Compilation and Real-Time Analysis of a Synchronous Data-Flow Application on the Kalray MPPA Many-Core Processor

Matthieu Moy (from a presentation by Hamza Rihani)

Verimag (Grenoble INP)
Grenoble, France

June 2017

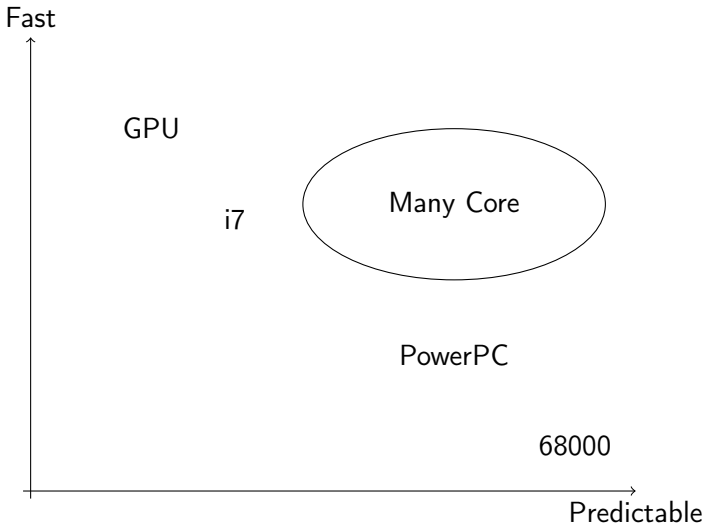
Outline

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

=

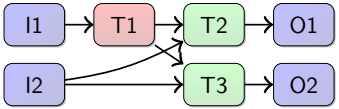
Lots of simple cores

Kalray MPPA (Massively Parallel Processor Array):

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

$\Rightarrow$ good fit for real-time?

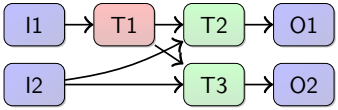
Hard Real-Time on Many-Core



High-level Data-Flow Application Model

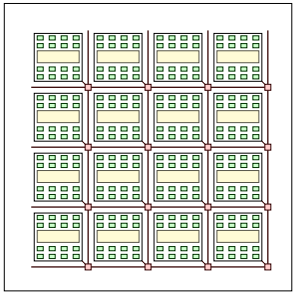
Synchronous hypothesis:
computation/communication in 0-time

