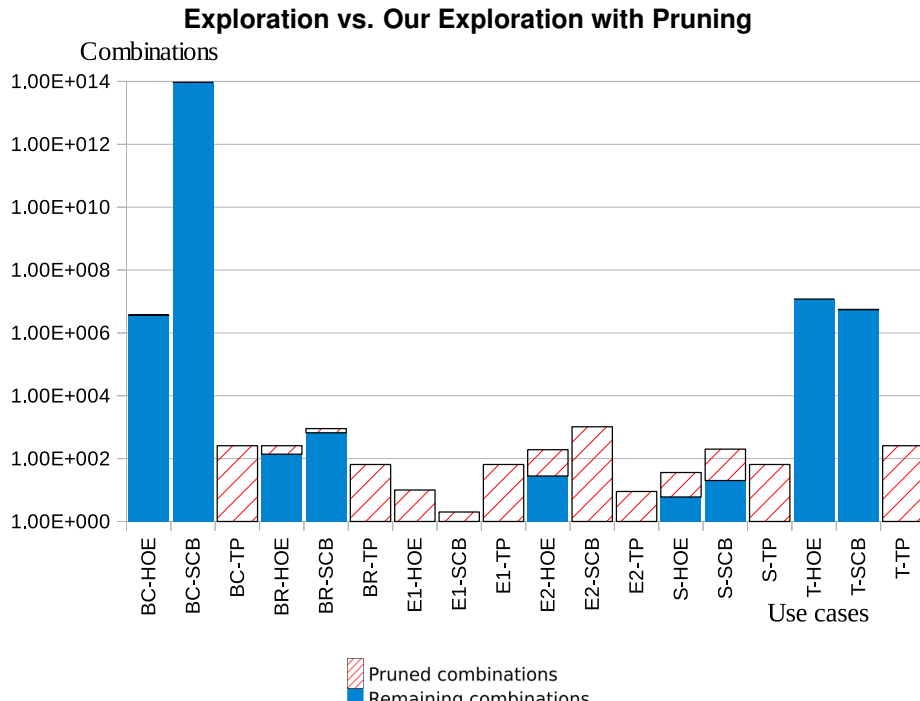
○ router

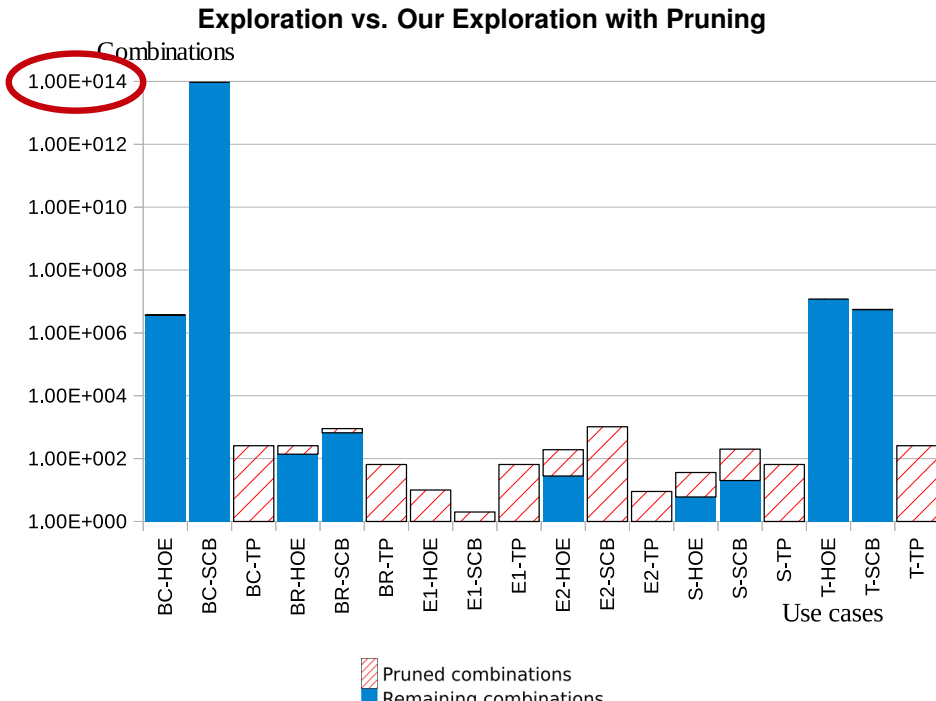
[Boyer M., Dupont de Dinechin B.,
Graillat A., Havet L, *ERTS 2018*]

- 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)

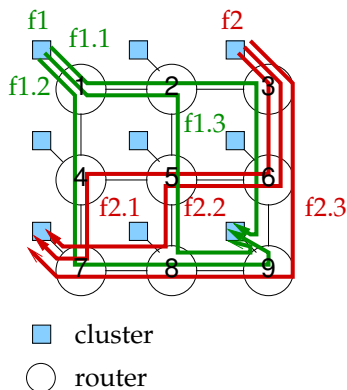
	f2.1	f2.2	f2.3
f1.1			
f1.2		1.0, 1.0	1.0, 1.0
f1.3	1.0, 1.0		1.0, 1.0

Enumeration of 4 combinations





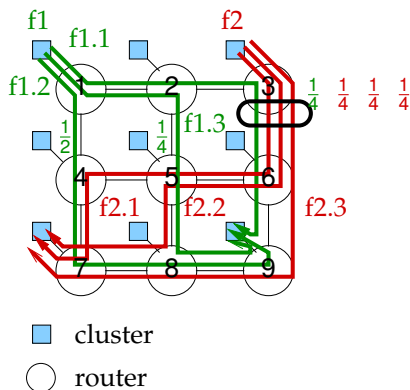
2. ROUTE SELECTION



- ▶ 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

[Boyer M., Dupont de Dinechin B., Graillat A., Havet L, *ERTS 2018*]

