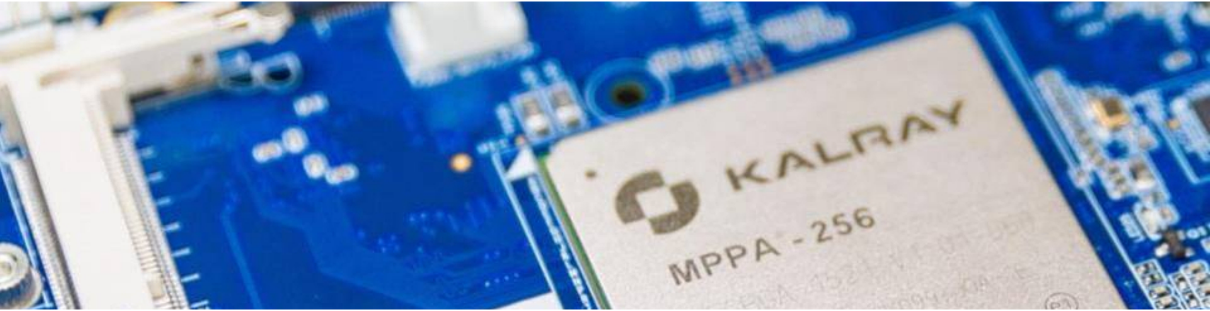
Parallel Code Generation of Synchronous Programs for a Many-core Architecture



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)

Verimag
KALRAY

Thesis defense, November 16th, 2018

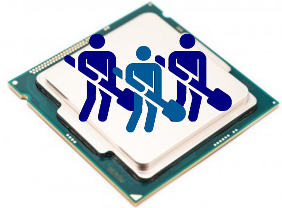
SPACE, WEIGHT, AND POWER

Single-Core

Available since 1971

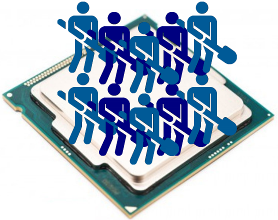
Aircraft: since 1980s
for digital fly-by-wire system

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

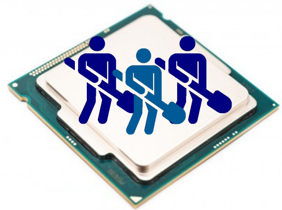
Single-Core

Available since 1971

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

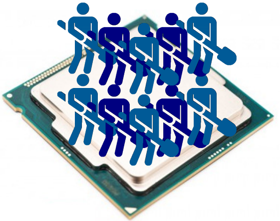
Well-proven for real-time

End of production
No longer sufficient



Multi-Core

Available since 2001



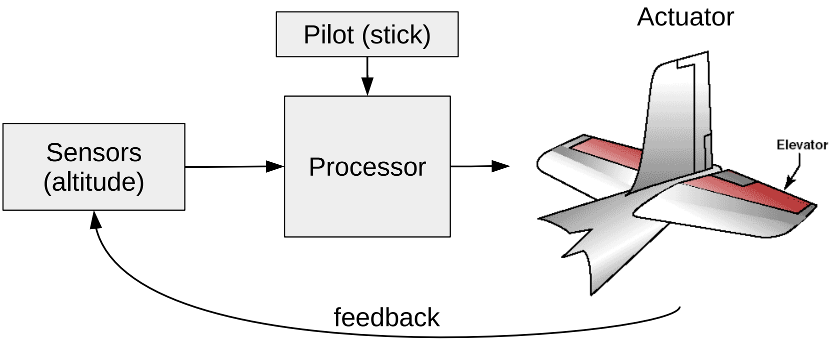
Many-Core

Available since 2007

More functions in one chip
More energy efficient

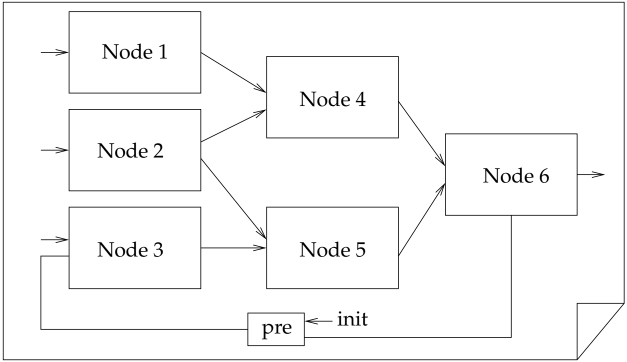
Safety Critical Systems

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

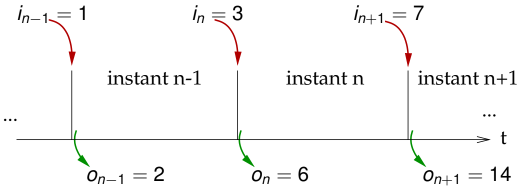
The Synchronous Data-Flow Languages (1/2)

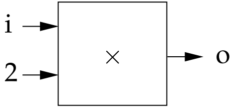


- Lustre (academic), Scade (industrial)
- Network of nodes

The Synchronous Data-Flow Languages (2/2)

Logical

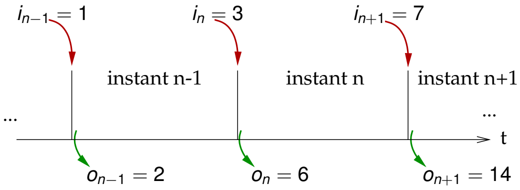


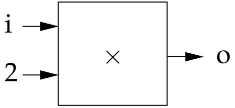


- One execution is called a logical instant

The Synchronous Data-Flow Languages (2/2)

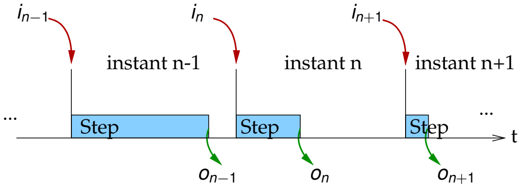
Logical





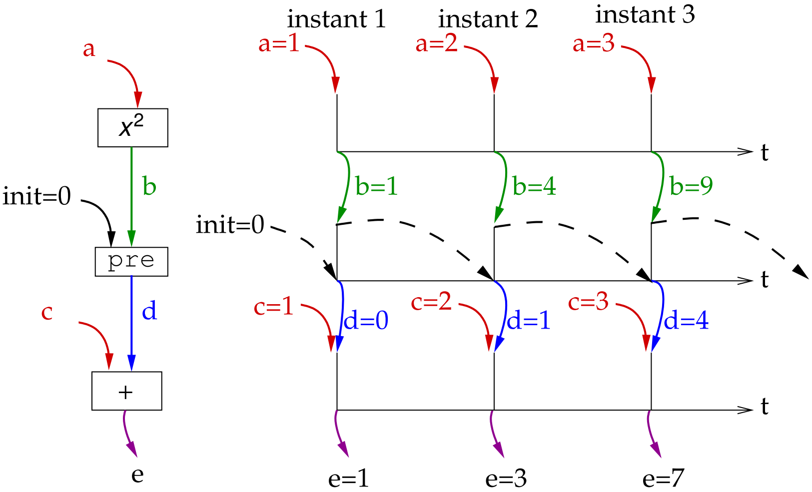
- One execution is called a logical instant

Physical



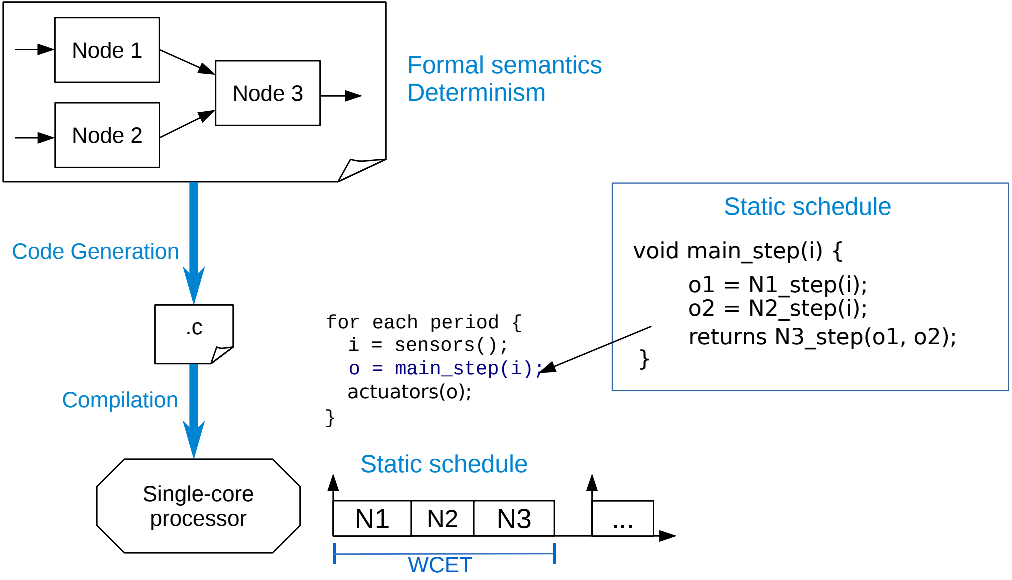
- Requirement: o_n before i_{n+1}.
- Worst-Case Execution Time (WCET)

LUSTRE/SCADE DELAY OPERATOR

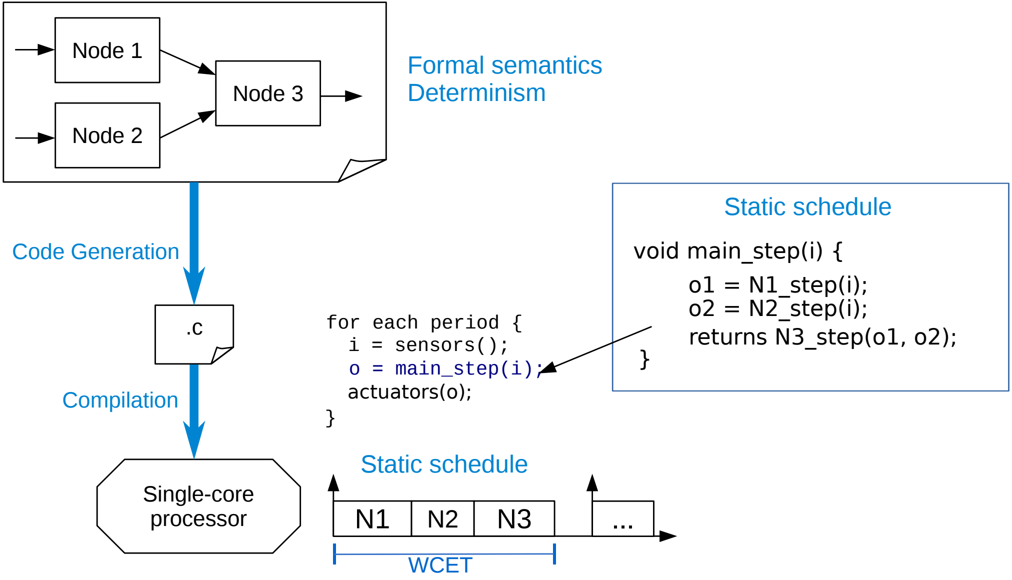


- **pre**vious operator
- Logical/functional delay

Code Generation for Single-Core

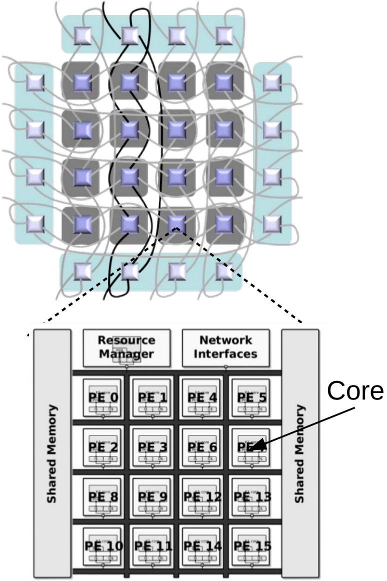


Code Generation for Single-Core



OK for sequential execution. What about parallel?