Hard Real-Time on Many-Core

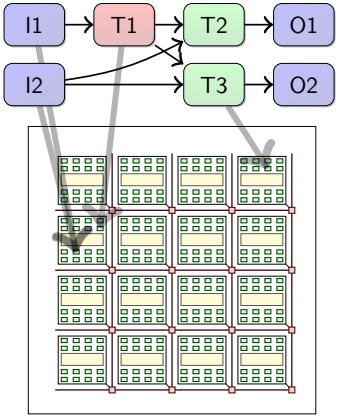


High-level Data-Flow Application Model

Synchronous hypothesis:
computation/communication in 0-time



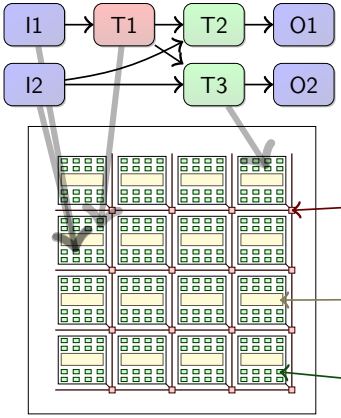
Hard Real-Time on Many-Core



High-level Data-Flow Application Model

Synchronous hypothesis:
computation/communication in 0-time

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

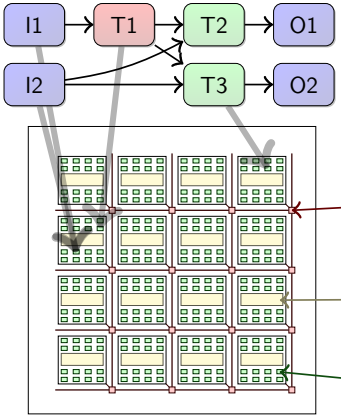
Shared Memory within Cluster

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

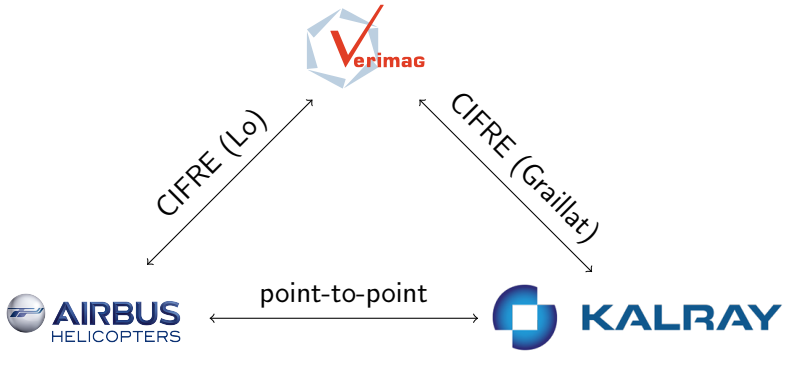
Interferences between tasks

Individual Cores

Cache, Pipeline, . . .

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

Context and Partners

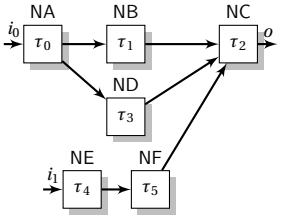


IRIT (WCET) IRISA (Scheduling) ONERA (NoC)

Projet CAPACITES (Ph.D Rihani)

Outline

Execution of Synchronous Data Flow Programs



High level representation

Single-core

code generation

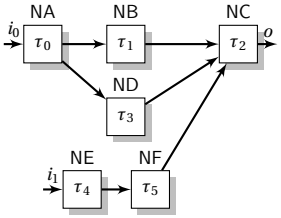
static non-preemptive scheduling

```
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

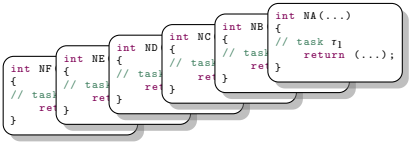
Execution of Synchronous Data Flow Programs



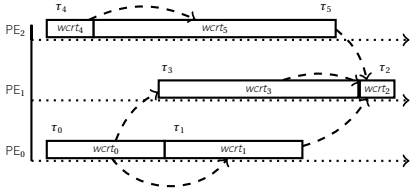
High level representation

Multi/Many-core

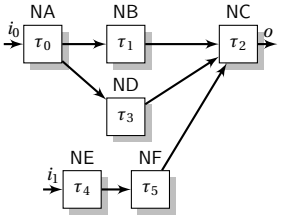
code generation



static non-preemptive scheduling



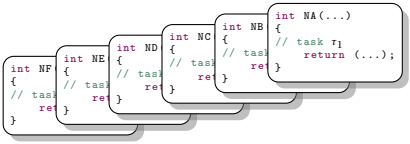
Execution of Synchronous Data Flow Programs



High level representation

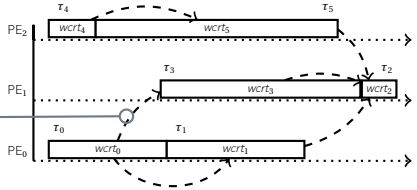
Multi/Many-core

code generation

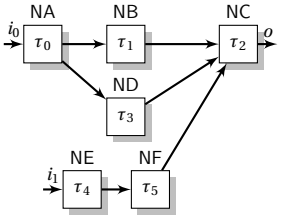


static non-preemptive scheduling

✓ Respect the dependency constraints



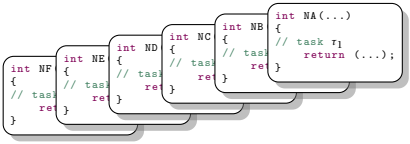
Execution of Synchronous Data Flow Programs



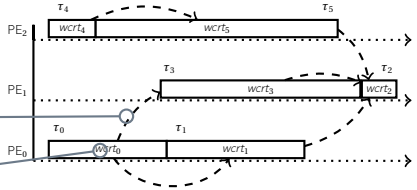
High level representation

Multi/Many-core

code generation



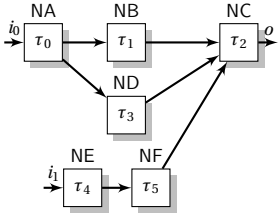
static non-preemptive scheduling

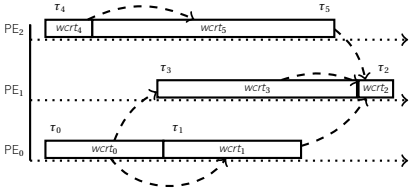


✓ Respect the dependency constraints

✓ Set the release dates to get precise upper bounds on the interference

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



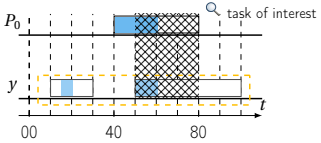