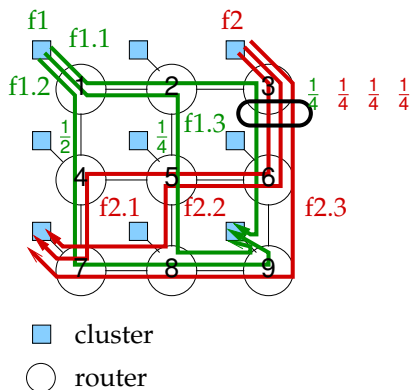
2. ROUTE SELECTION



- 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

[Boyer M., Dupont de Dinechin B.,
Graillat A., Havet L, *ERTS 2018*]

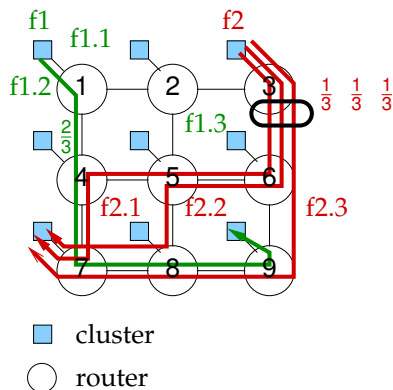
2. ROUTE SELECTION



[Boyer M., Dupont de Dinechin B.,
Graillat A., Havet L, *ERTS 2018*]

- 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

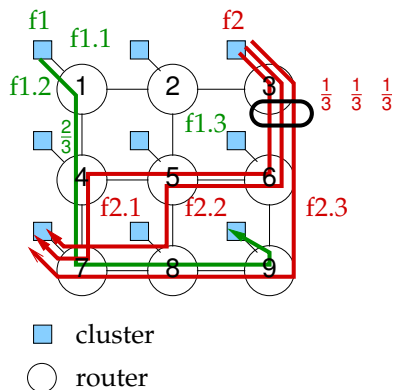
2. ROUTE SELECTION



[Boyer M., Dupont de Dinechin B.,
Graillat A., Havet L, *ERTS 2018*]

- 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

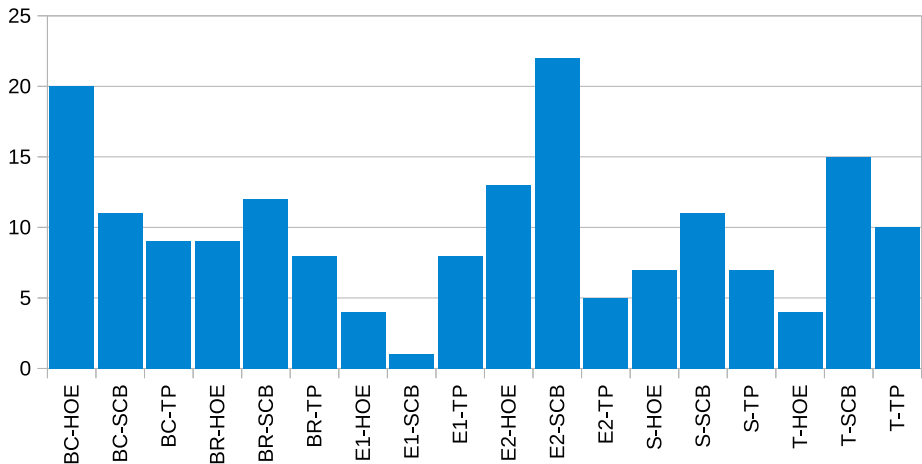


[Boyer M., Dupont de Dinechin B.,
Graillat A., Havet L, *ERTS 2018*]

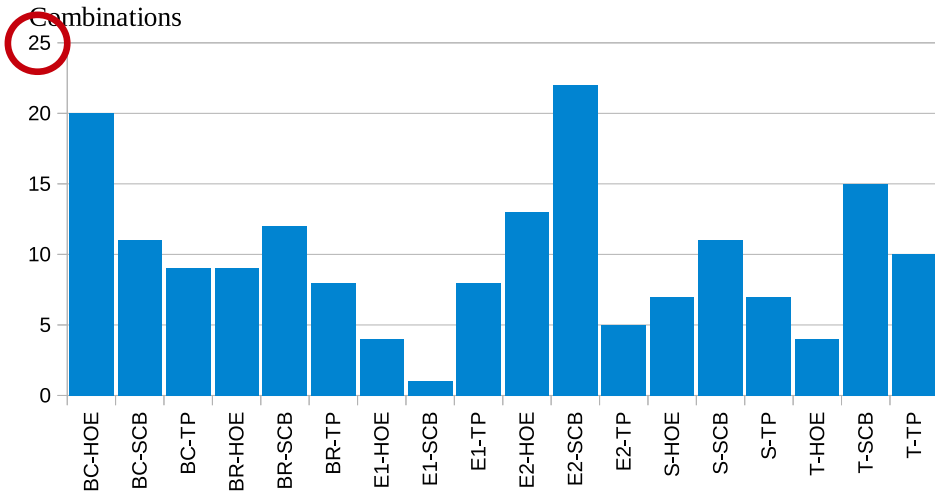
- ▶ Input: set of route for each flow
 - ▶ Output: one route per flow
 - ▶ Maximize fairness
-
- ▶ **Naive exploration:** Enumeration all combinations (9 combinations)
 - ▶ **Our Exploration with pruning:** Ignore non-minimal bottleneck (4 combinations)
 - ▶ **Our LP-based Heuristic:**
 - ▶ Consider all alternative routes at once
 1. Attribute fair rates
 2. Remove smallest alternative routes
 3. Enumerate the 3 combinations

