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

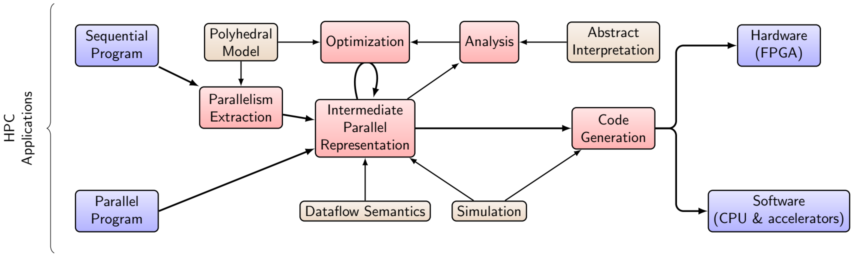
Matthieu Moy

LIP (Univ. Lyon 1)

November 2018

CASH: Topics - People

Optimized (software/hardware) compilation for HPC software with data-intensive computations.
$\rightsquigarrow$ Means: dataflow IR, static analyses, optimisations, simulation.



Christophe Alias, Laure Gonnord, Ludovic Henrio, Matthieu Moy `http://www.ens-lyon.fr/LIP/CASH/`

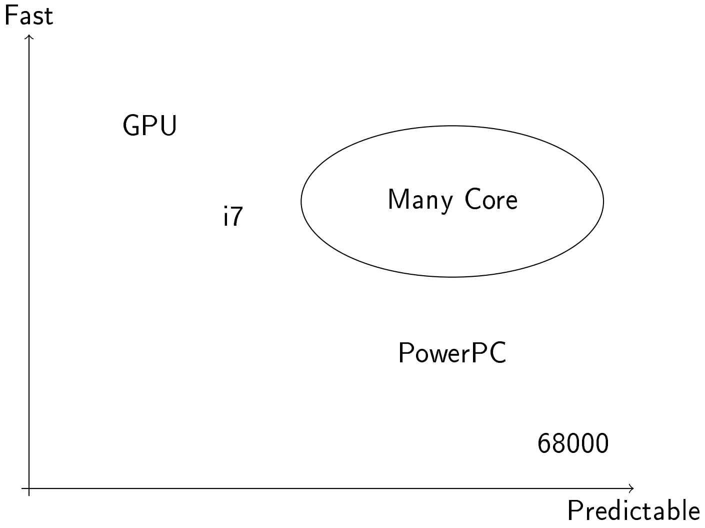
Outline

Outline

Time-critical, compute intensive applications

- Hard Real-Time: we must guarantee that task execution completes before deadline
- Compute-intensive
- Space/power bounded

Performance Vs Predictability



Many-core

=

Lots of simple cores

Many-core

=

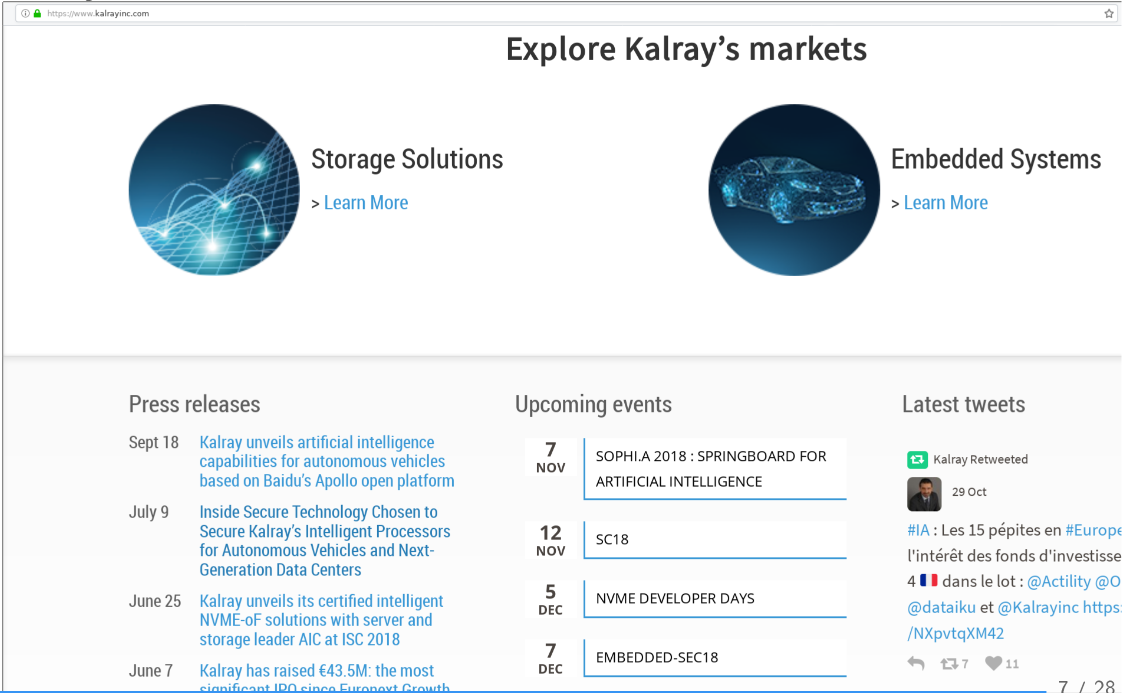
Lots of simple cores

Kalray MPPA (Massively Parallel Processor Array):

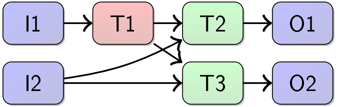
- 256 cores
- No cache consistency
- No out-of-order execution
- No branch prediction
- No timing anomaly
- Predictable NoC

$\Rightarrow$ good fit for real-time?