The Kalray MPPA2



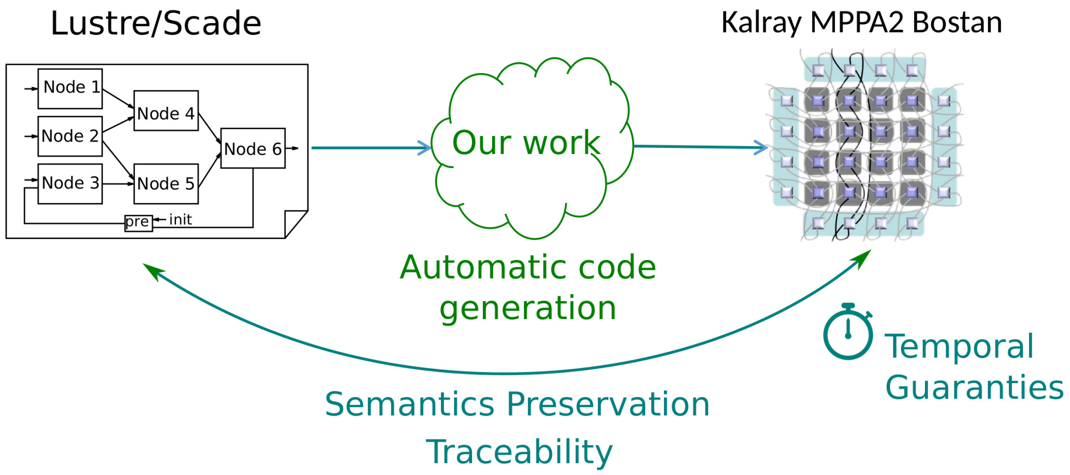
Cluster (=multi-core)

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
Node 1
Node 2
Node 3
Node 4
Node 5
Node 6
pre
init
Our work
Kalray MPPA2 Bostan
Automatic code generation
Temporal Guaranties
Semantics Preservation
Traceability

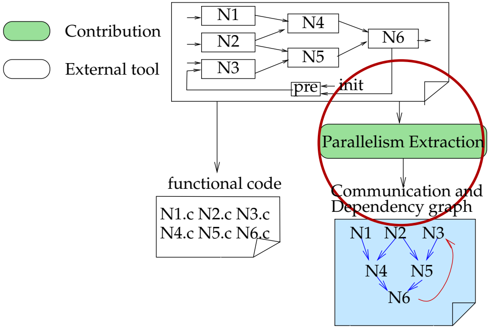
Part I: Semantics Preserving Parallelization

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

Part II: Real-Time Guarantees

Part III: Evaluation and Conclusion

Extraction of Parallelism



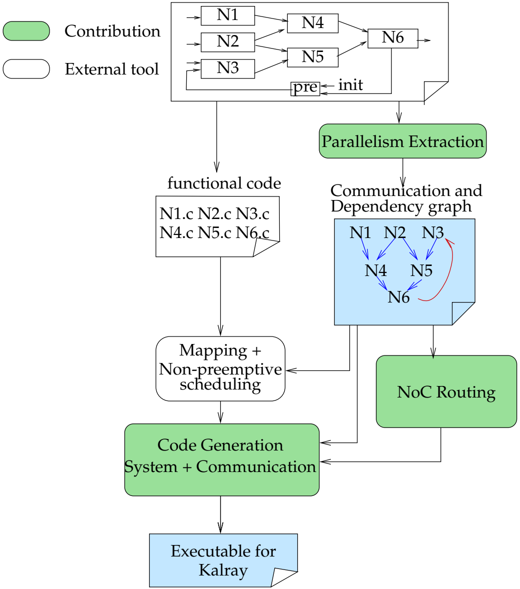
Method 1:

- 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



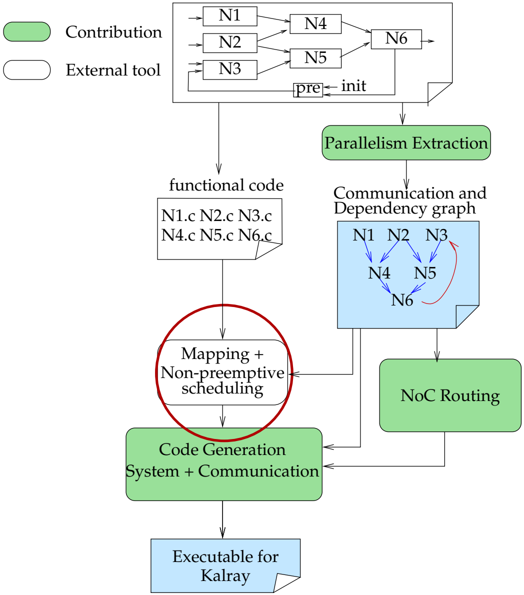
Method 1:

- 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:

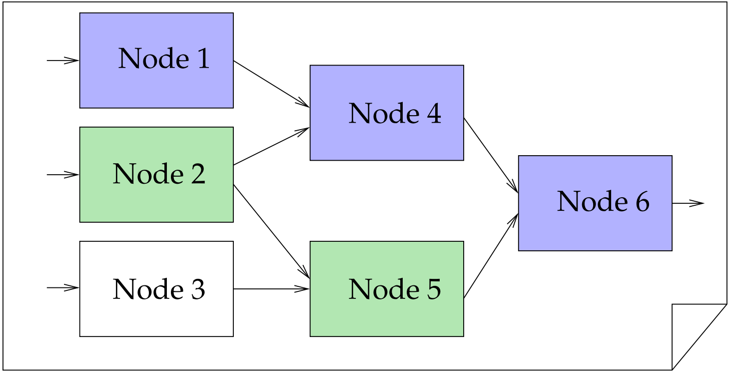
- see manuscript

Mapping / Scheduling



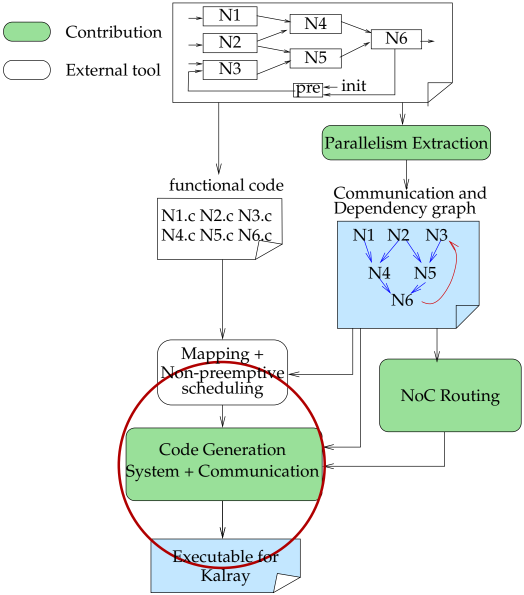
- Need non-preemptive static schedule
- External scheduling tool
- We can easily check the schedule

Mapping/Scheduling: Example

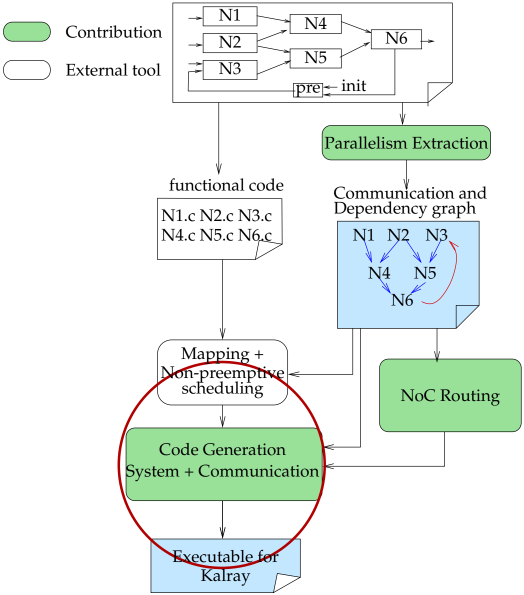


- Core 0: N1 ; N4 ; N6
  Core 1: N2 ; N5
  Core 2: N3
- Schedule checked using the dependency graph

CODE GENERATION



Code Generation

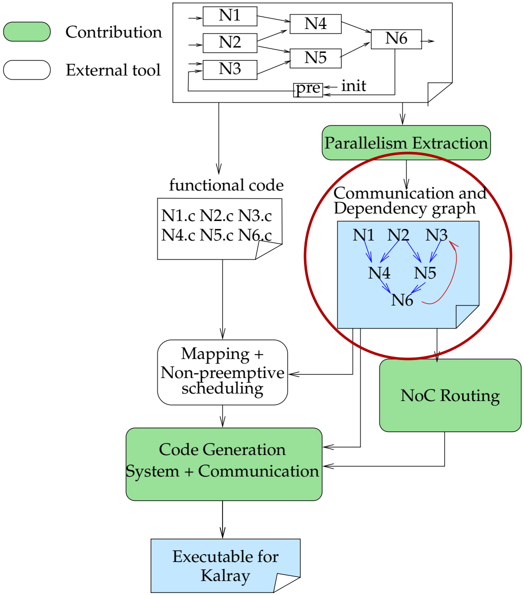


- Dependencies compiled into "wait for input".

Sketch of code for core 0.

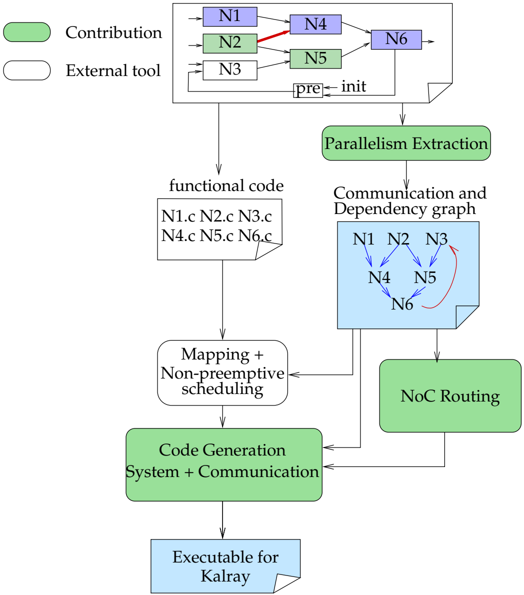
```
for each period{
  wait_inputs_N1(); // N1
  N1_step();
  write_outputs_N1();

  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 (→)

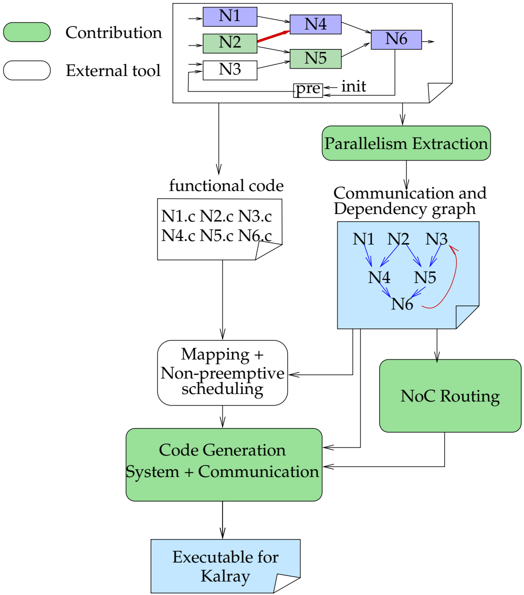
Instantaneous Communication




```
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

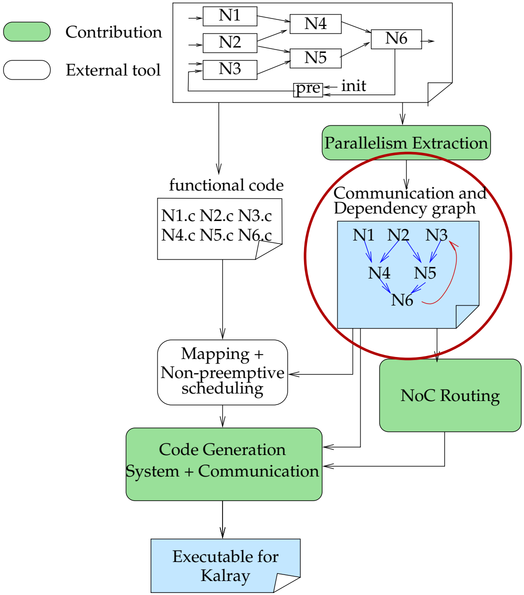
Instantaneous Communication




```
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
}
```

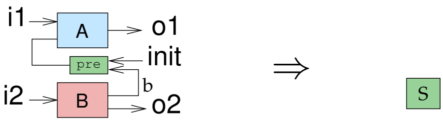
- Efficient hardware synchronization
- Software-based cache coherency

Communication Channels



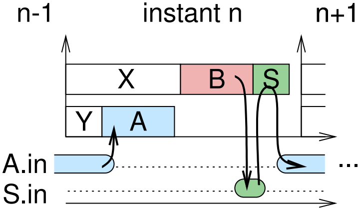
- Two kinds of communications:
  - instantaneous (→)
  - **delayed (→)**

DELAYED COMMUNICATION



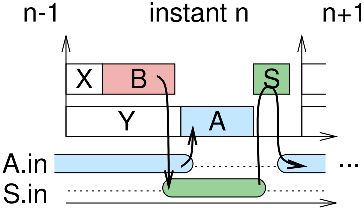
Transformation into a `SWAP` + scheduling constraints.

Scenario 1



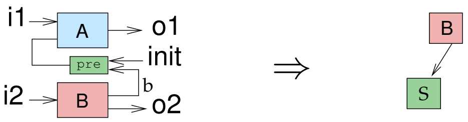
Constraint: B→S

Scenario 2

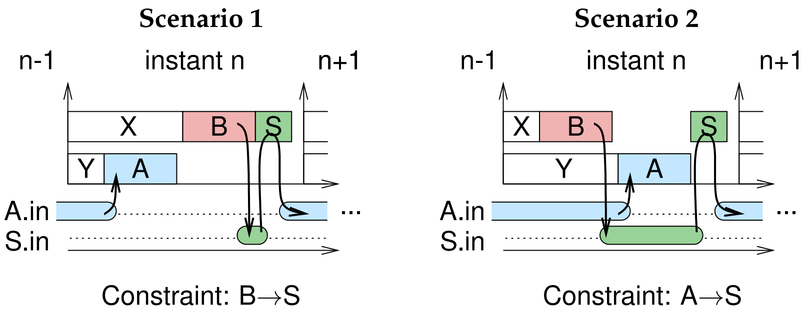


Constraint: A→S

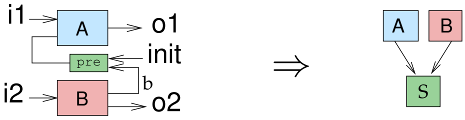
Delayed Communication



Transformation into a `SWAP` + scheduling constraints.