EVALUATION OF OUR LP-BASED HEURISTIC

Combinations

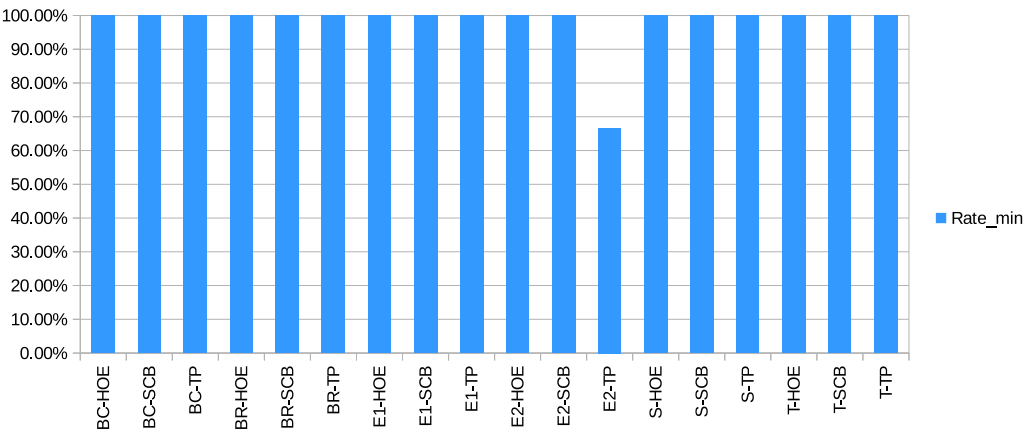


EVALUATION OF OUR LP-BASED HEURISTIC



LP-BASED HEURISTIC VS. OPTIMAL EXPLORATION

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



Part I: Semantics Preserving Parallelization

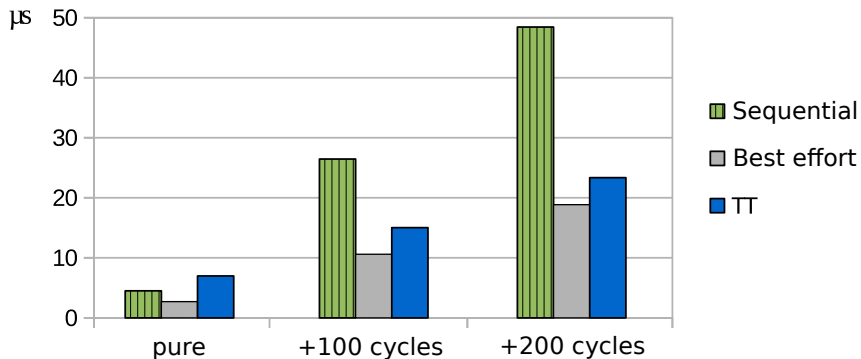
Part II: Real-Time Guarantees

Part III: Evaluation and Conclusion

- ▶ Use Cases and Evaluation
- ▶ Conclusion

THE ROSACE CASE STUDY

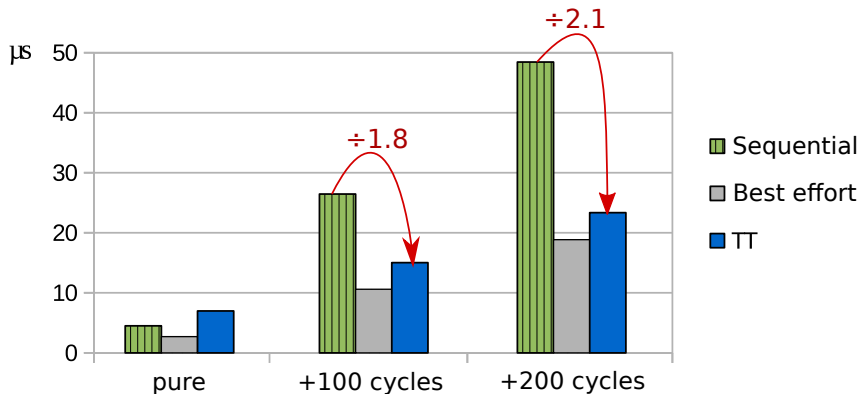
- ▶ Altitude-only flight controller
- ▶ Open source (Simulink, Lustre, Giotto) [Pagetti, Soussié, RTAS'14]



- ▶ +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

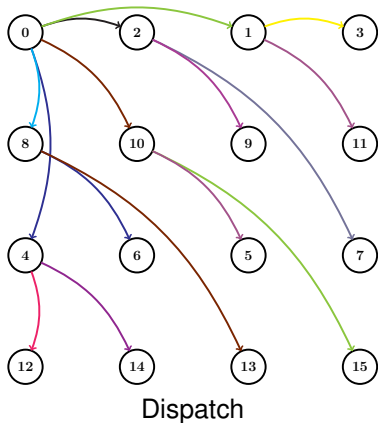
- ▶ Altitude-only flight controller
- ▶ Open source (Simulink, Lustre, Giotto) [Pagetti, Soussié, RTAS'14]



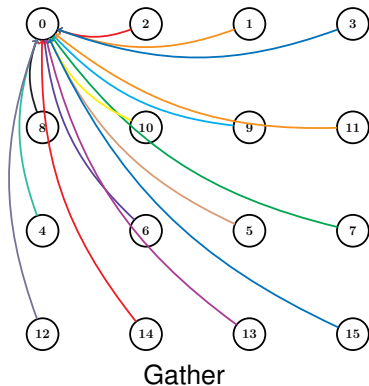
- ▶ +100: each task augmented with 100 cycles
- ▶ +200: each task augmented with 200 cycles
- ▶ Best effort (ASAP): no real-time guarantees
- ▶ Time-Triggered (TT): real-time guarantees