```
// Generated by SCADE KCG
void NA(ctx_a *ctx) {
    // ... computation ...
}

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) ...
```

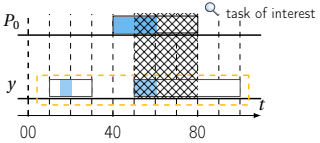
Contributions (part of Ph.D Hamza Rihani, with Claire Maiza)

1. Precise accounting for interference on shared resources in a many-core processor

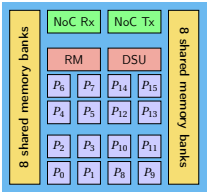


Contributions (part of Ph.D Hamza Rihani, with Claire Maiza)

1. Precise accounting for interference on shared resources in a many-core processor

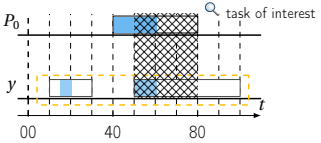


2. Model of a multi-level arbiter to the shared memory

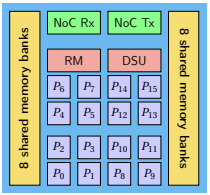


Contributions (part of Ph.D Hamza Rihani, with Claire Maiza)

1. Precise accounting for interference on shared resources in a many-core processor



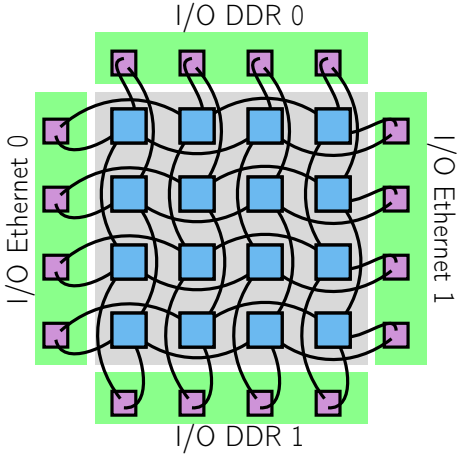
2. Model of a multi-level arbiter to the shared memory



3. Response time and release dates analysis respecting dependencies.

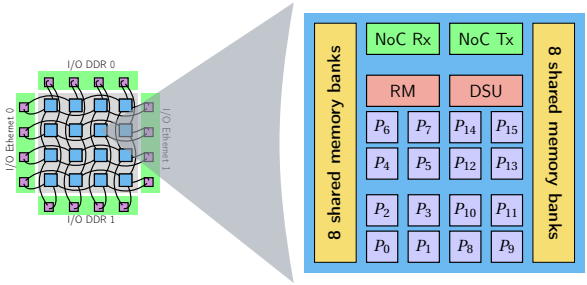
Outline

Architecture Model



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

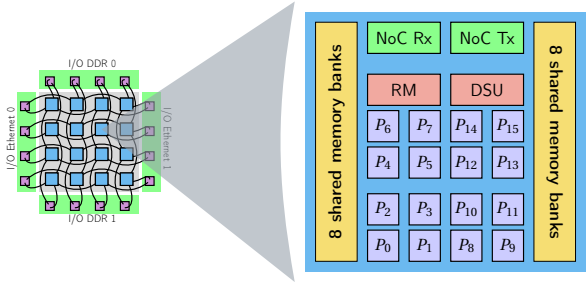
Architecture Model



Per cluster:

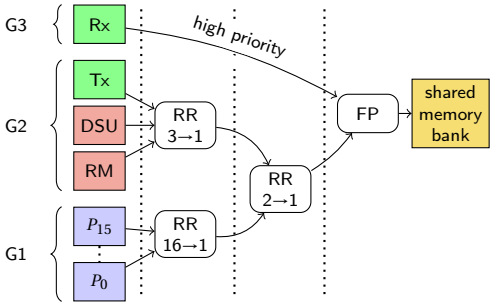
- 16 cores + 1 Resource Manager
- NoC Tx, NoC Rx, Debug Unit
- 16 shared memory banks (total size: 2 MB)

Architecture Model

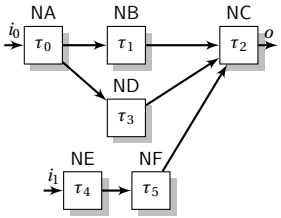


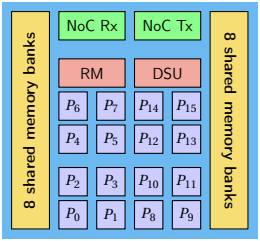
Per cluster:

- 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



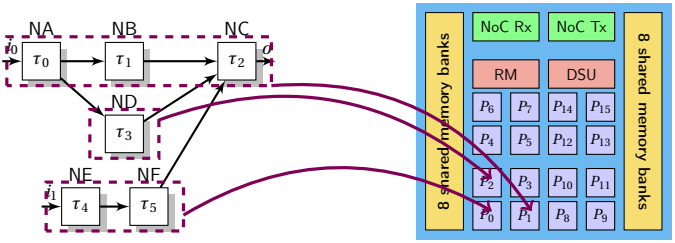