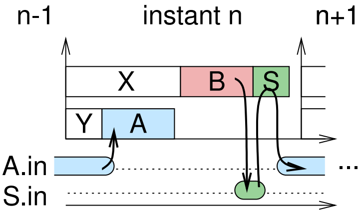


Delayed Communication



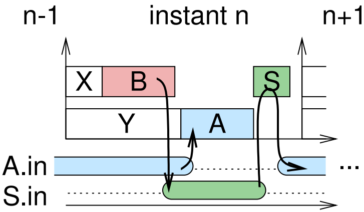
Transformation into a `SWAP` + scheduling constraints.

Scenario 1



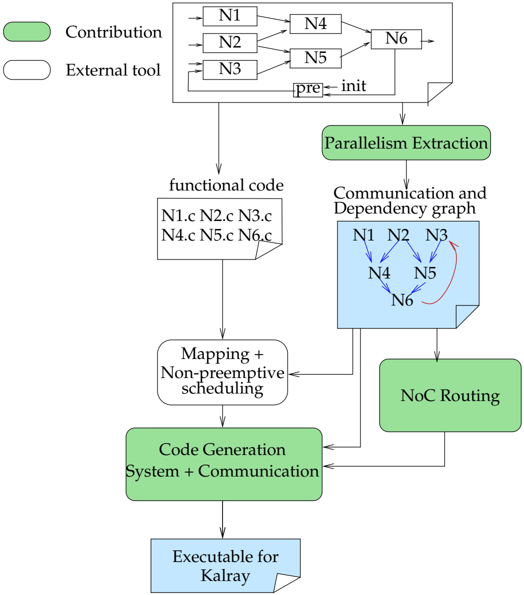
Constraint: B→S

Scenario 2



Constraint: A→S

Conclusion of Part I



- Semantics preserved
- Next step: Temporal Guaranties

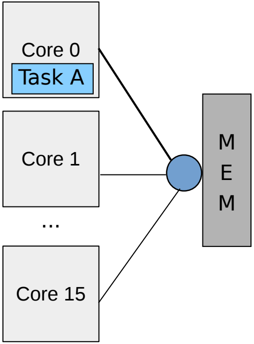
Part I: Semantics Preserving Parallelization

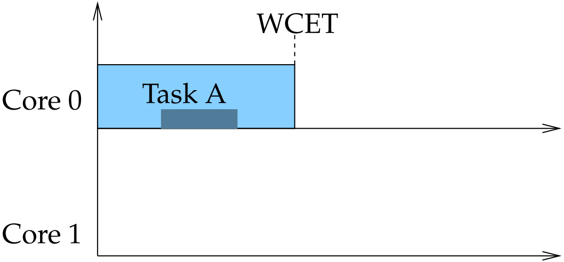
Part II: Real-Time Guarantees

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

Part III: Evaluation and Conclusion

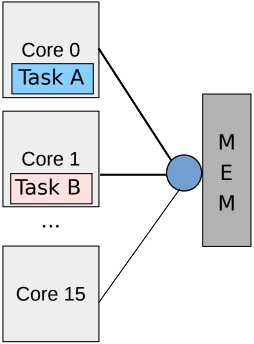
Interference and Reaction Time

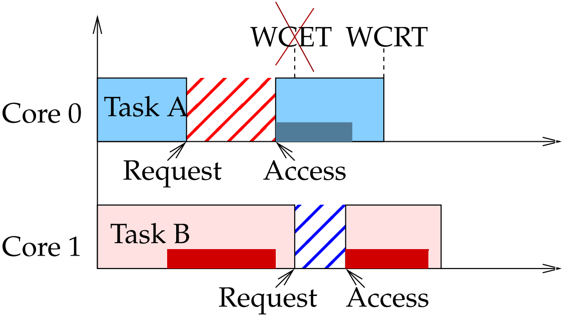




- Single-core: WCET is sufficient

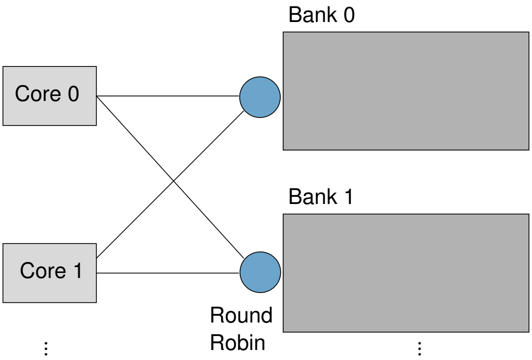
Interference and Reaction Time





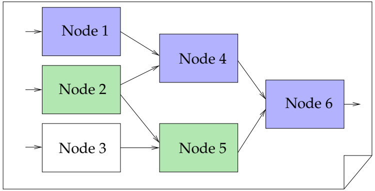
- Single-core: WCET is sufficient
- Multi-core: WCET+interference on shared resources
  = Worst-Case Response Time (WCRT).
- **WCET+interference too pessimistic in the general case**
  ⇒ exploit the execution model in the analysis

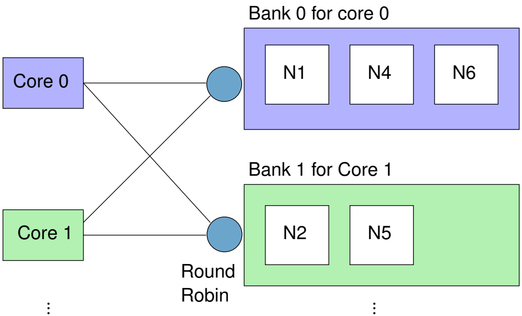
Banked Memory



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

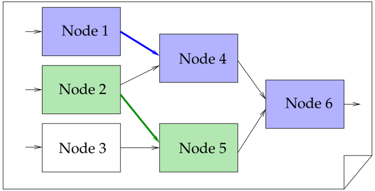
Banked Memory in Kalray MPPA2

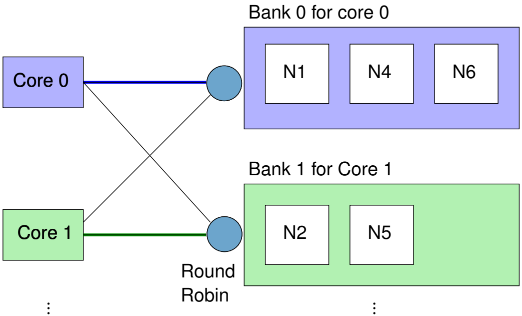




- Choice: Code, input buffer, local variables are mapped in core's bank of the core
- Interference on communication only

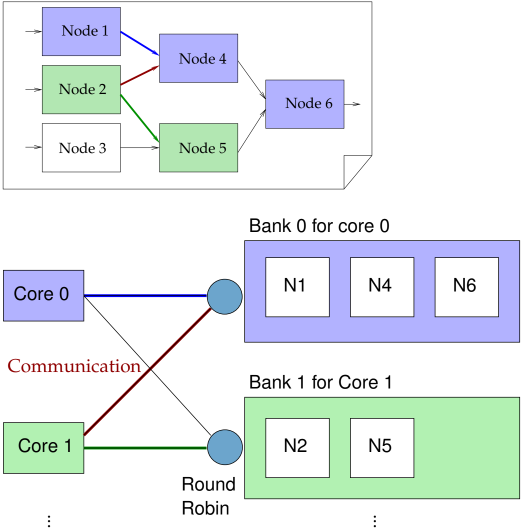
BANKED MEMORY IN KALRAY MPPA2





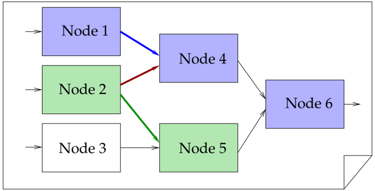
- Choice: Code, input buffer, local variables are mapped in core's bank of the core
- Interference on communication only

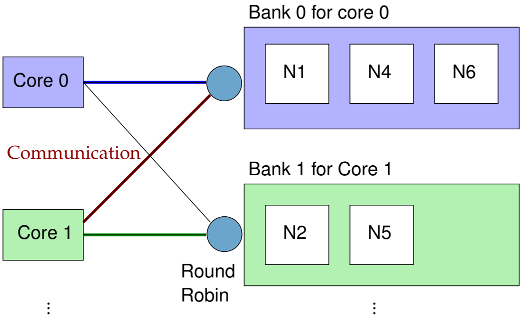
Banked Memory in Kalray MPPA2



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

Banked Memory in Kalray MPPA2

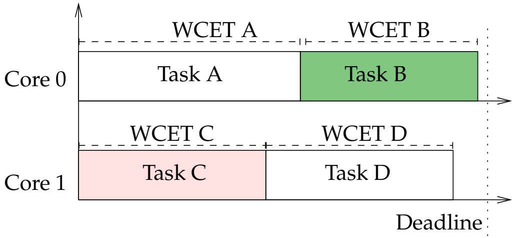




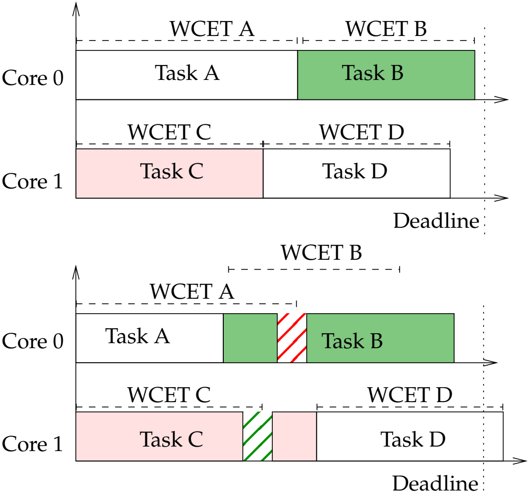
- Choice: Code, input buffer, local variables are mapped in core's bank of the core
- Interference on communication only

How to activate tasks?

Problem With ASAP Execution



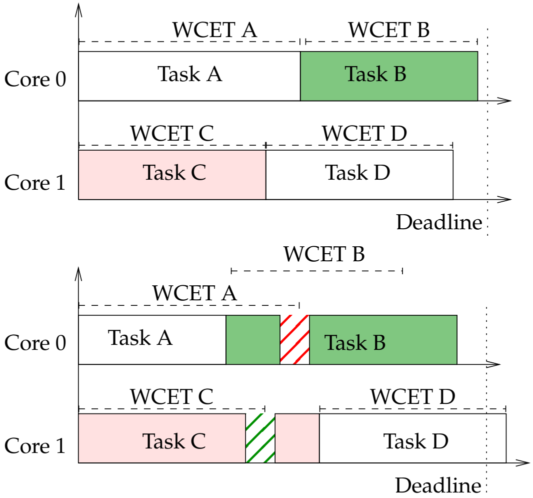
Problem With ASAP Execution



Faster execution of A ⇒ interference between B and C.

Timing Anomaly

Problem With ASAP Execution



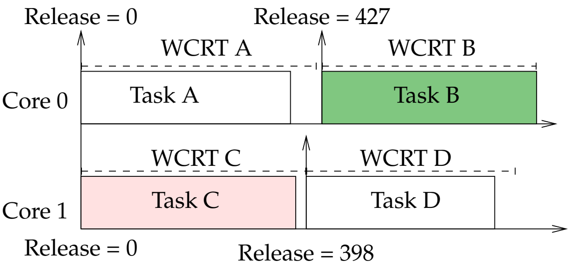
Faster execution of A ⇒ interference between B and C.

Timing Anomaly

Solution: release dates for tasks (time-triggered)

Time-Triggered Execution Model

Principle

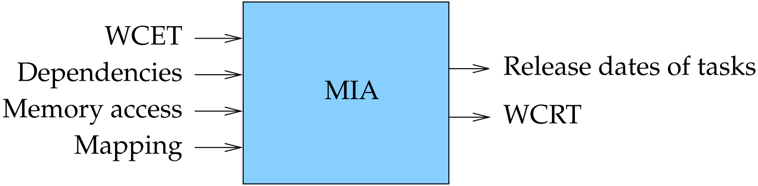


- Compute a static release date when data is guaranteed to be available
- Time-Triggered execution prevents tasks from starting earlier