SYNTHETIC BENCHMARK ON 64 CORES

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

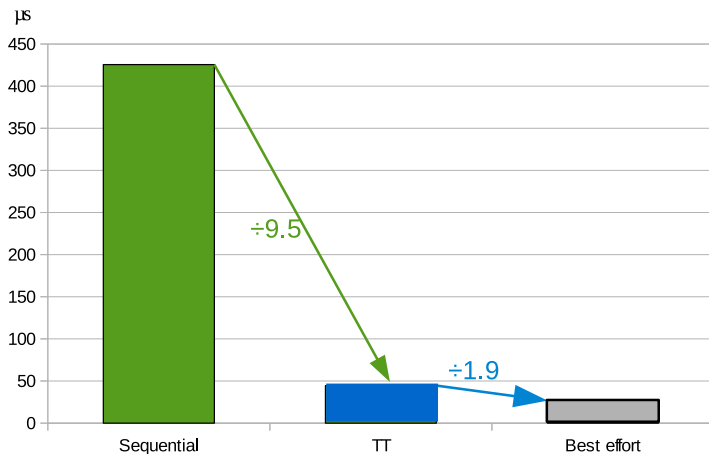


Compute



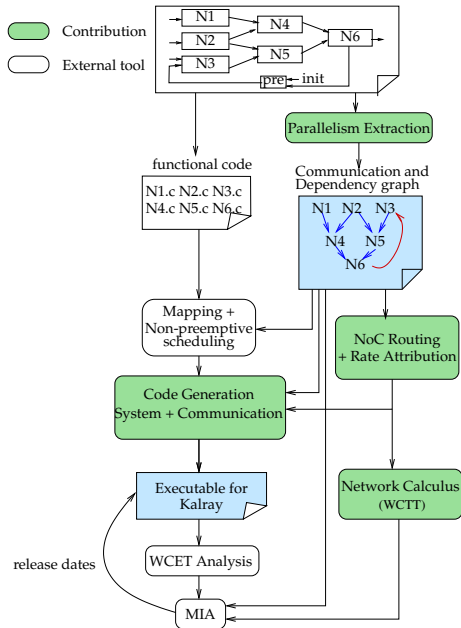
SYNTHETIC BENCHMARK ON 64 CORES

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

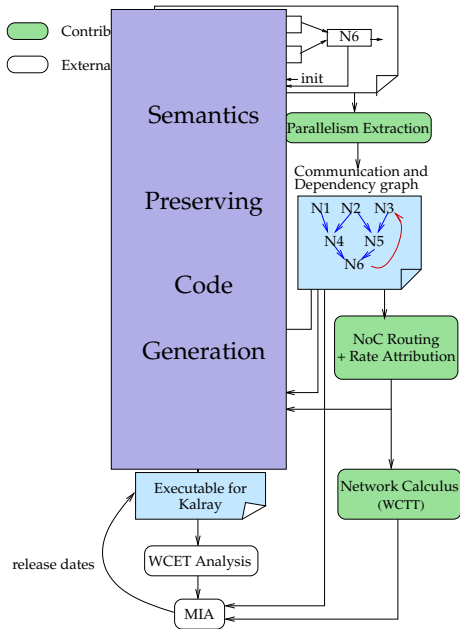


- ▶ 54% of WCRT for functional code
- ▶ 46% of WCRT for communication and system code