Kalray's business model



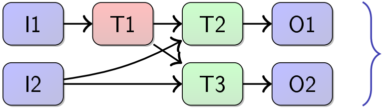
Hard Real-Time on Many-Core



High-level Data-Flow Application Model

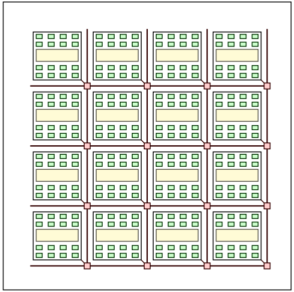
Synchronous hypothesis:
computation/communication in 0-time

Hard Real-Time on Many-Core

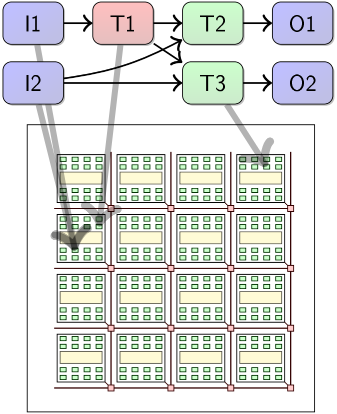


High-level Data-Flow Application Model

Synchronous hypothesis:
computation/communication in 0-time



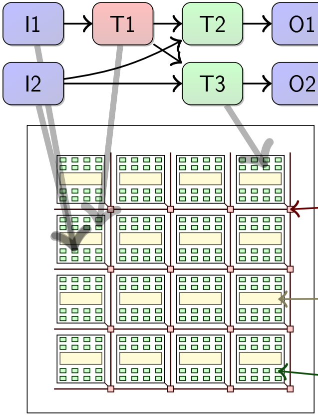
Hard Real-Time on Many-Core



High-level Data-Flow Application Model

Synchronous hypothesis:
computation/communication in 0-time

Hard Real-Time on Many-Core



High-level Data-Flow Application Model

Synchronous hypothesis: computation/communication in 0-time

Network On Chip

Communication takes time

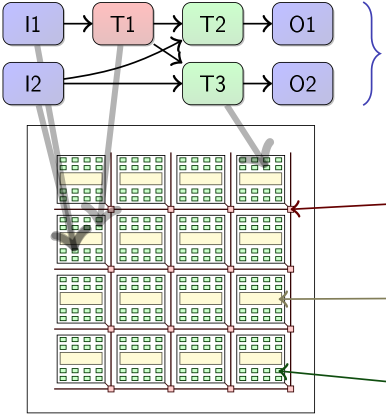
Shared Memory within Cluster

Interferences between tasks

Individual Cores

Cache, Pipeline, . . .

Hard Real-Time on Many-Core



High-level Data-Flow Application Model

Synchronous hypothesis: computation/communication in 0-time

Network On Chip

Communication takes time

Shared Memory within Cluster

Interferences between tasks

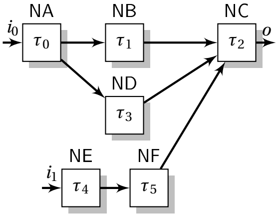
Individual Cores

Cache, Pipeline, . . .

$\rightsquigarrow$ Take into account all levels
in Worst-Case Execution Time (WCET) analysis
and programming model

Outline

Execution of Synchronous Data Flow Programs



High level representation

Single-core

code generation

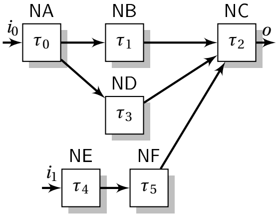
static non-preemptive scheduling

```
int main_app(i1, i2)
{
    na = NA(i1);
    ne = NE(i2);
    nb = NB(na);
    nd = ND(na);
    nf = NF(ne);
    o = NC(nb,nd,nf);
    return o;
}
```

Industrialized as SCADE (1993)

heavily used in avionics and nuclear

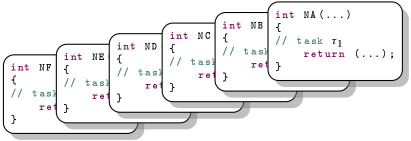
Execution of Synchronous Data Flow Programs



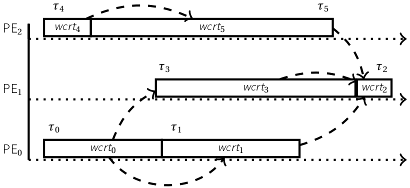
High level representation

Multi/Many-core

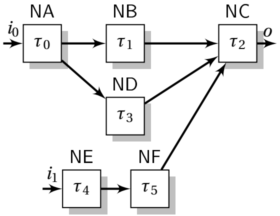
code generation

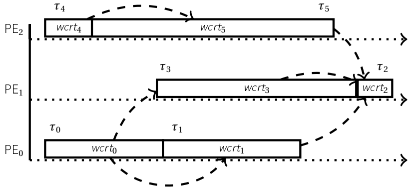


static non-preemptive scheduling



Parallel code generation from Lustre/SCADE (pseudo-code)






```
// Generated by SCADE KCG
void NA(ctx_a *ctx) {
    // ... computation ...
}

void NA_wrapper(ctx_a *ctx) {
    RECV_NA(i0);
    NA(ctx);
    SEND_NA_NB(...);
}
```