Implementation

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

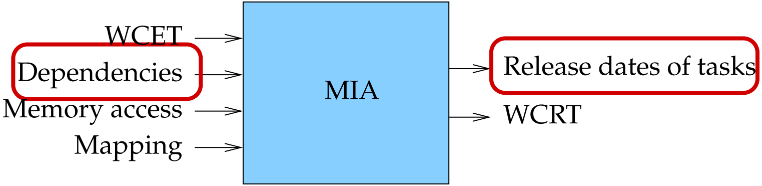
Multi-Core Interference Analysis



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

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

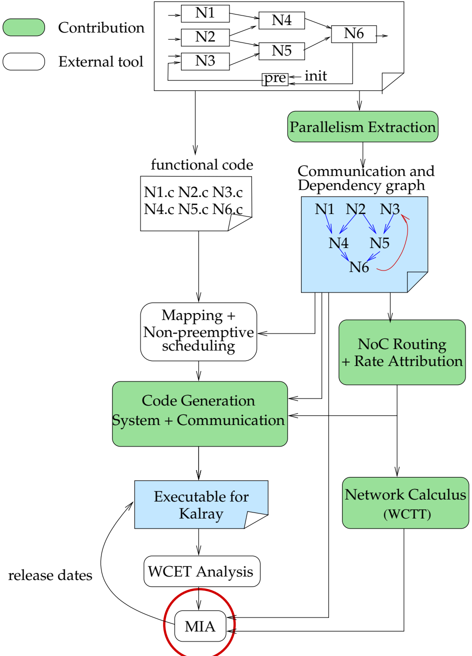
Multi-Core Interference Analysis



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

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

Framework



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

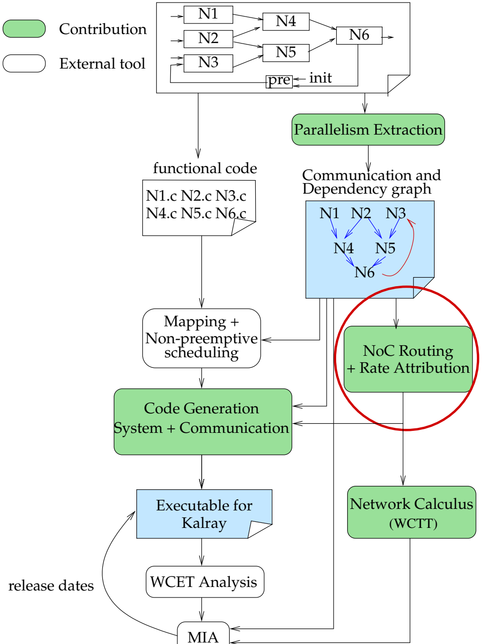
[Submitted: Graillat A., Rihani H., Maiza C., Moy M., Raymond P., Dupont de Dinechin B., *Real-Time Systems Journal*]

Our Execution Model

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

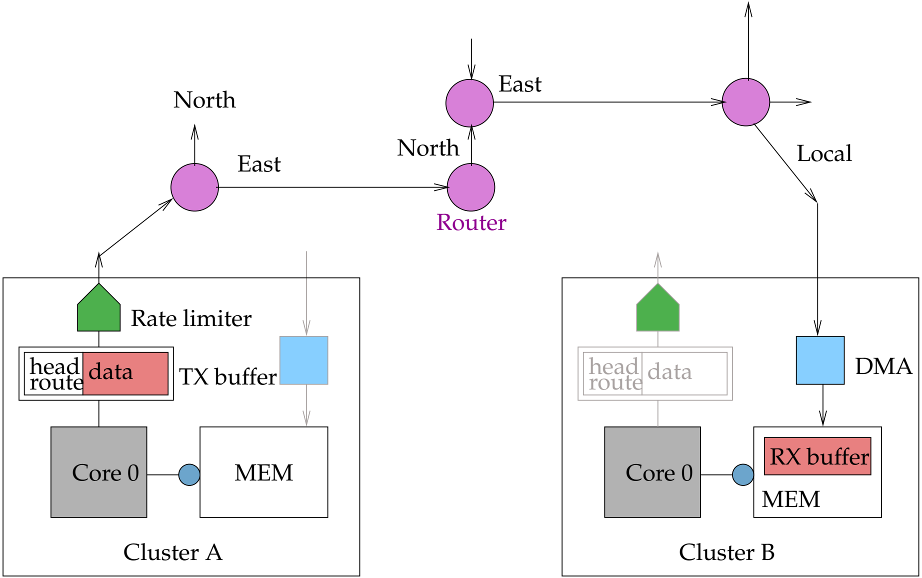
Deterministic and WCRT guarantee

Overview

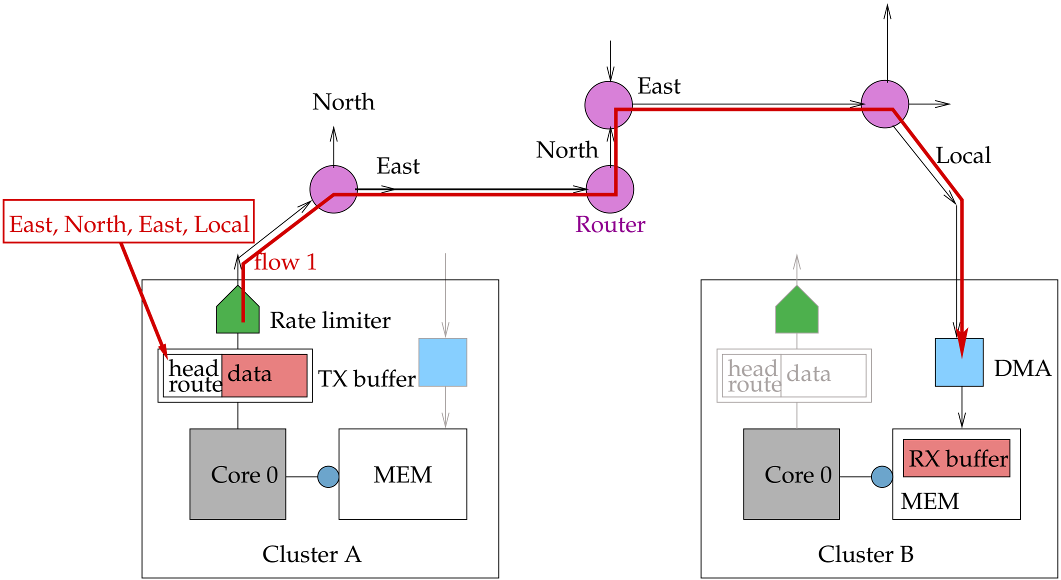


Contribution
External tool
N1
N2
N3
N4
N5
N6
pre
init
Parallelism Extraction
functional code
N1.c N2.c N3.c
N4.c N5.c N6.c
Communication and
Dependency graph
N1 N2 N3
N4 N5
N6
Mapping +
Non-preemptive
scheduling
NoC Routing
+ Rate Attribution
Code Generation
System + Communication
Executable for
Kalray
Network Calculus
(WCTT)
WCET Analysis
release dates
MIA

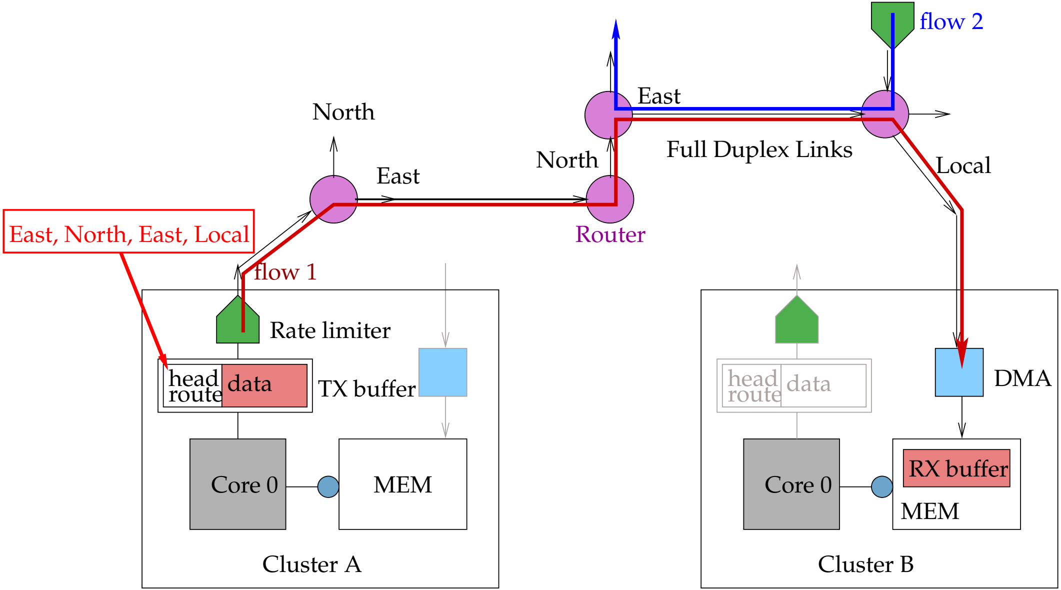
The Kalray MPPA2's NoC



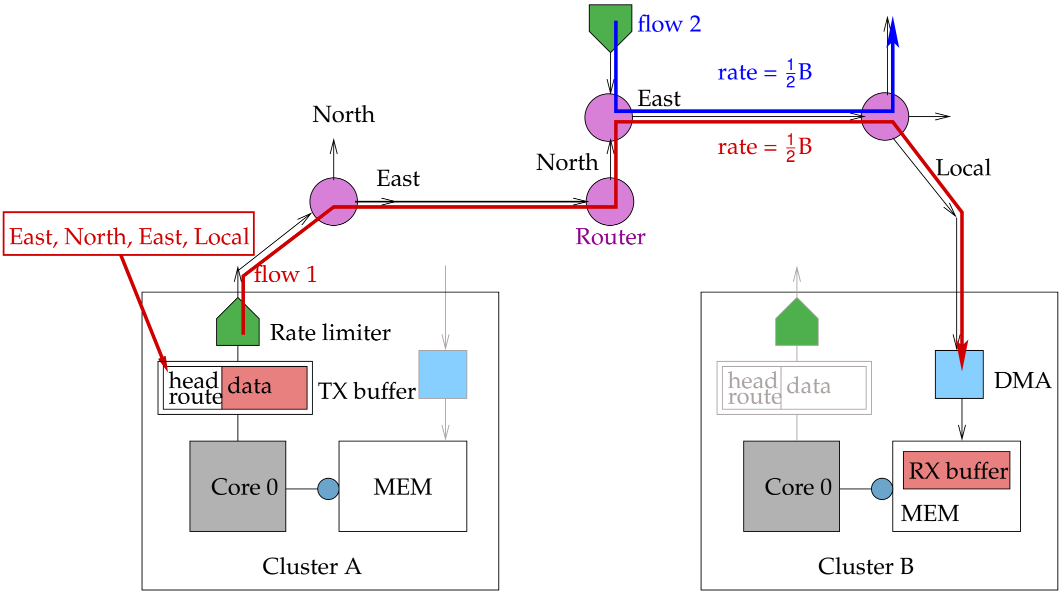
THE KALRAY MPPA2'S NOC



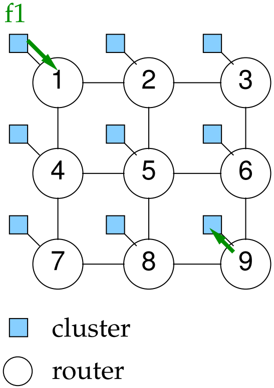
The Kalray MPPA2's NoC



THE KALRAY MPPA2'S NOC



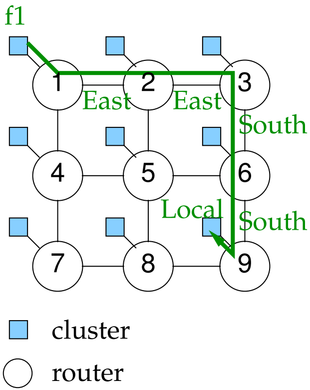
Real-Time Guarantees with Network-on-Chip



1. Routing
   Route = sequence of directions computed by sender

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

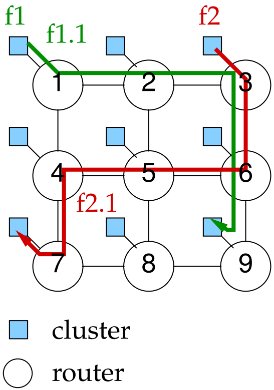
Real-Time Guarantees with Network-on-Chip



1. Routing
   Route = sequence of directions computed by sender

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

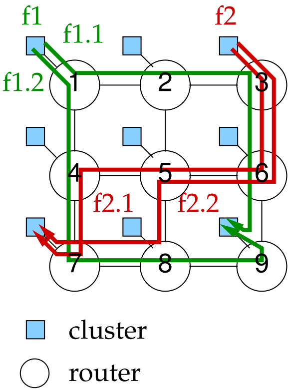
Real-Time Guarantees with Network-on-Chip



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

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

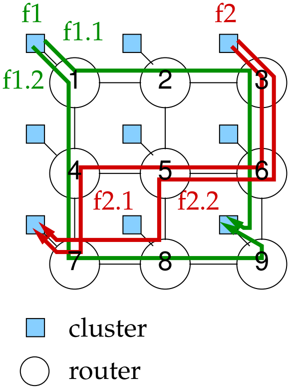
Real-Time Guarantees with Network-on-Chip



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*]

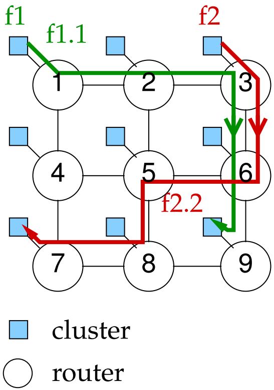
Real-Time Guarantees with Network-on-Chip



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*]

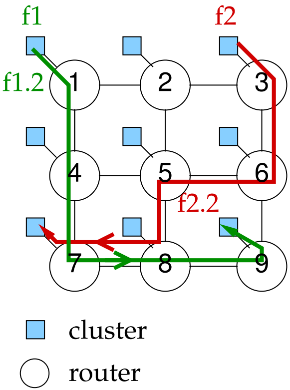
Real-Time Guarantees with Network-on-Chip



[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},

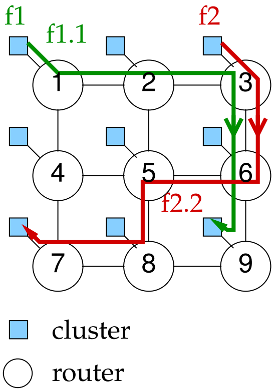
Real-Time Guarantees with Network-on-Chip



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*]

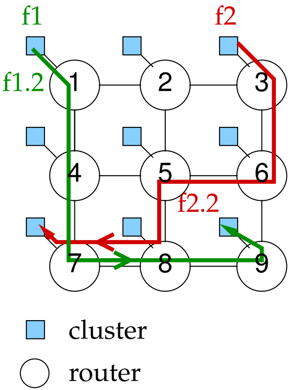
Real-Time Guarantees with Network-on-Chip



[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},

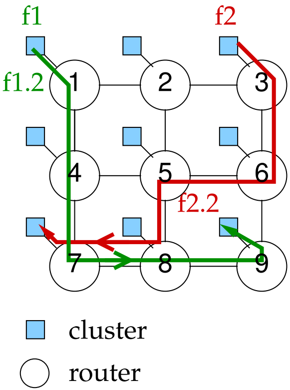
Real-Time Guarantees with Network-on-Chip



[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}...