- Tasks mapping on cores
- Static non-preemptive scheduling
- Spatial Isolation

  different tasks go to different memory banks
- Interference from communications

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

Execution Model

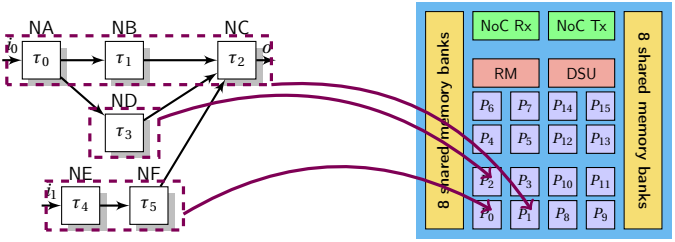


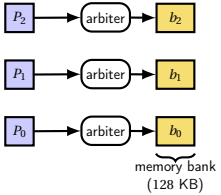
- Tasks mapping on cores
- Static non-preemptive scheduling
- Spatial Isolation

  different tasks go to different memory banks
- Interference from communications

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

Execution Model



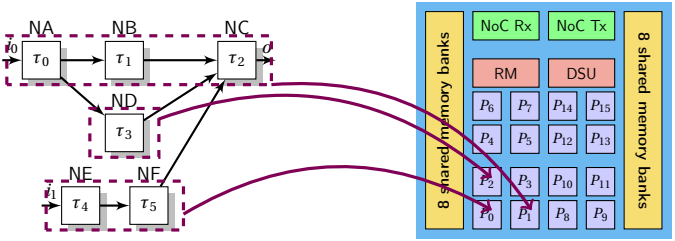


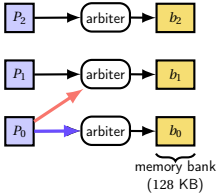
- Tasks mapping on cores
- Static non-preemptive scheduling
- Spatial Isolation

  different tasks go to different memory banks
- Interference from communications

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

Execution Model



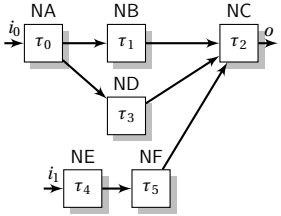


- Tasks mapping on cores
- Static non-preemptive scheduling
- Spatial Isolation

  different tasks go to different memory banks
- Interference from communications

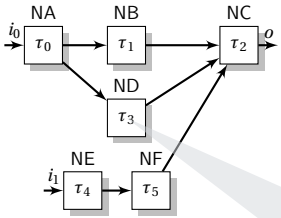
- Execution model:
  - execute in a "local" bank
  - write to a "remote" bank

Application Model

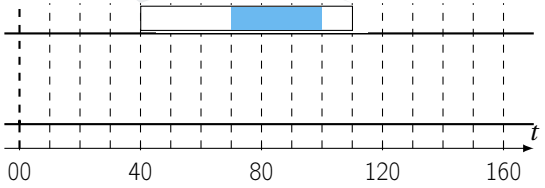


- Directed Acyclic Task Graph
- Mono-rate
- Fixed mapping and execution order

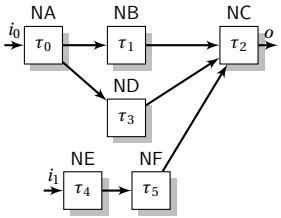
Application Model



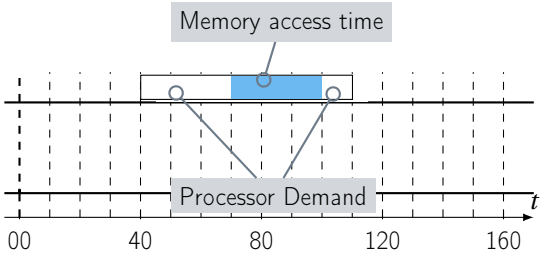
- Directed Acyclic Task Graph
- Mono-rate
- Fixed mapping and execution order
- **Each task τ_i:**