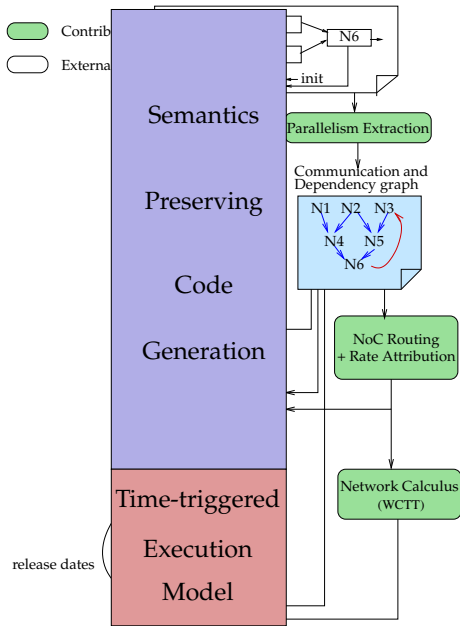
CONCLUSION



CONCLUSION

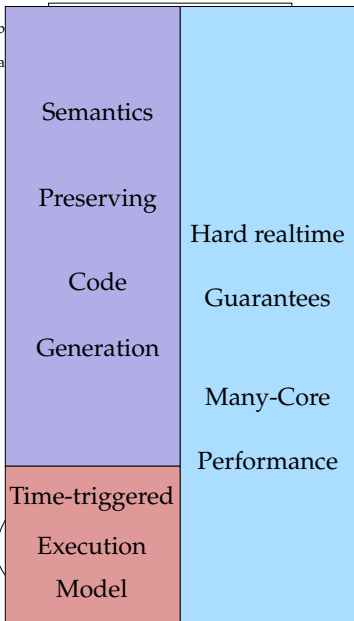


CONCLUSION



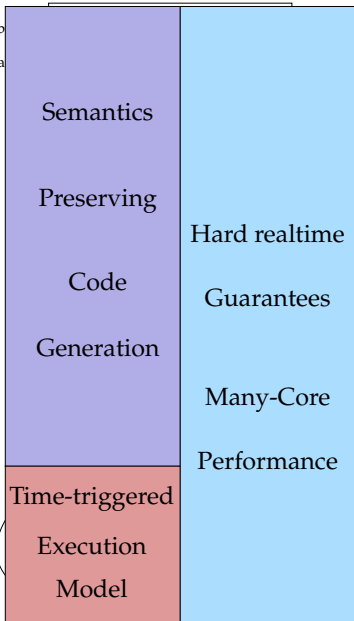
CONCLUSION

- Contrib
- External



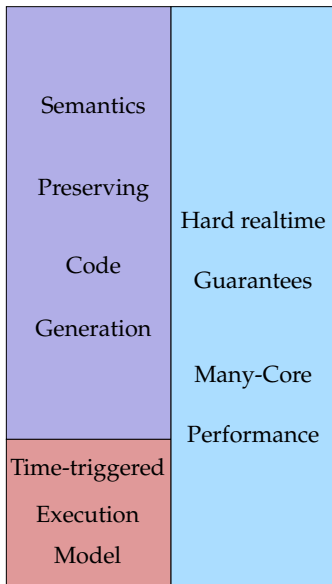
CONCLUSION

- Contrib
- External

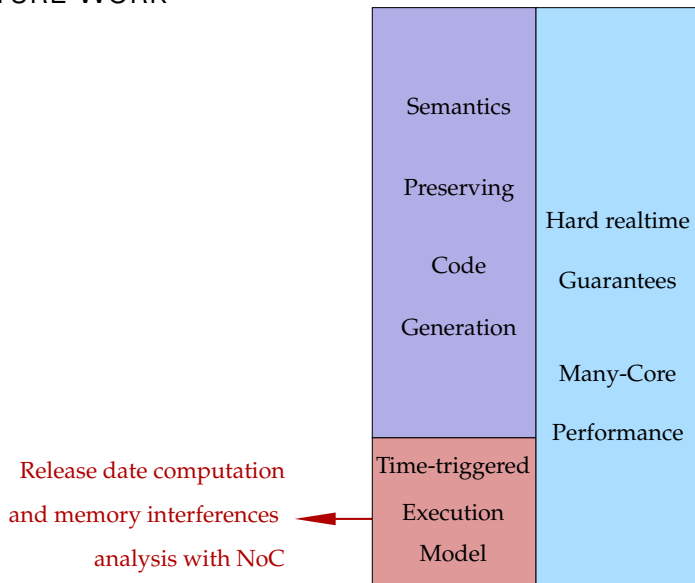


Bridge between academic and industry

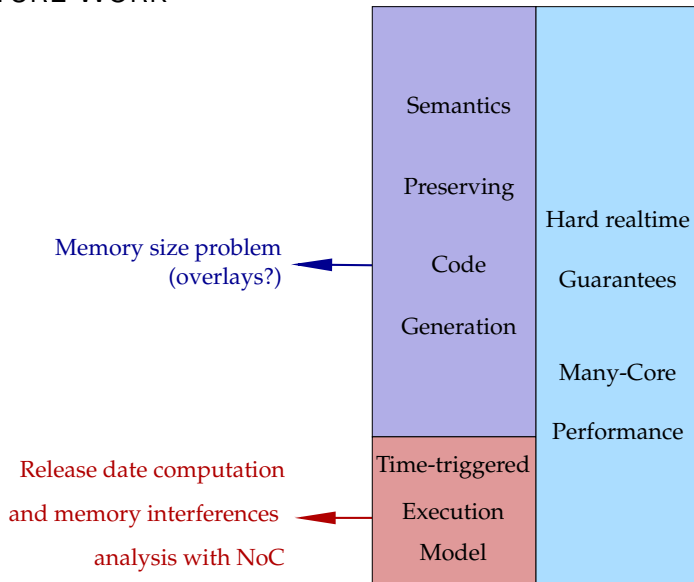
FUTURE WORK



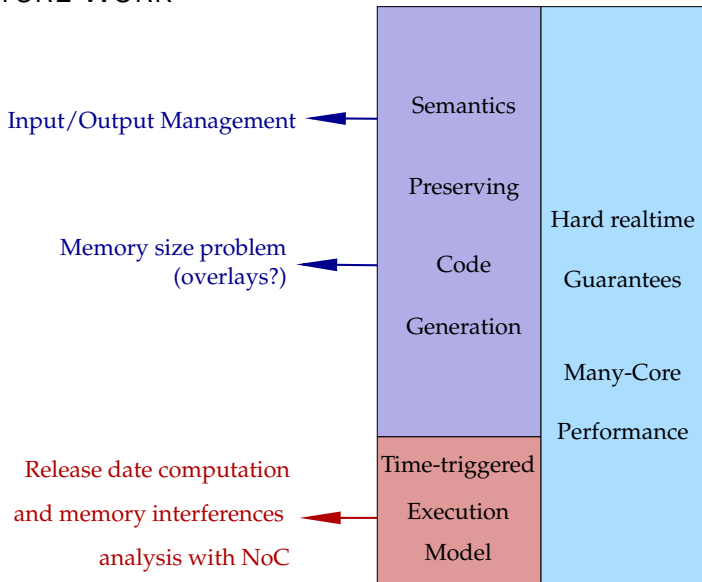
FUTURE WORK



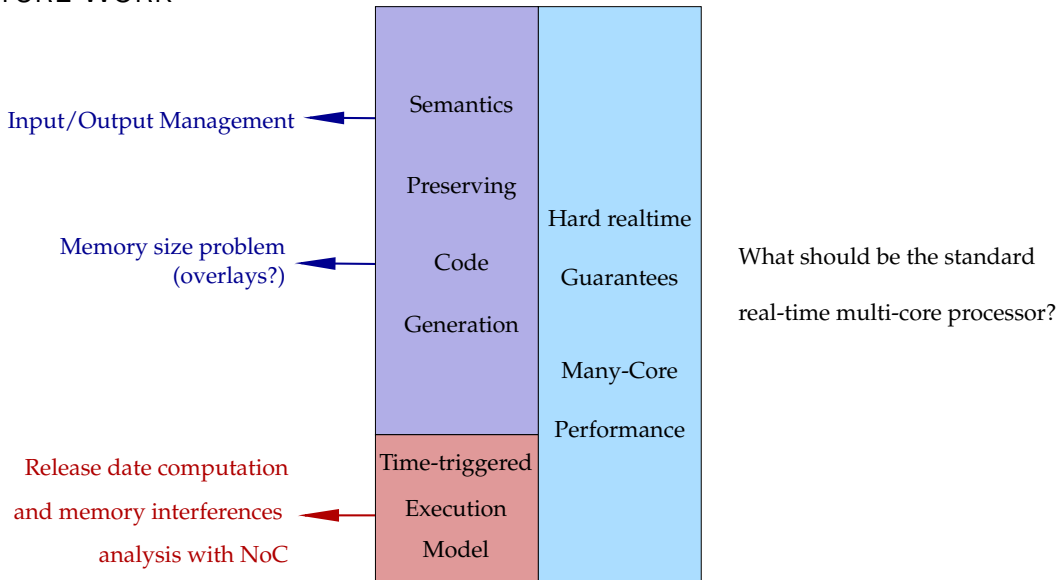
FUTURE WORK



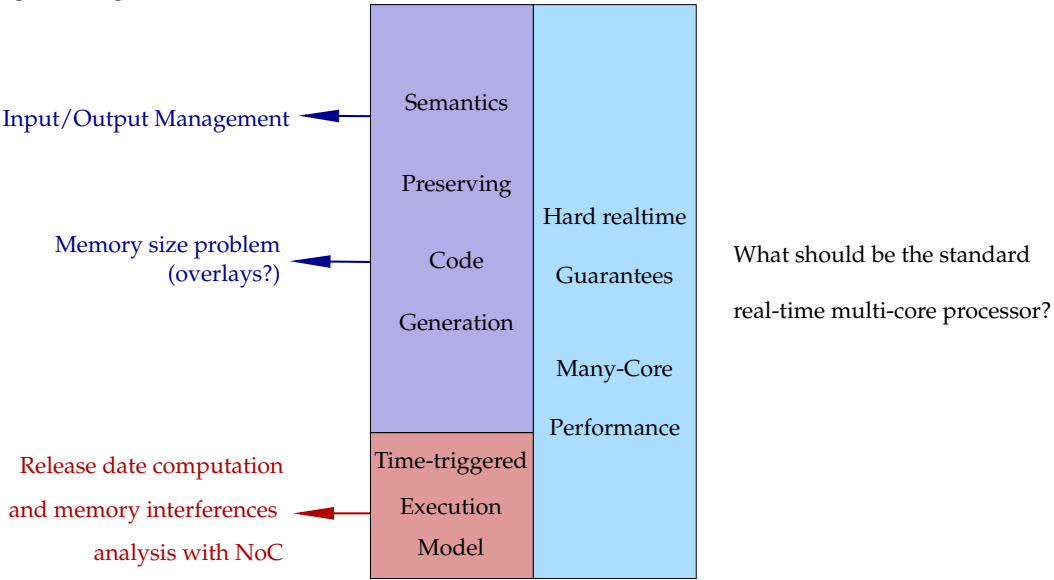
FUTURE WORK



FUTURE WORK



FUTURE WORK



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

Graillat A., Rihani H., Maiza C., Moy M., Raymond P., Dupont de Dinechin B., *Real-Time Systems Journal*

Implementation Framework for Real-Time Data-Flow Synchronous Programs on Many-Cores.

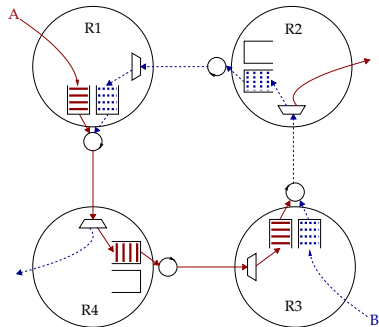
Graillat A, Maiza C., Moy M., Raymond P., Dupont de Dinechin B., *DATE 2019*

Response Time Analysis of Dataflow Applications on a Many-Core Processor with Shared-Memory and Network-on-Chip.

REFERENCES

Backup

DEADLOCK IN WORMHOLE NETWORKS



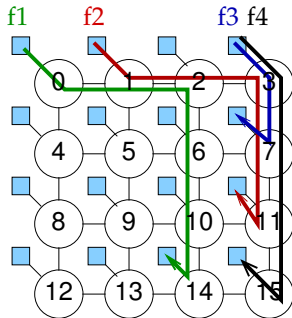
- This instance of flows deadlocks