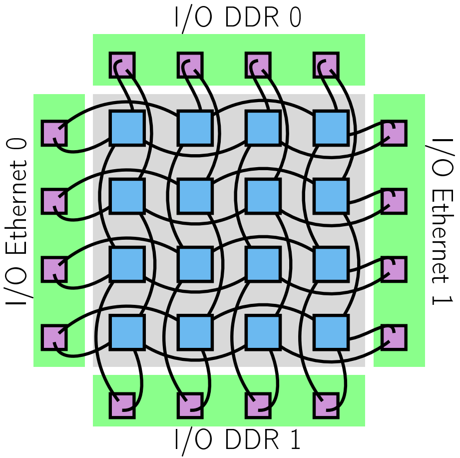
```
// Generated by us
void worker_PE0(void) {
    ctx_a ctxa; ctx_b ctxb;
    while (1) {
        NA_wrapper(&ctxa);
        wait(release_t2);
        NB_wrapper(&ctxb);
        wait(end_of_period);
    }
}

#define RECV_NA(data) ...
```

Contribution

- Previous work:
  - Predictable execution model within each cluster
  - Mathematical model of arbitration for memory accesses
  - Algorithm to compute a time-triggered schedule (fix-point resolution)
- This talk:
  - Multi-cluster application
  - Time-triggered schedule taking Network on Chip (NoC) accesses into account

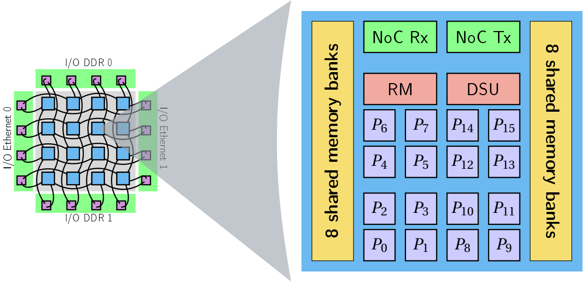
Outline

Architecture Model



- Kalray MPPA 256 Bostan
- 16 compute clusters + 4 I/O clusters
- Dual NoC (Network on Chip)

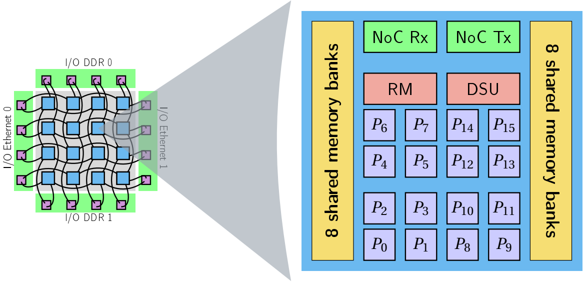
Architecture Model



Per cluster:

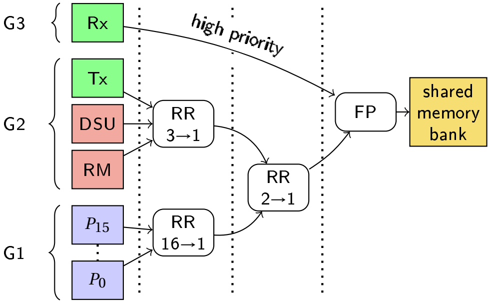
- 16 cores + 1 Resource Manager
- NoC Tx, NoC Rx, Debug Unit
- 16 shared memory banks (total size: 2 MB)

Architecture Model

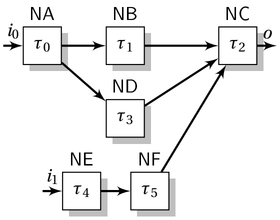


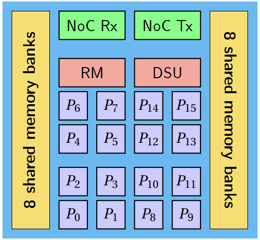
Per cluster:

- 16 cores + 1 Resource Manager
- NoC Tx, NoC Rx, Debug Unit
- 16 shared memory banks (total size: 2 MB)
- Multi-level bus arbiter per memory bank



Execution Model: Within a Cluster



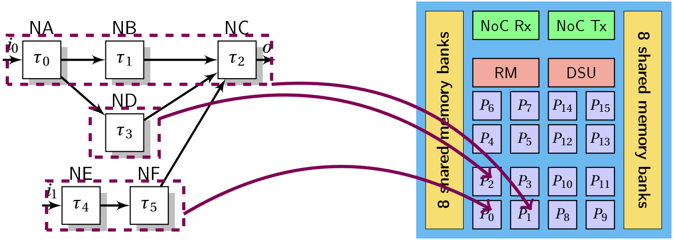


- Tasks mapping on cores
- Static non-preemptive scheduling
- Spatial Isolation

  different tasks go to different memory banks
- Interference from communications

- Execution model:
  - execute in a "local" bank
  - write to a "remote" bank

Execution Model: Within a Cluster

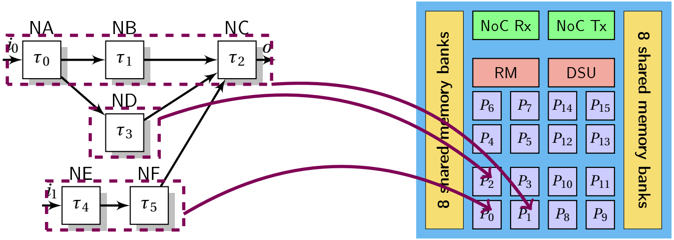


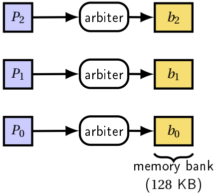
- Tasks mapping on cores
- Static non-preemptive scheduling
- Spatial Isolation