Real-Time Guarantees with Network-on-Chip

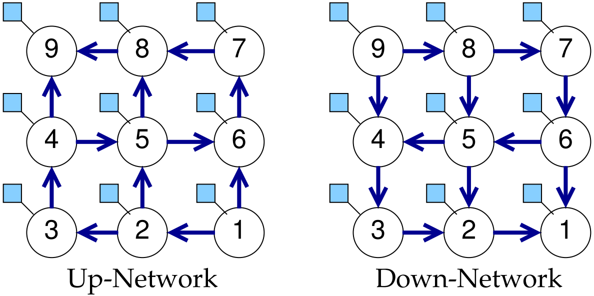


[Dupont de Dinechin B., Graillat A., *NoCArc 2017*]

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

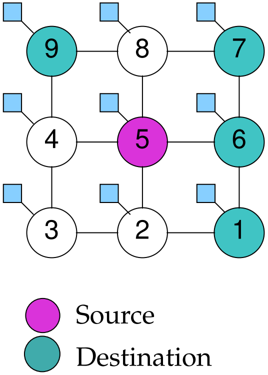
1. Unicast Routing

Hamiltonian Routing



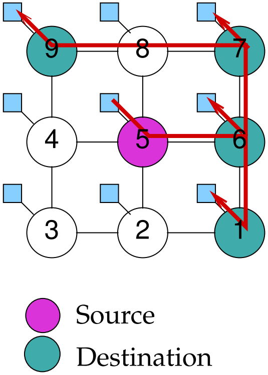
- Deadlock-free
- Path-Diversity

1. Multicast Routing Problem



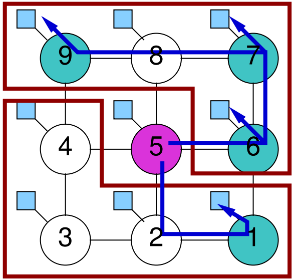
- Deadlock-free heuristic to "travelling salesman problem":
- One route? Tree is no possible with Kalray architecture
- Hamiltonian: minimum of two routes

1. Multicast Routing Problem



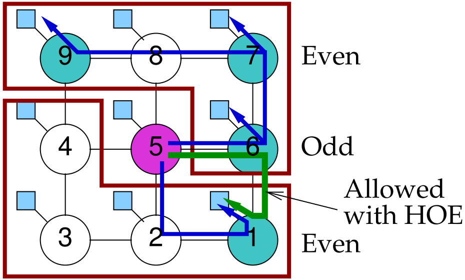
- Deadlock-free heuristic to “travelling salesman problem”:
- One route? Tree is no possible with Kalray architecture
- Hamiltonian: minimum of two routes

1. Multicast Routing: Hamiltonian



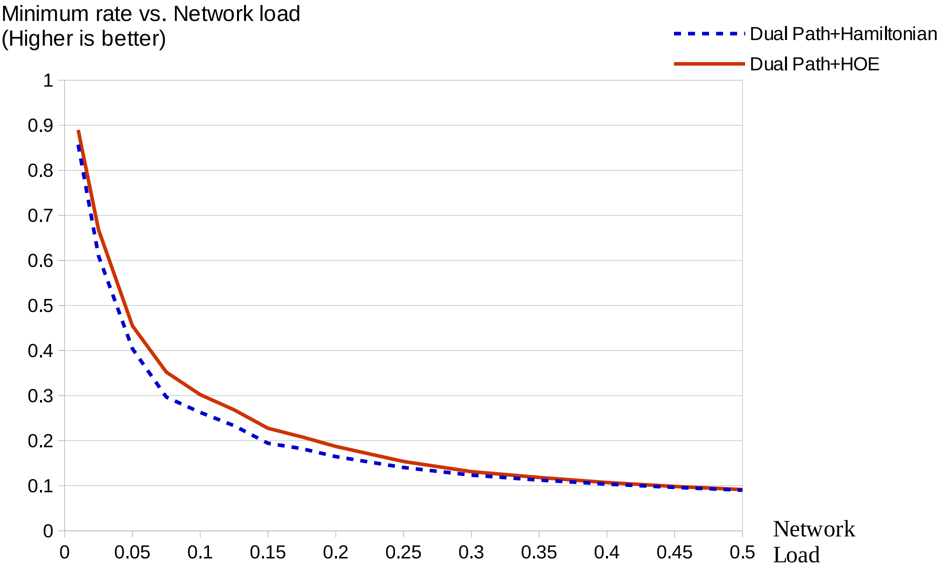
- Dual-Path + Hamiltonian without path diversity (Lin et al., 1992)
- Small path diversity, we can do better.

1. Multicast Routing: HOE

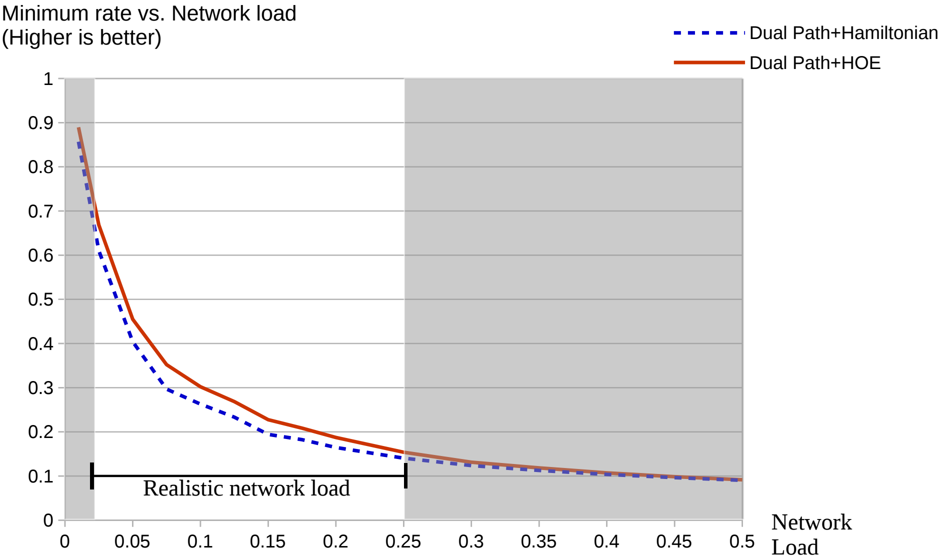


- Hamiltonian Odd Even (Bahrebar et al.) routing: allows some non-Hamiltonian paths
- Our choice: Dual-Path + HOE

Evaluation of Deadlock-free Multicast Routing

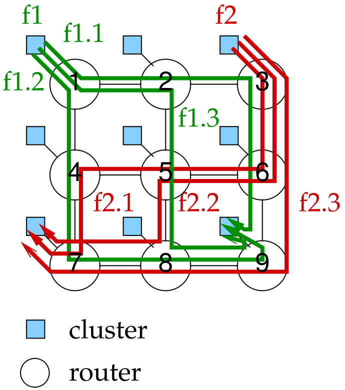


Evaluation of Deadlock-free Multicast Routing



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

2. Route Selection



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

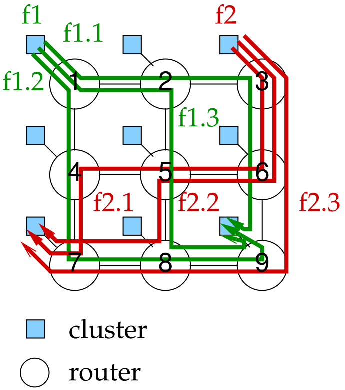
- Input: set of route for each flow
- Output: one route per flow
- Maximize fairness

- **Naive exploration**: Enumeration all combinations, 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



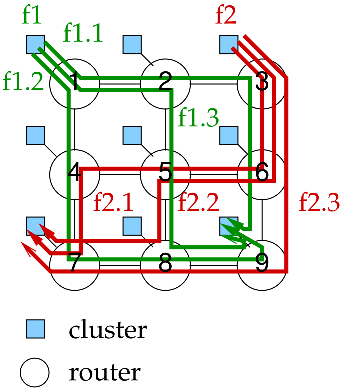
[Boyer M., Dupont de Dinechin B., Graillat A., Havet L, *ERTS 2018*]

- 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



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

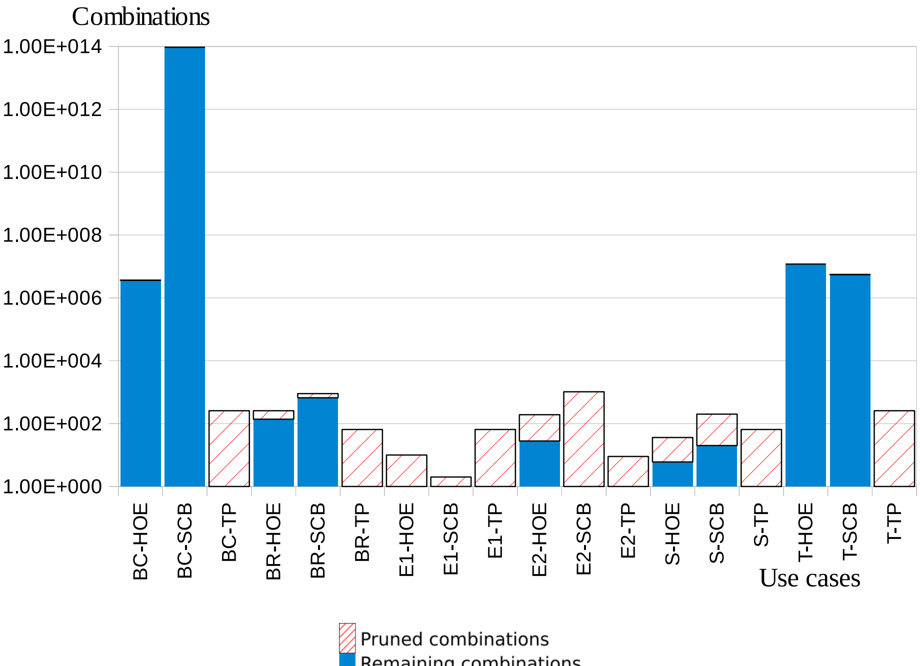
- Input: set of route for each flow
- Output: one route per flow
- Maximize fairness

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

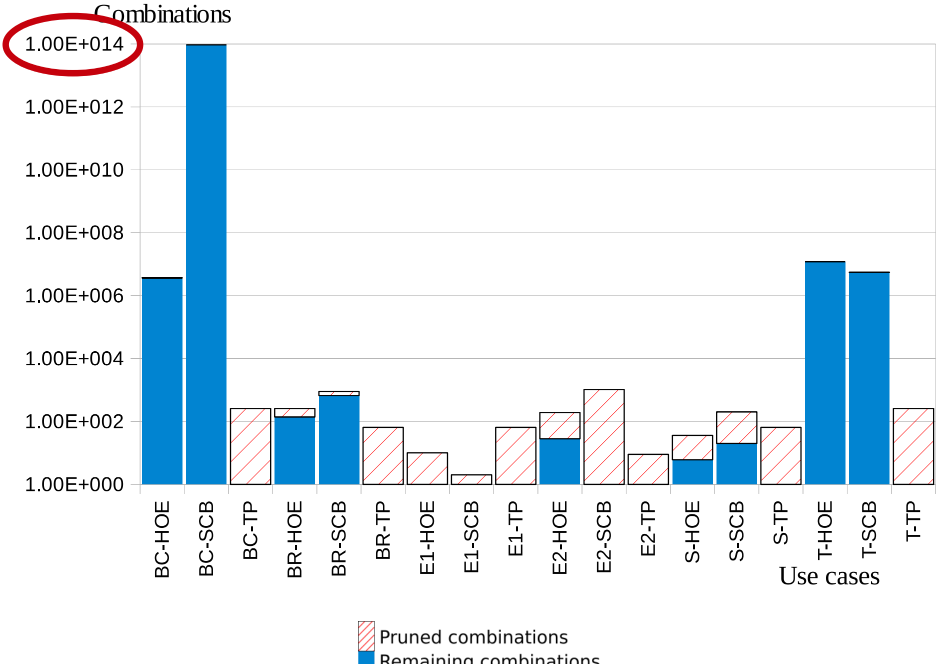
	f2.1	f2.2	f2.3
f1.1			
f1.2		**1.0, 1.0**	**1.0, 1.0**
f1.3	**1.0, 1.0**		**1.0, 1.0**