- A wormhole packet is “spread” along the route
- Links 1-4 and 3-2 are shared
- A holds 1-4 but waits for 3-2
- B holds 3-2 but waits for 1-4

- 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 ρ_i
- Valid: for each link,

$$\sum_i \rho_i \leq 1 \text{ flit/cycle}$$
- Fair attribution: max-min fairness [?]: “cannot increase a rate without decreasing an already smaller (or equal) rate”



■ cluster
○ router

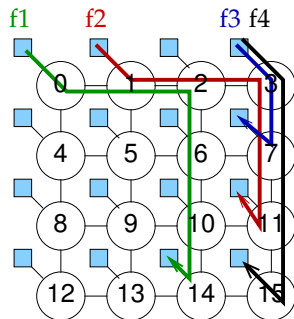
MAX MIN FAIR RATE ATTRIBUTION

- Rate limiter configuration to avoid buffer overflow
- Rate of a flow f_i noted ρ_i
- Valid: for each link,

$$\sum_i \rho_i \leq 1 \text{ flit/cycle}$$
- Fair attribution: max-min fairness [?]: “cannot increase a rate without decreasing an already smaller (or equal) rate”

Example 1:

$$f1 = \frac{1}{2}, f2 = \frac{1}{2}, f3 = \frac{1}{4}, f4 = \frac{1}{4}$$



■ cluster
○ router

MAX MIN FAIR RATE ATTRIBUTION

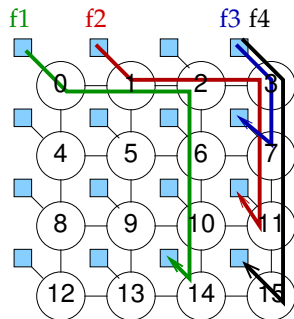
- Rate limiter configuration to avoid buffer overflow
- Rate of a flow f_i noted ρ_i
- Valid: for each link,

$$\sum_i \rho_i \leq 1 \text{ flit/cycle}$$
- Fair attribution: max-min fairness [?]: “cannot increase a rate without decreasing an already smaller (or equal) rate”