  different tasks go to different memory banks
- Interference from communications

- Execution model:
  - execute in a "local" bank
  - write to a "remote" bank

Execution Model: Within a Cluster



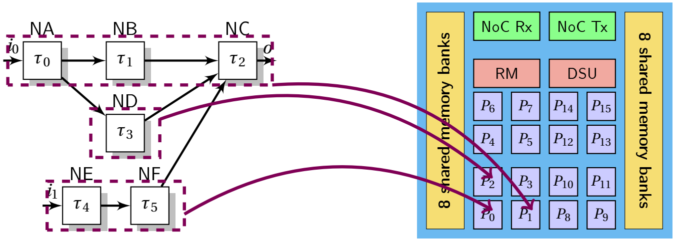


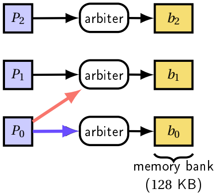
- Tasks mapping on cores
- Static non-preemptive scheduling
- Spatial Isolation

  different tasks go to different memory banks
- Interference from communications

- Execution model:
  - execute in a “local” bank
  - write to a “remote” bank

Execution Model: Within a Cluster



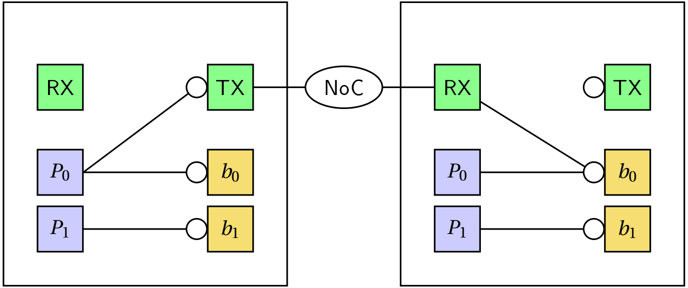


- Tasks mapping on cores
- Static non-preemptive scheduling
- Spatial Isolation

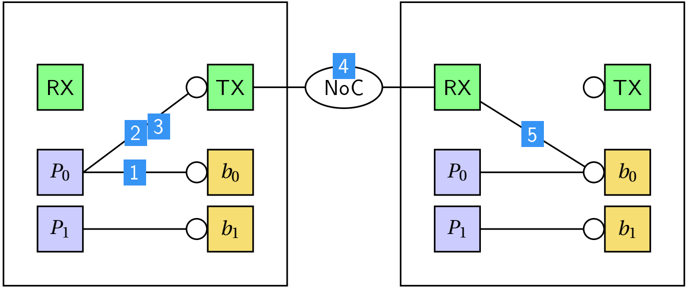
  different tasks go to different memory banks
- Interference from communications

- Execution model:
  - execute in a “local” bank
  - write to a “remote” bank

NoC Communications



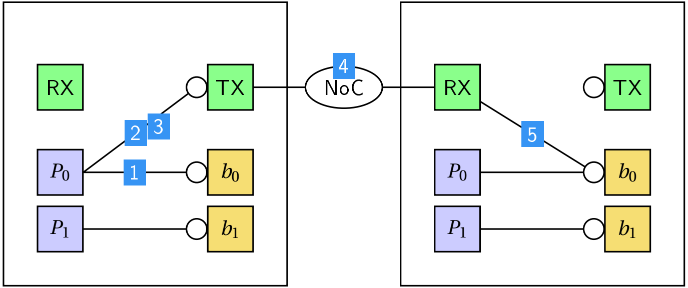
NoC Communications



Steps:

1. Read from memory
2. Write to TX's buffer
3. Start NoC transfer
4. Data transmission through the NoC
5. Write to memory

NoC Communications



Steps:

1 Read from memory

2 Write to TX's buffer

3 Start NoC transfer

4 Data transmission through the NoC

5 Write to memory

Interference:

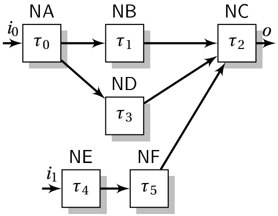
1 Same as other reads

2, 3 One TX channel per sender
$\Rightarrow$ independent accesses.

4 Interferences in each router
$\rightarrow$ network calculus

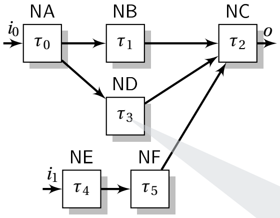
5 High-priority interference $\Rightarrow$ ⚠

Application Model and Interferences

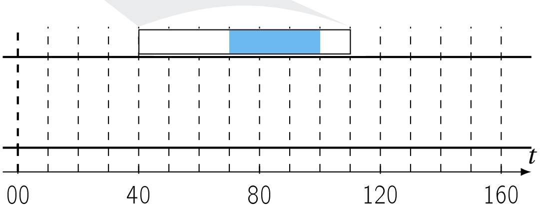


- Directed Acyclic Task Graph
- Mono-rate
- Fixed mapping and execution order
- **For each task τ_i:**
  $\mathrm{WCRT}_i = \mathrm{WCET}_i + \sum_{j \neq i} \mathrm{interference}_{i,j}$

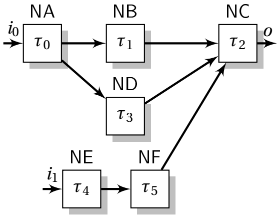
Application Model and Interferences



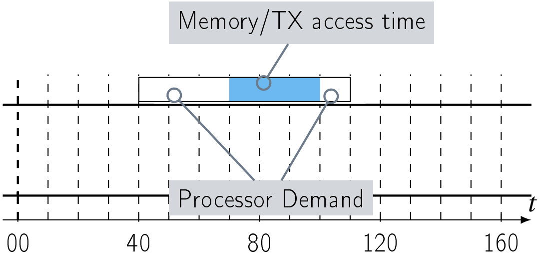
- Directed Acyclic Task Graph
- Mono-rate
- Fixed mapping and execution order
- **For each task τ_i:**
  $\text{WCRT}_i = \text{WCET}_i + \sum_{j \neq i} \text{interference}_{i,j}$