Enumeration of 4 combinations

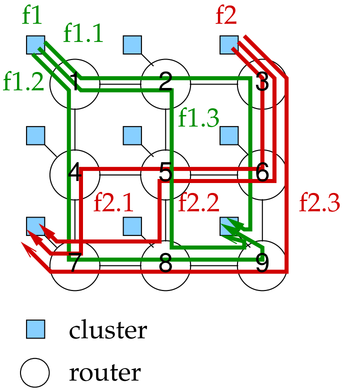
Exploration vs. Our Exploration with Pruning



Exploration vs. Our Exploration with Pruning



2. Route Selection

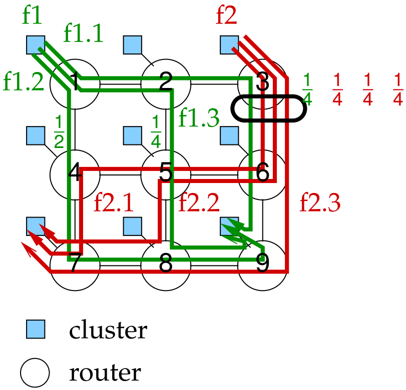


[Boyer M., Dupont de Dinechin B., Graillat A., Havet L, *ERTS 2018*]

- 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. Route Selection

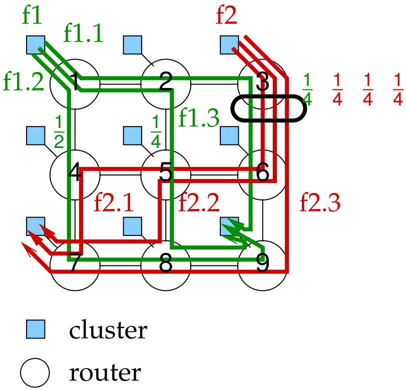


[Boyer M., Dupont de Dinechin B., Graillat A., Havet L, *ERTS 2018*]

- 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. Route Selection

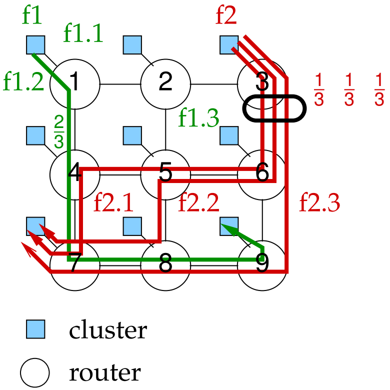


[Boyer M., Dupont de Dinechin B., Graillat A., Havet L, *ERTS 2018*]

- 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

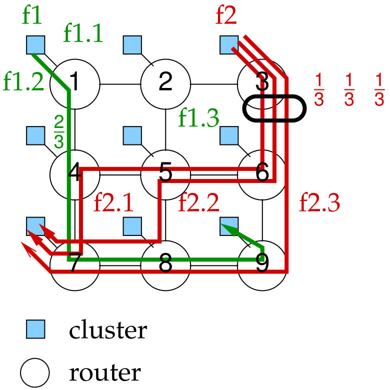


[Boyer M., Dupont de Dinechin B., Graillat A., Havet L, *ERTS 2018*]

- 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

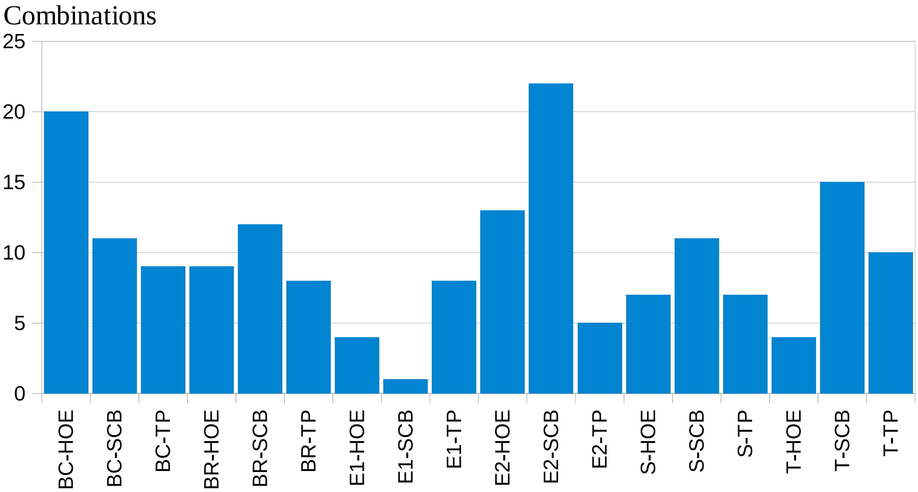


- Input: set of route for each flow
- Output: one route per flow
- Maximize fairness

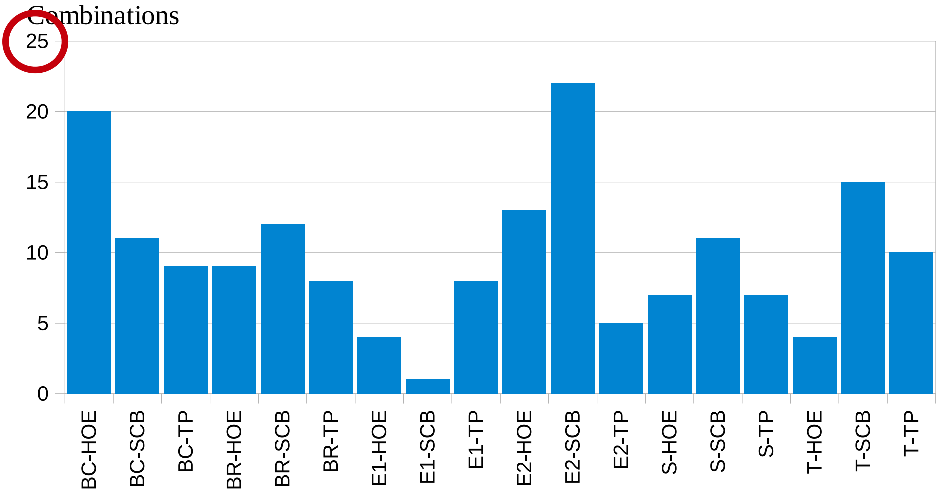
- **Naive exploration**: Enumeration all combinations (9 combinations)
- **Our Exploration with pruning**: Ignore non-minimal bottleneck (4 combinations)
- **Our LP-based Heuristic**:
  - Consider all alternative routes at once
    1. Attribute fair rates
    2. Remove smallest alternative routes
  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

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

Evaluation of Our LP-based Heuristic

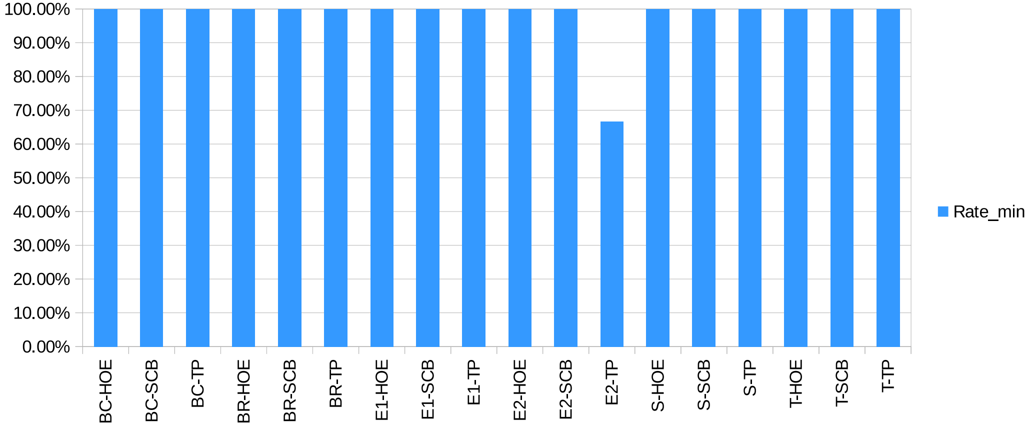


Evaluation of Our LP-based Heuristic



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

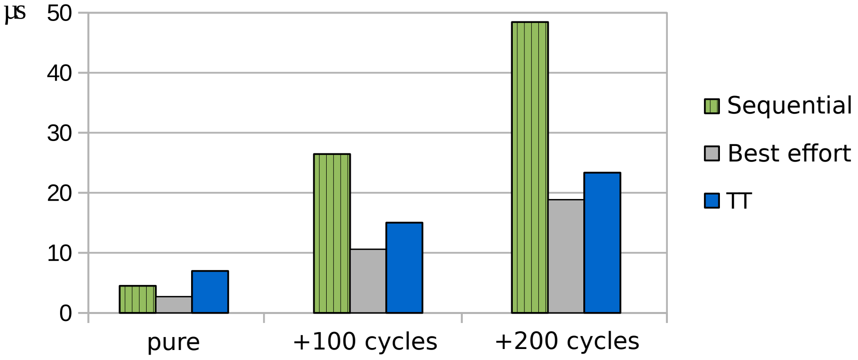
Part II: Real-Time Guarantees

Part III: Evaluation and Conclusion

- Use Cases and Evaluation
- Conclusion

The ROSACE Case Study

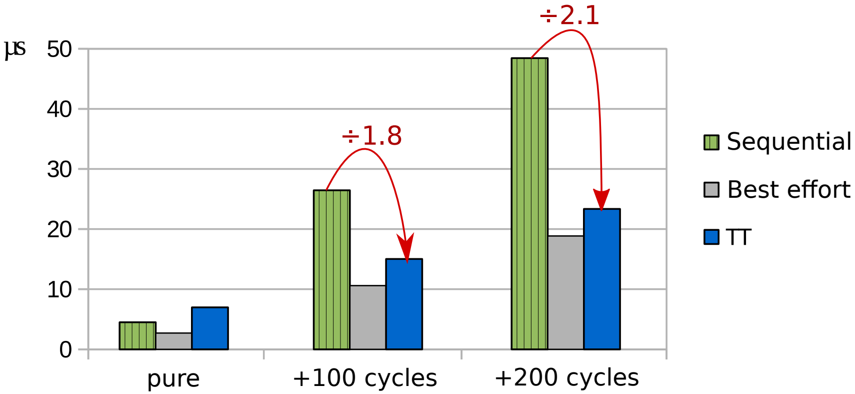
- Altitude-only flight controller
- Open source (Simulink, Lustre, Giotto) [Pagetti, Soussié, RTAS’14]



- +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

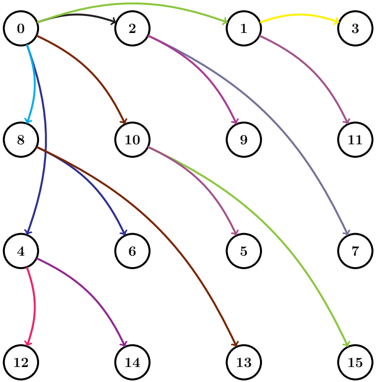
- Altitude-only flight controller
- Open source (Simulink, Lustre, Giotto) [Pagetti, Soussié, RTAS'14]



- +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

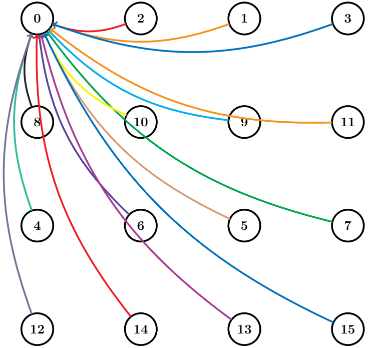
Synthetic Benchmark on 64 cores

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



Dispatch

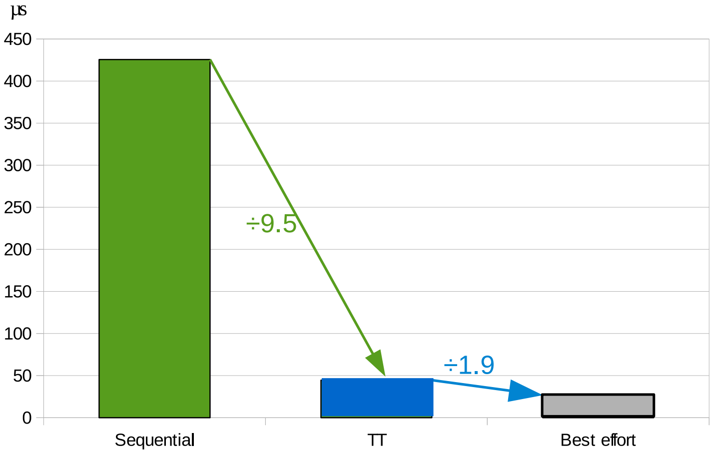
Compute



Gather

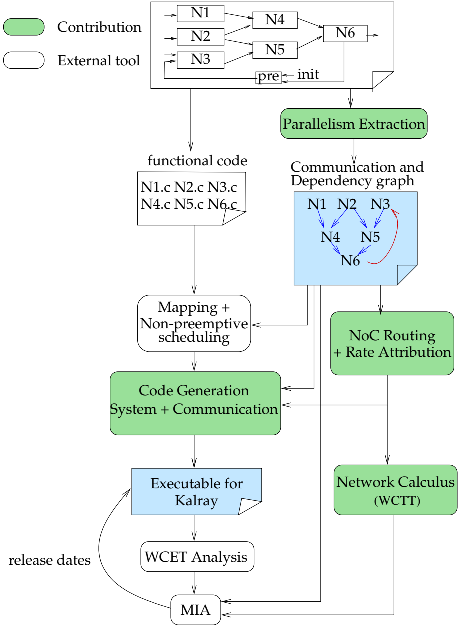
SYNTHETIC BENCHMARK ON 64 CORES

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



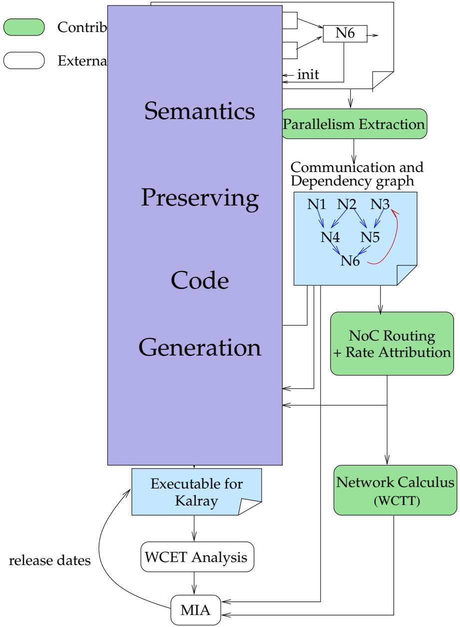
- 54% of WCRT for functional code
- 46% of WCRT for communication and system code