Example 1:

$$f1 = \frac{1}{2}, f2 = \frac{1}{2}, f3 = \frac{1}{4}, f4 = \frac{1}{4}$$

→ Valid **but not** max-min fair (since increasing $f3$ or $f4$ can be done by reducing the greater $f2$)



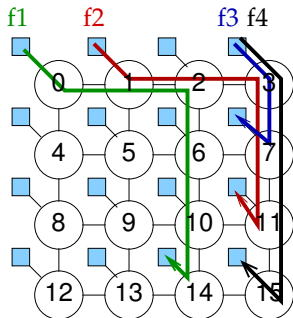
■ cluster

○ router

MAX MIN FAIR RATE ATTRIBUTION

- ▶ Rate limiter configuration to avoid buffer overflow
- ▶ Rate of a flow f_i noted ρ_i
- ▶ Valid: for each link,

$$\sum_i \rho_i \leq 1 \text{ flit/cycle}$$
- ▶ Fair attribution: max-min fairness [?]: “cannot increase a rate without decreasing an already smaller (or equal) rate”



■ cluster
○ router

Example 1:

$$f1 = \frac{1}{2}, f2 = \frac{1}{2}, f3 = \frac{1}{4}, f4 = \frac{1}{4}$$

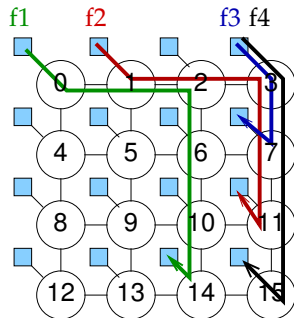
→ Valid **but not** max-min fair (since increasing $f3$ or $f4$ can be done by reducing the greater $f2$)

Example 2:

$$f1 = \frac{2}{3}, f2 = \frac{1}{3}, f3 = \frac{1}{3}, f4 = \frac{1}{3}$$

MAX MIN FAIR RATE ATTRIBUTION

- ▶ Rate limiter configuration to avoid buffer overflow
- ▶ Rate of a flow f_i noted ρ_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”



■ cluster
○ router

Example 1:

$$f1 = \frac{1}{2}, f2 = \frac{1}{2}, f3 = \frac{1}{4}, f4 = \frac{1}{4}$$

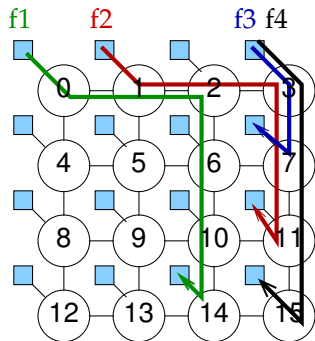
→ Valid **but not** max-min fair (since increasing $f3$ or $f4$ can be done by reducing the greater $f2$)

Example 2:

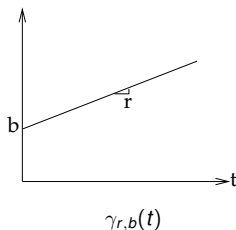
$$f1 = \frac{2}{3}, f2 = \frac{1}{3}, f3 = \frac{1}{3}, f4 = \frac{1}{3}$$

→ Valid and max-min fair (increasing $f2$, $f3$ or $f4$ cannot be done without reducing one of them)

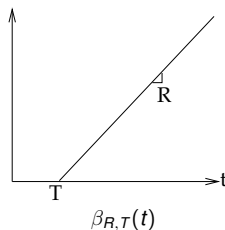
DETERMINISTIC NETWORK CALCULUS (DNC) PRINCIPLE



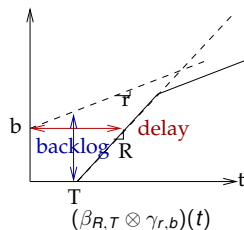
■ cluster
○ router



Arrival Curve



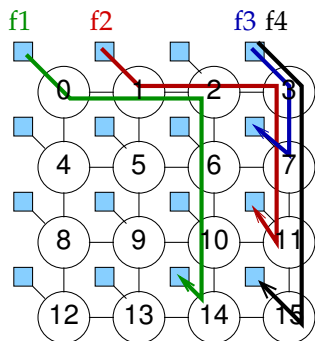
Service Curve



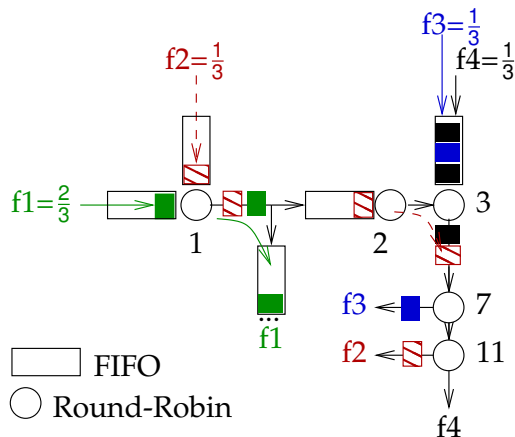
Convolution

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

KALRAY MPPA2 NETWORK-ON-CHIP



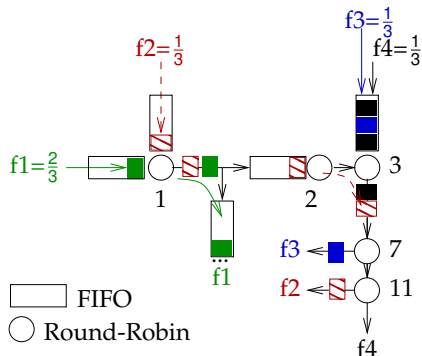
■ cluster
○ router



FIFO
 Round-Robin

Kalray MPPA2 Network Elements

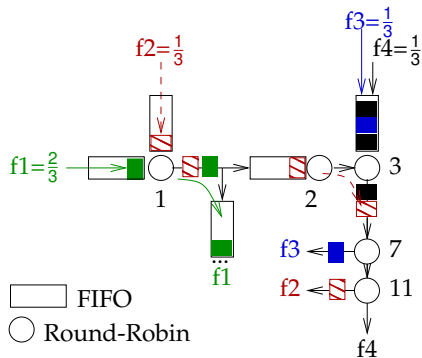
SEPARATED FLOW ANALYSIS (1/2)