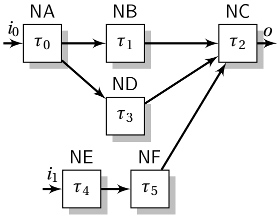
Application Model and Interferences



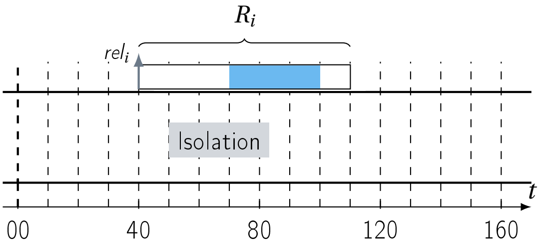
- Directed Acyclic Task Graph
- Mono-rate
- Fixed mapping and execution order
- **For each task τ_i:**
  $\mathrm{WCRT}_i = \mathrm{WCET}_i + \sum_{j \neq i} \mathrm{interference}_{i,j}$



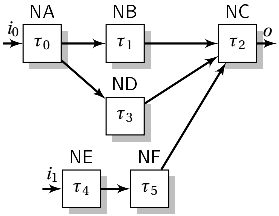
Application Model and Interferences



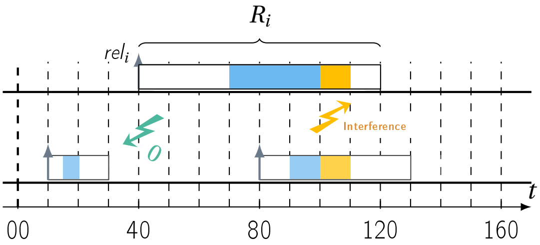
- Directed Acyclic Task Graph
- Mono-rate
- Fixed mapping and execution order
- **For each task τ_i:**
  $\text{WCRT}_i = \text{WCET}_i + \sum_{j \neq i} \text{interference}_{i,j}$



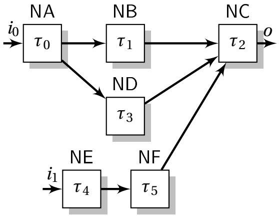
Application Model and Interferences



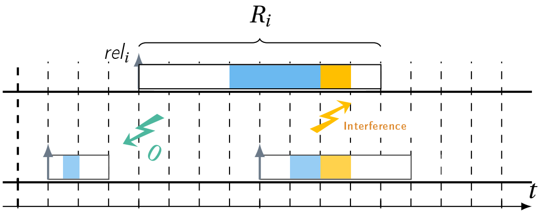
- Directed Acyclic Task Graph
- Mono-rate
- Fixed mapping and execution order
- **For each task τ_i:**
  $\mathrm{WCRT}_i = \mathrm{WCET}_i + \sum_{j \neq i} \mathrm{interference}_{i,j}$



Application Model and Interferences



- Directed Acyclic Task Graph
- Mono-rate
- Fixed mapping and execution order
- **For each task τ_i:**
  $\text{WCRT}_i = \text{WCET}_i + \sum_{j \neq i} \text{interference}_{i,j}$

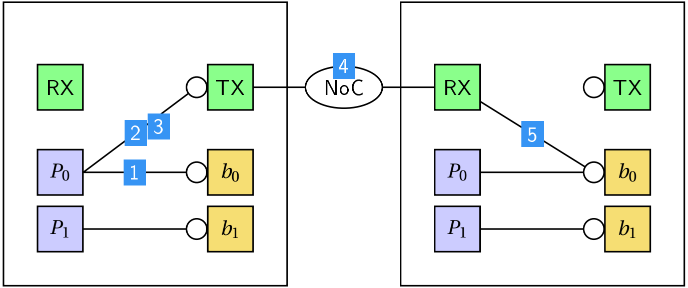


Previous work:

- Find R_i (including the interference)
- Find rel_i respecting precedence constraints

Outline

Reminder: NoC Communications



Interference:

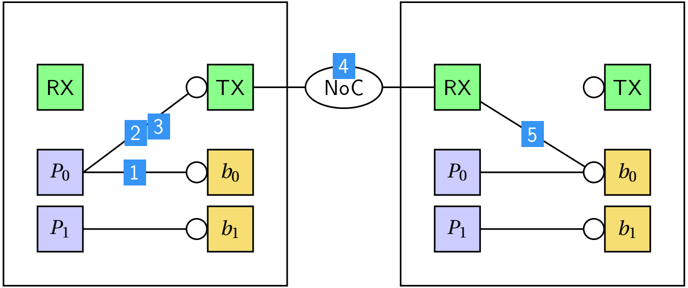
1 Same as other reads

2, 3 One TX channel per sender
$\Rightarrow$ independent accesses.

4 Interferences in each router
$\rightarrow$ network calculus

5 High-priority interference $\Rightarrow$ ⚠

Reminder: NoC Communications



Interference:

1 Same as other reads

2, 3 One TX channel per sender
⇒ independent accesses.

4 Interferences in each router
→ network calculus

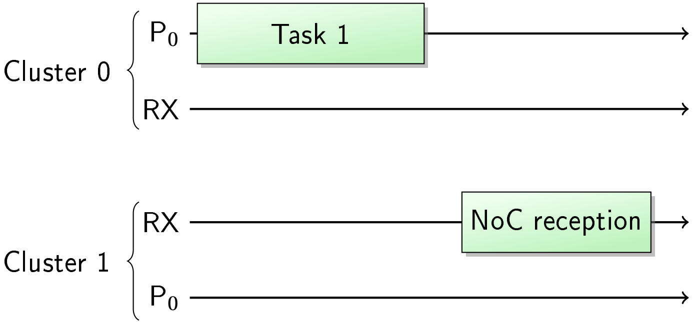
5 High-priority interference ⇒ ⚠

Issue:

Predict the possible execution time of 5 as precisely as possible.