Conclusion

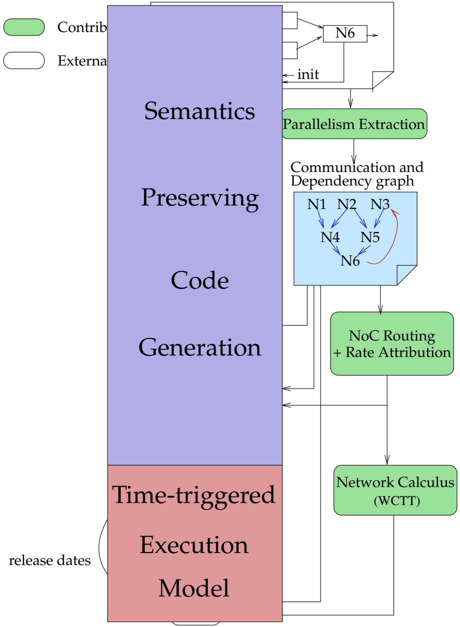


Contribution
External tool
N1
N2
N3
N4
N5
N6
pre
init
Parallelism Extraction
functional code
N1.c N2.c N3.c
N4.c N5.c N6.c
Communication and
Dependency graph
N1 N2 N3
N4 N5
N6
Mapping +
Non-preemptive
scheduling
NoC Routing
+ Rate Attribution
Code Generation
System + Communication
Executable for
Kalray
Network Calculus
(WCTT)
WCET Analysis
release dates
MIA

Conclusion

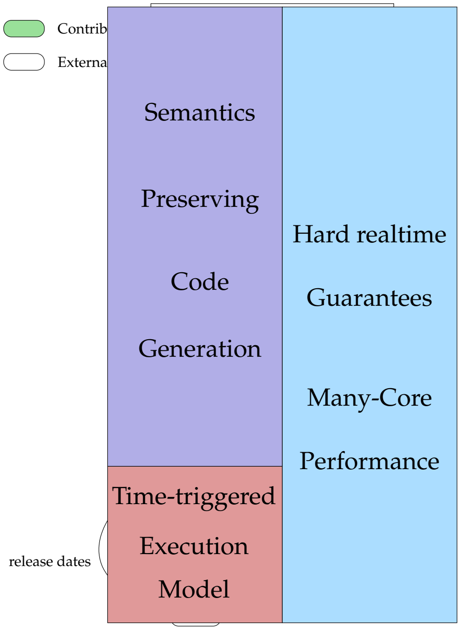

Contrib
Externa
Semantics
Preserving
Code
Generation
N6
init
Parallelism Extraction
Communication and
Dependency graph
N1 N2 N3
N4 N5
N6
NoC Routing
+ Rate Attribution
Network Calculus
(WCTT)
Executable for
Kalray
WCET Analysis
release dates
MIA

Conclusion

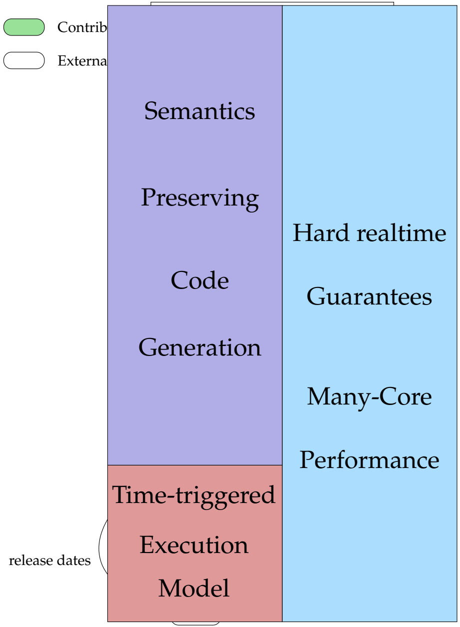


Contrib
Externa
N6
init
Semantics
Preserving
Code
Generation
Parallelism Extraction
Communication and
Dependency graph
N1 N2 N3
N4 N5
N6
NoC Routing
+ Rate Attribution
Network Calculus
(WCTT)
Time-triggered
Execution
Model
release dates

CONCLUSION

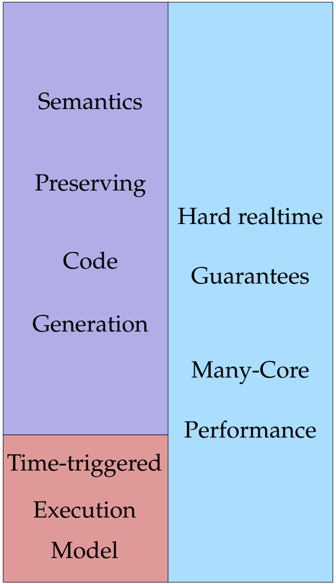

Contrib
Externa
Semantics
Preserving
Code
Generation
Hard realtime
Guarantees
Many-Core
Performance
Time-triggered
Execution
Model
release dates

CONCLUSION

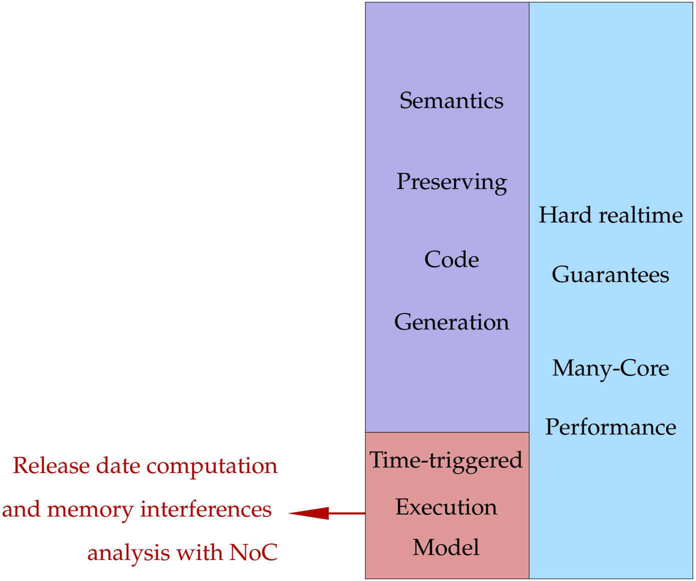


Bridge between academic and industry

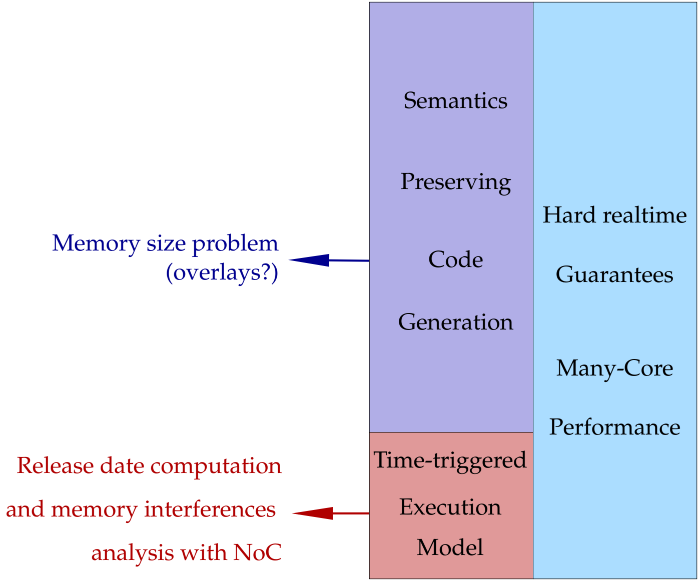
Future Work


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

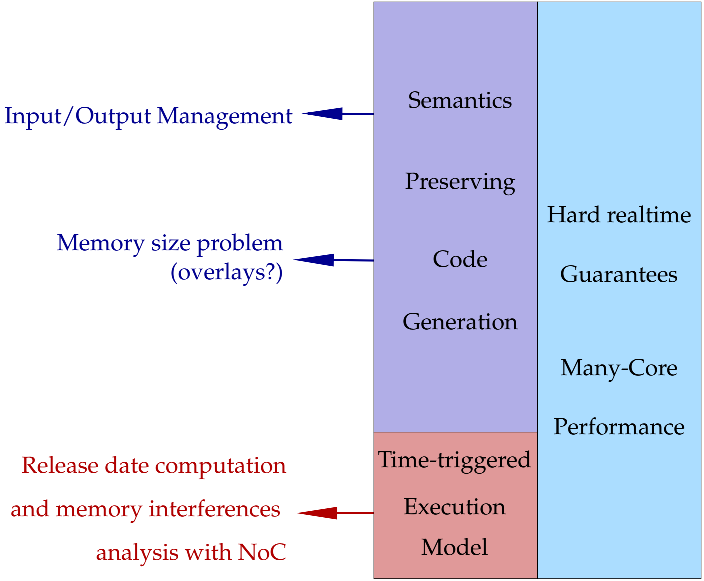
Future Work


Semantics
Preserving
Code
Generation
Hard realtime
Guarantees
Many-Core
Performance
Time-triggered
Execution
Model
Release date computation
and memory interferences
analysis with NoC

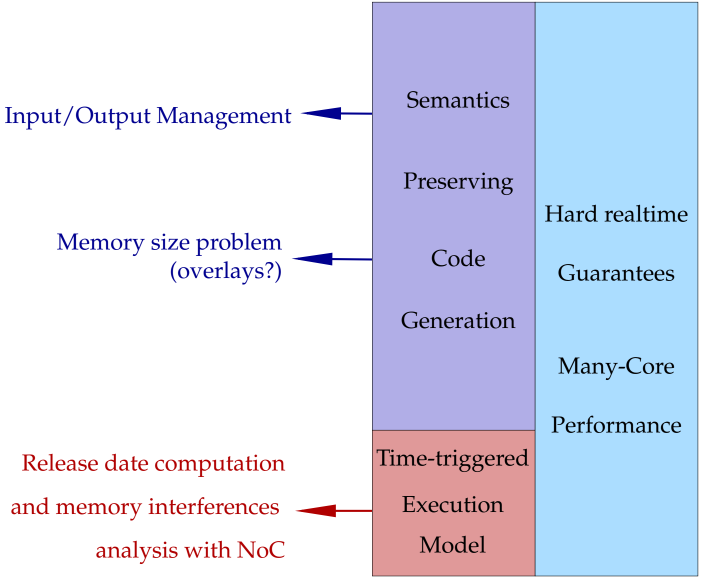
Future Work


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

Future Work


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

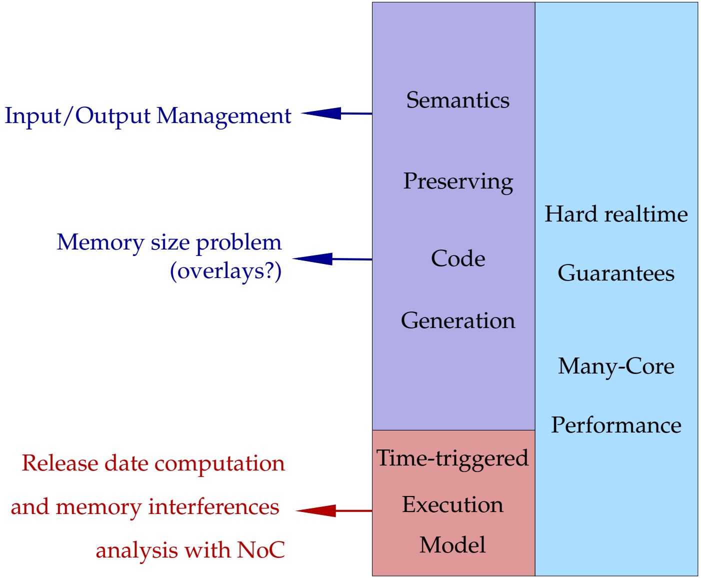
FUTURE WORK



What should be the standard

real-time multi-core processor?

Future Work



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.

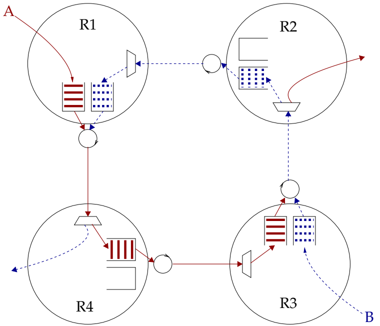
Submitted

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.