Separated Flow Analysis:

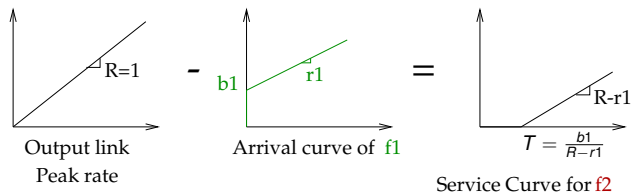
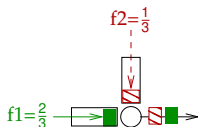
- Compute the service curve of each network element
- Compute the successive arrival curves at each network element
- Convolution of the element service curves \rightarrow network service curve

SEPARATED FLOW ANALYSIS (2/2)



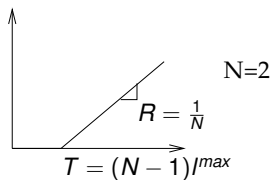
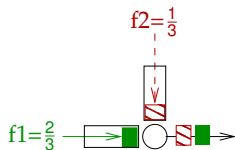
- Service curve offered to $f2$?
- At routers 1, 2, 3, 7 and 11.

BLIND MULTIPLEXING



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

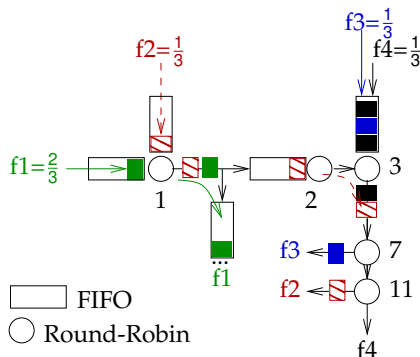
ROUND ROBIN MULTIPLEXING



Service Curve for $f2$

- Packets of size l^{max}
- Restriction: Rate $\leq R$
(not applicable to $f1$)
- Blind multiplexing is the conservative solution.

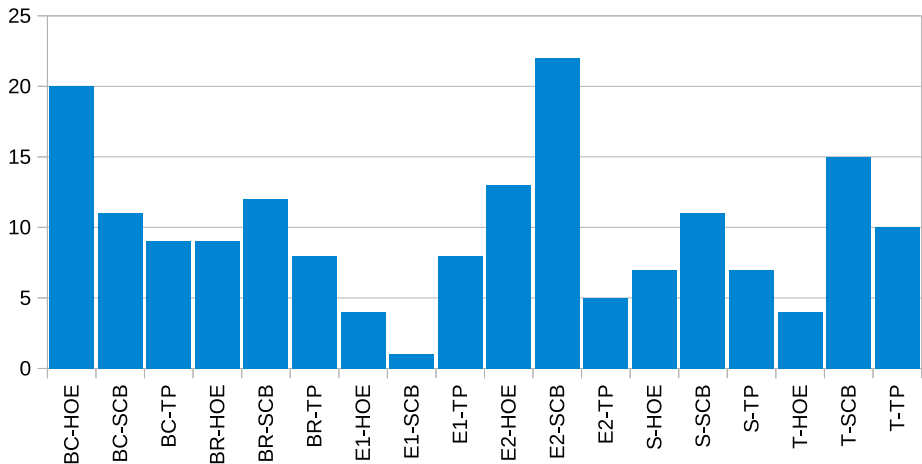
WORST-CASE TRAVERSAL TIME (WCTT): APPLICATION (1/2)



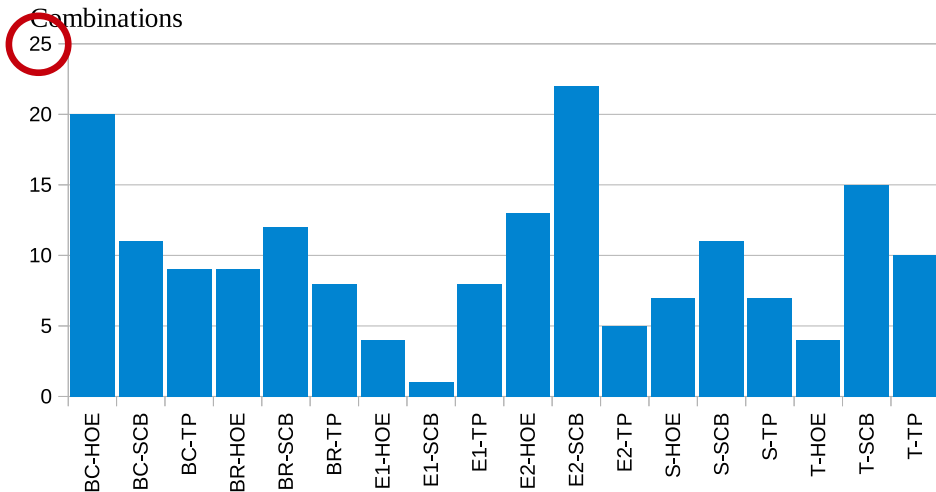
- ▶ Service curve offered to **f2**?
- ▶ Router 1: Round Robin multiplexing ($N=2$)
- ▶ Router 2: non active (alone)
- ▶ Router 3: Round Robin multiplexing ($N=2$, **f3** and **f4** are aggregated with $b_a = \sum_{i \neq 2} b_i$, $r_a = \sum_{i \neq 2} r_i$)
- ▶ Router 7 and 11: non active

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