Example Tasks with NoC Transmission

Issue 1: overapproximation of RX execution interval

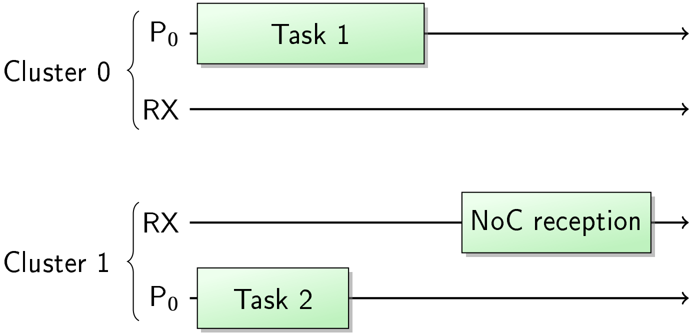


- **Issue:**
  - NoC reception starts after BCET[a](computation before sending) + BCET(transmission of first flit)
  - We don't have the BCET
    $\Rightarrow$ large overapproximation for RX tasks

[a] Best Case Execution Time

Example Tasks with NoC Transmission

Issue 1: overapproximation of RX execution interval

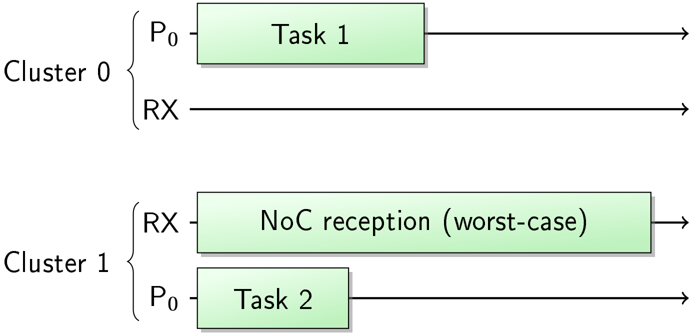


- **Issue:**
  - NoC reception starts after BCET[a](computation before sending) + BCET(transmission of first flit)
  - We don't have the BCET
    ⇒ large overapproximation for RX tasks

[a]Best Case Execution Time

Example Tasks with NoC Transmission

Issue 1: overapproximation of RX execution interval

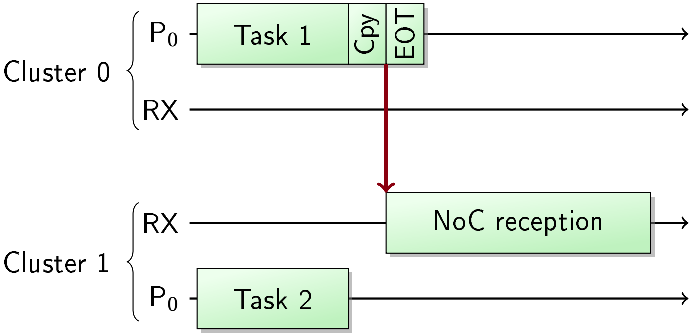


- **Issue:**
  - NoC reception starts after BCET[a](computation before sending) + BCET(transmission of first flit)
  - We don't have the BCET $\Rightarrow$ large overapproximation for RX tasks
- **Solution: Split sending task**
  - Compute: no NoC access
  - Copy to TX (Cpy): write to the TX's buffer
  - Start NoC transfer (EOT): write to TX's control register

[a] Best Case Execution Time

Example Tasks with NoC Transmission

Issue 1: overapproximation of RX execution interval

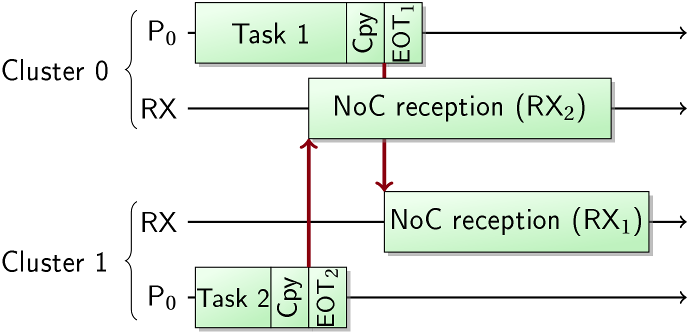


- **Issue:**
  - NoC reception starts after BCET[a](computation before sending) + BCET(transmission of first flit)
  - We don't have the BCET
    ⇒ large overapproximation for RX tasks
- **Solution: Split sending task**
  - Compute: no NoC access
  - Copy to TX (Cpy): write to the TX's buffer
  - Start NoC transfer (EOT): write to TX's control register

[a] Best Case Execution Time

Example Tasks with NoC Transmission

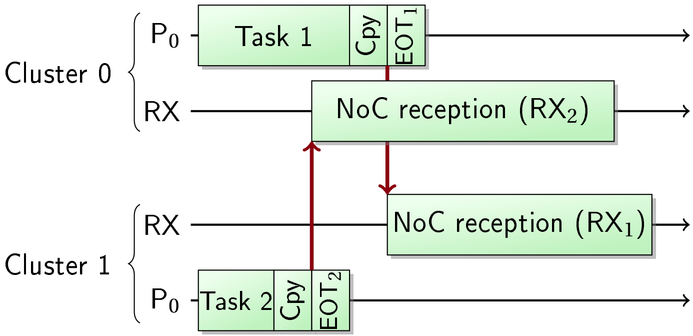
Issue 2: circular dependency



- **Issue:**
  - WCRT(RX_1) = WCRT(EOT_1) + WCTT($1 \rightarrow 0$)
  - WCRT(EOT_1) depends on WCRT(RX_2), which depends on WCRT(EOT_2) which depends on WCRT(RX_1)
  - $\Rightarrow$ fix-point