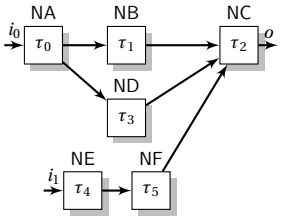
Application Model



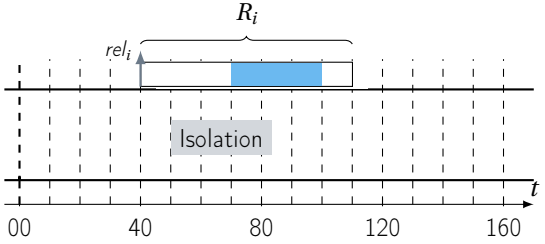
- Directed Acyclic Task Graph
- Mono-rate
- Fixed mapping and execution order
- **Each task τ_i:**
  - Input: Processor Demand, Memory Demand



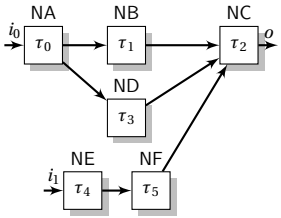
Application Model



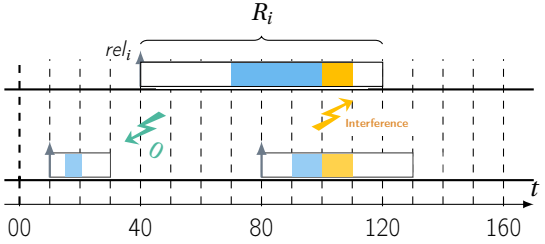
- Directed Acyclic Task Graph
- Mono-rate
- Fixed mapping and execution order
- **Each task τ_i:**
  - Input: Processor Demand, Memory Demand
  - Output: Release date (rel_i), response time (R_i)



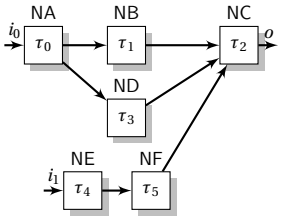
Application Model



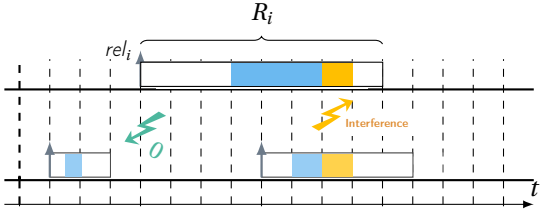
- Directed Acyclic Task Graph
- Mono-rate
- Fixed mapping and execution order
- **Each task τ_i:**
  - Input: Processor Demand, Memory Demand
  - Output: Release date (rel_i), response time (R_i)



Application Model



- Directed Acyclic Task Graph
- Mono-rate
- Fixed mapping and execution order
- **Each task τ_i:**
  - Input: Processor Demand, Memory Demand
  - Output: Release date (rel_i), response time (R_i)



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

Outline

Response Time Analysis

$$R = PD + I^{BUS}(R)$$

- Response Time

Response Time Analysis

$$R = PD + I^{BUS}(R)$$

- Response Time
  - Processor Demand

Response Time Analysis

$$R = PD + I^{BUS}(R)$$

- Response Time
  - Processor Demand
    - Bus Interference

(given a model of the bus arbiter)

Response Time Analysis

$$R = PD + I^{BUS}(R) + I^{PROC}(R) + I^{DRAM}(R)$$

- Response Time
  - Processor Demand
    - Bus Interference