References

Backup

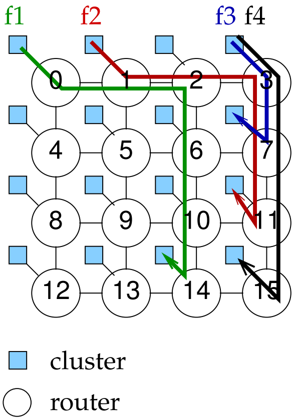
Deadlock in Wormhole Networks



- 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

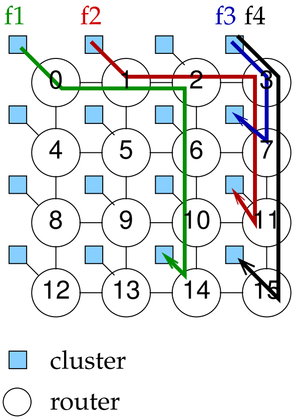
- Deadlock-freeness can be ensured at routing time
- Solutions: XY, Hamiltonian Odd-Even, Turn Prohibition, etc

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

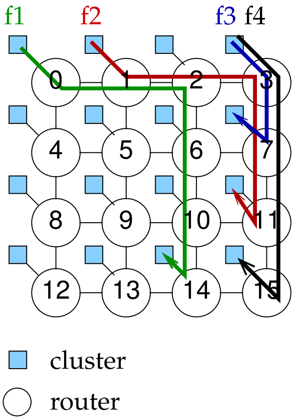
Max Min Fair Rate Attribution



- 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"

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

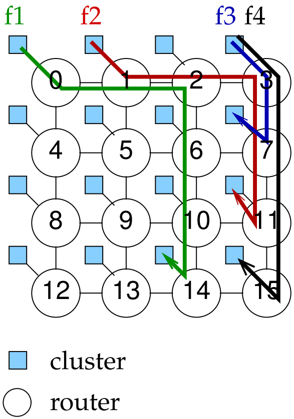
MAX MIN FAIR RATE ATTRIBUTION



- 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"

Example 1:
f1 = $\frac{1}{2}$, f2 = $\frac{1}{2}$, f3 = $\frac{1}{4}$, f4 = $\frac{1}{4}$
$\rightarrow$ Valid **but not** max-min fair (since increasing f3 or f4 can be done by reducing the greater f2)

Max Min Fair Rate Attribution

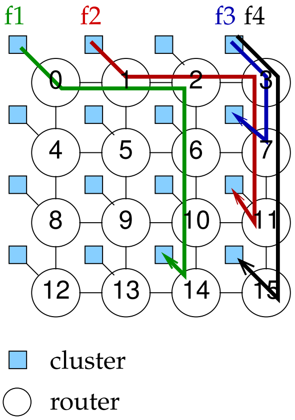


- Rate limiter configuration to avoid buffer overflow
- Rate of a flow f_i noted ρ_i
- Valid: for each link, $\sum_i \rho_i \leq 1$ flit/cycle
- Fair attribution: max-min fairness [**?**]: "cannot increase a rate without decreasing an already smaller (or equal) rate"

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



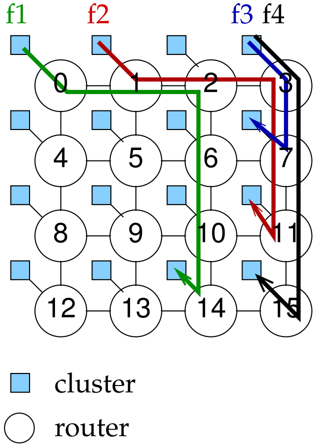
- Rate limiter configuration to avoid buffer overflow
- Rate of a flow f_i noted ρ_i
- Valid: for each link, $\sum_i \rho_i \leq 1$ flit/cycle
- Fair attribution: max-min fairness [**?**]: "cannot increase a rate without decreasing an already smaller (or equal) rate"

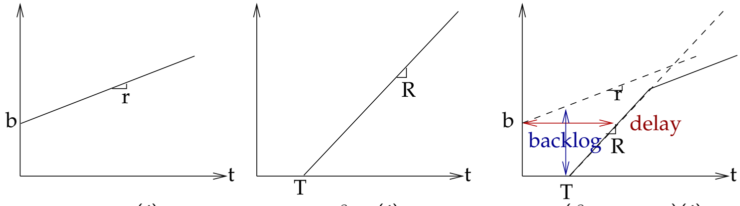
Example 1:
f1 = $\frac{1}{2}$, f2 = $\frac{1}{2}$, f3 = $\frac{1}{4}$, f4 = $\frac{1}{4}$
$\rightarrow$ Valid **but not** max-min fair (since increasing f3 or f4 can be done by reducing the greater f2)

Example 2:
f1 = $\frac{2}{3}$, f2 = $\frac{1}{3}$, f3 = $\frac{1}{3}$, f4 = $\frac{1}{3}$
$\rightarrow$ Valid and max-min fair (increasing f2, f3 or f4 cannot be done without reducing one of them)

Classical solution: the "Water Filling" algorithm [**?**]

Deterministic Network Calculus (DNC) Principle





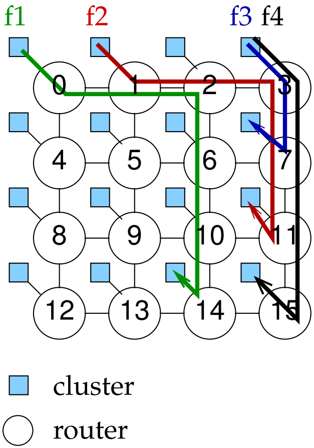
$\gamma_{r,b}(t)$ Arrival Curve

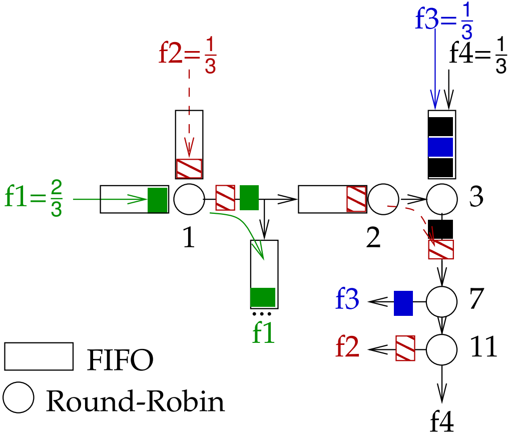
$\beta_{R,T}(t)$ Service Curve

$(\beta_{R,T} \otimes \gamma_{r,b})(t)$ Convolution

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

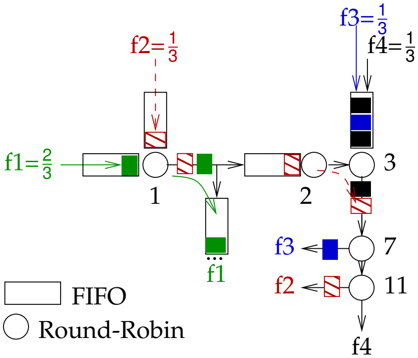
Kalray MPPA2 Network-on-Chip





Kalray MPPA2 Network Elements

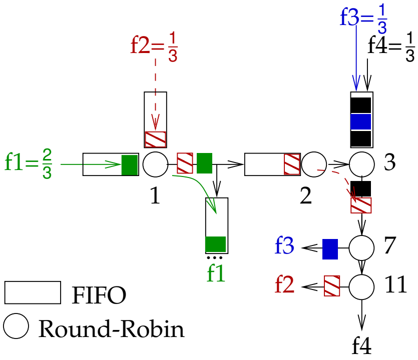
SEPARATED FLOW ANALYSIS (1/2)



Separated Flow Analysis:

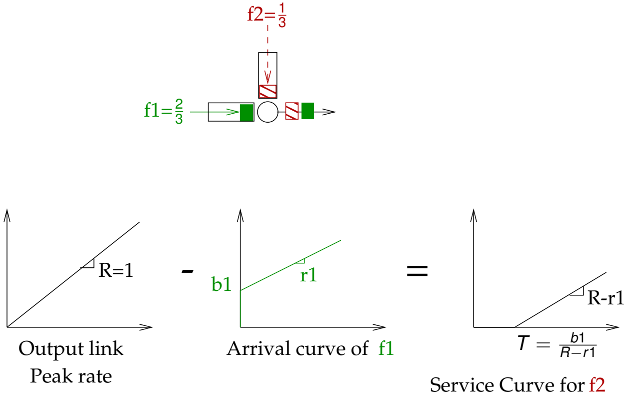
- Compute the service curve of each network element
- Compute the successive arrival curves at each network element
- Convolution of the element service curves $\rightarrow$ network service curve

Separated Flow Analysis (2/2)



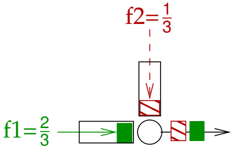
- Service curve offered to f2?
- At routers 1, 2, 3, 7 and 11.

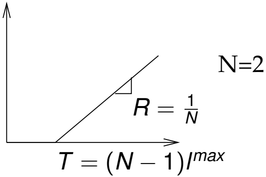
Blind Multiplexing



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

ROUND ROBIN MULTIPLEXING

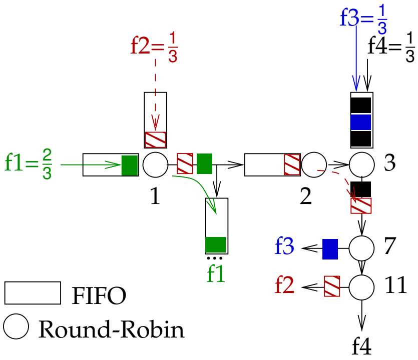




Service Curve for f2

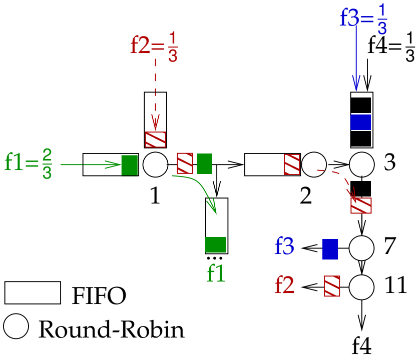
- Packets of size l^{max}
- Restriction: Rate $\leq$ R (not applicable to f1)
- Blind multiplexing is the conservative solution.

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

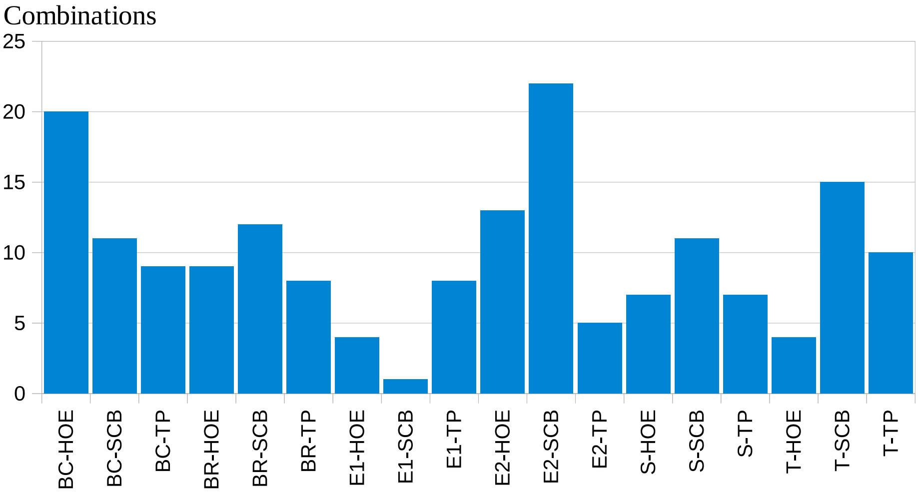
Worst-Case Traversal Time (WCTT): application (2/2)



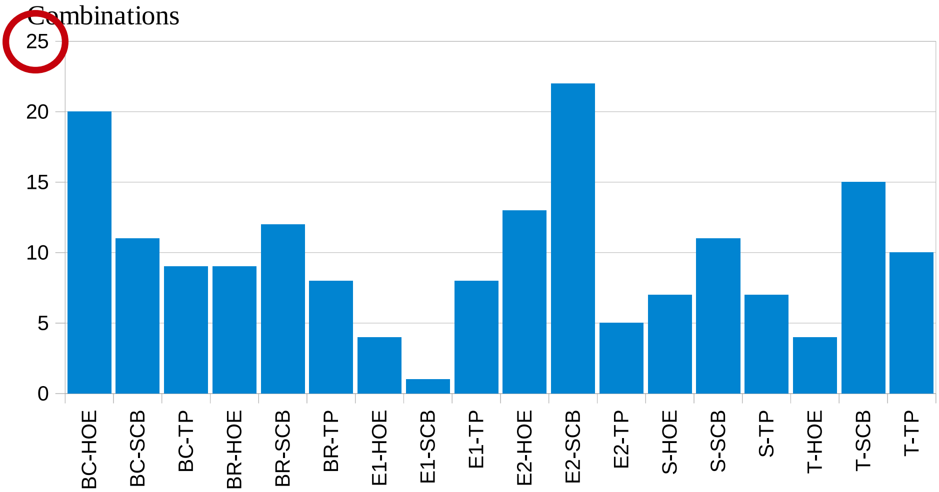
	f1	f2	f3	f4
WCTT (cycles)	25	68	76	76

With packets of l^{max}=17 flits.

Evaluation of Our LP-based Heuristic



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