Example Tasks with NoC Transmission

Issue 2: circular dependency



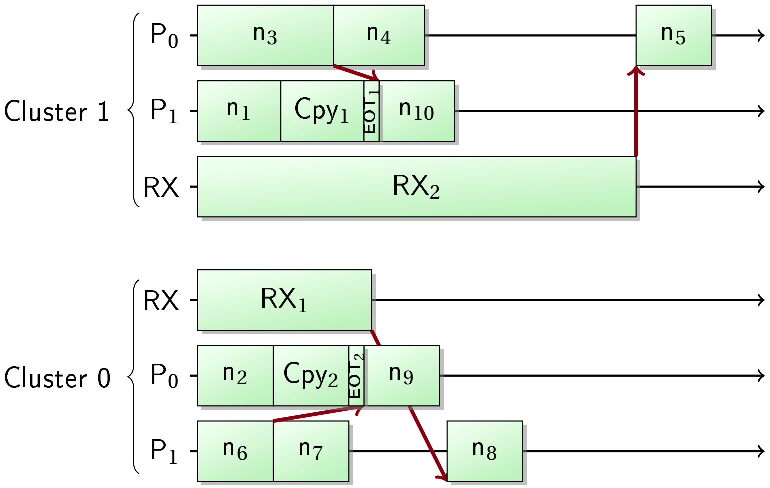
- **Issue:**
  - $WCRT(RX_1) = WCRT(EOT_1) + WCTT(1 \rightarrow 0)$
  - $WCRT(EOT_1)$ depends on $WCRT(RX_2)$, which depends on $WCRT(EOT_2)$ which depends on $WCRT(RX_1)$
  - $\Rightarrow$ fix-point
- **Solution: get rid of interference on EOT**
  - EOT = only one control register access
  - Preload code to avoid instruction cache miss

3-phase Tasks Analysis

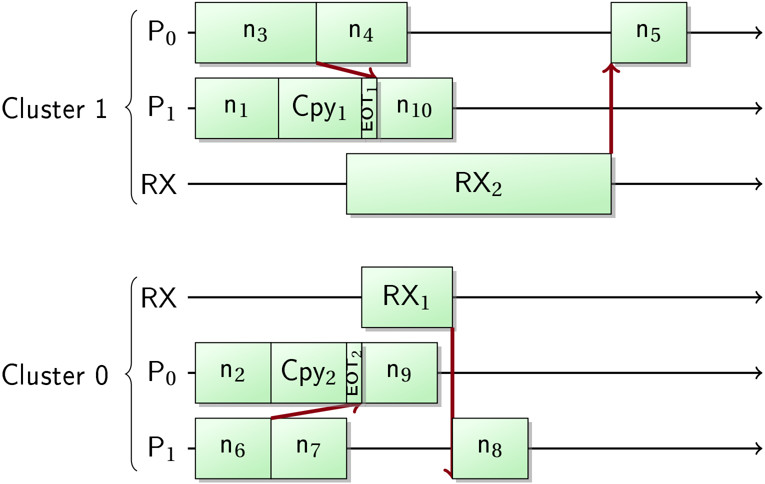
- Compute:
  - Fits in previous work model
- Copy to TX:
  - Force non-interfering schedule (add artificial dependencies if needed)
- Start NoC transfer (EOT):
  - No interference
- On the RX side:
  - RX can only start after "Start NoC transfer" has started
    $\Rightarrow$ edge from "Copy to TX" to "RX" in the task dependency graph.

Outline

Example Application: Naive Schedule



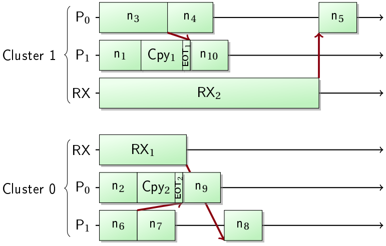
Example Application: Improved Schedule



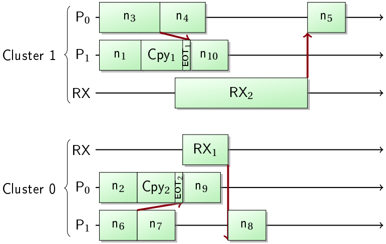
Recap

Task	WCRT		Release date	
	Naive	Improved	Naive	Improved
Application	**680**	**650**		
RX_1	**230**	**120**	**0**	**220**
RX_2	**580**	**350**	**0**	**200**
n_3	**180**	**160**	0	0
n_4	120	120	**180**	**160**
n_2	100	100	**10**	**0**
n_5	100	100	**580**	**550**
n_8	100	100	**330**	**340**
n_1	110	110	0	0
n_6	100	100	0	0
n_7	100	100	100	100
n_9	100	100	220	220
n_{10}	100	100	240	240
Cpy_1	110	110	110	110
Cpy_2	100	100	100	100
EOT_1	20	20	220	220
EOT_2	20	20	200	200

Naive:



Improved:



Outline

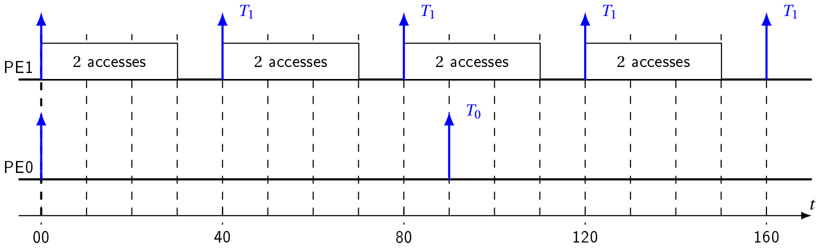
Conclusion and Future Work

- Code generation and real-time analysis for many-core (Kalray MPPA 256) = major challenge for industry and research
- Hard Real-Time ⇒ simplicity, predictability ⇒ static, time-driven schedule
- Critical ⇒ traceability ⇒ no aggressive optimization
- Our work:
  - Understand and model the precise architecture of MPPA
  - Extension of Multi-Core Response Time Analysis framework
  - Integration analysis ↔ code generation
- Future Work:
  - Model Kalray MPPA3 chip (new NoC, new arbiters)
  - Improve the static scheduling algorithm: $O(n^4)$ currently, we can do better.
  - Integration with RTOS?

BACKUP

Multicore Response Time Analysis

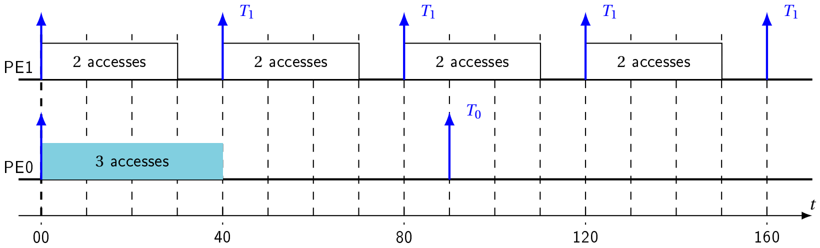
Example: Fixed Priority bus arbiter, PE1 > PE0
Bus access delay = 10



[1] Altmeyer et al., RTNS 2015

Multicore Response Time Analysis

Example: Fixed Priority bus arbiter, PE1 > PE0
Bus access delay = 10



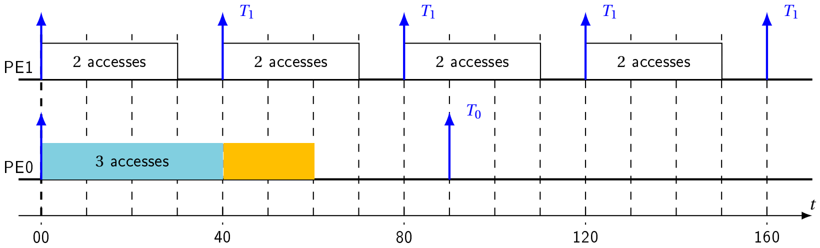
- Task of interest running on $PE0$:

  $R_0 = 10 + 3 \times 10$ (response time in isolation)

[1] Altmeyer et al., RTNS 2015

Multicore Response Time Analysis

Example: Fixed Priority bus arbiter, PE1 > PE0
Bus access delay = 10



- Task of interest running on $PE0$:

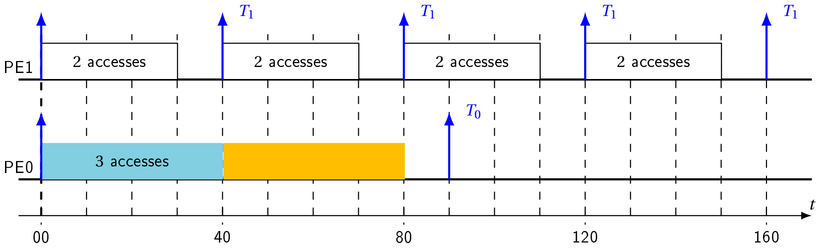
  $R_0 = 10 + 3 \times 10$ (response time in isolation)

  $R_1 = 10 + 3 \times 10 + 2 \times 10 = 60$

[1] Altmeyer et al., RTNS 2015

Multicore Response Time Analysis

Example: Fixed Priority bus arbiter, PE1 > PE0
Bus access delay = 10



- Task of interest running on $PE0$:

  $R_0 = 10 + 3 \times 10$ (response time in isolation)

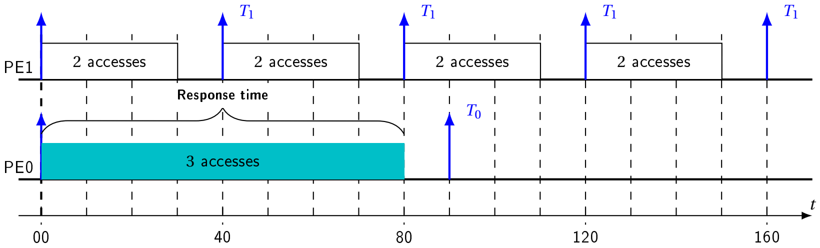
  $R_1 = 10 + 3 \times 10 + 2 \times 10 = 60$

  $R_2 = 10 + 3 \times 10 + 2 \times 10 + 2 \times 10 = 80$

[1] Altmeyer et al., RTNS 2015

Multicore Response Time Analysis

Example: Fixed Priority bus arbiter, PE1 > PE0
Bus access delay = 10



- Task of interest running on $PE0$:

  $R_0 = 10 + 3 \times 10$ (response time in isolation)

  $R_1 = 10 + 3 \times 10 + 2 \times 10 = 60$

  $R_2 = 10 + 3 \times 10 + 2 \times 10 + 2 \times 10 = 80$

  $R_3 = 10 + 3 \times 10 + 2 \times 10 + 2 \times 10 + 0 = 80$ (fixed-point)

[1] Altmeyer et al., RTNS 2015

The Global Picture