(given a model of the bus arbiter)

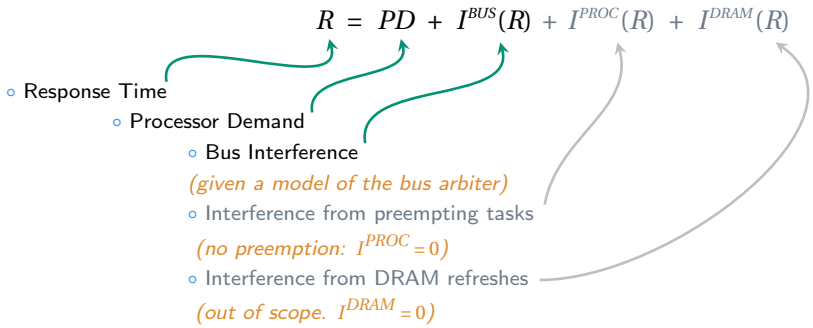
- Interference from preempting tasks

(no preemption: $I^{PROC} = 0$)

- Interference from DRAM refreshes

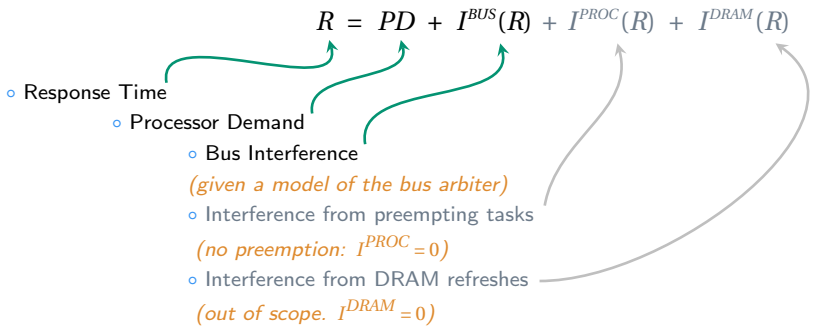
(out of scope. $I^{DRAM} = 0$)

Response Time Analysis



- Fix-point formula ⇒ iterative algorithm.

Response Time Analysis



- Fix-point formula ⇒ iterative algorithm.
- Multiple shared resources (memory banks)

Response Time Analysis

$$R = PD + I^{BUS}(R) + I^{PROC}(R) + I^{DRAM}(R)$$

- Response Time
  - Processor Demand
    - Bus Interference

(given a model of the bus arbiter)

- Interference from preempting tasks

(no preemption: $I^{PROC} = 0$)

- Interference from DRAM refreshes

(out of scope. $I^{DRAM} = 0$)

- Fix-point formula ⇒ iterative algorithm.
- Multiple shared resources (memory banks)

$$I^{BUS}(R) = \sum_{b \in B} I_b^{BUS}(R)$$

where B: a set of memory banks

Response Time Analysis

$$R = PD + I^{BUS}(R) + I^{PROC}(R) + I^{DRAM}(R)$$

- Response Time
  - Processor Demand
    - Bus Interference

(given a model of the bus arbiter)

- Interference from preempting tasks

(no preemption: $I^{PROC} = 0$)

- Interference from DRAM refreshes

(out of scope. $I^{DRAM} = 0$)

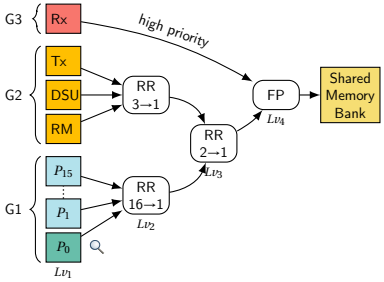
- Fix-point formula ⇒ iterative algorithm.
- Multiple shared resources (memory banks)

$$I^{BUS}(R) = \sum_{b \in B} I_b^{BUS}(R)$$

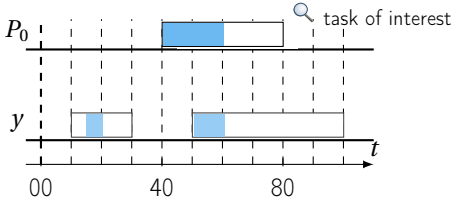
where B: a set of memory banks

🔍 Requires a model of the bus arbiter

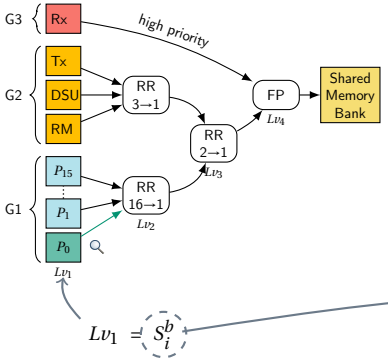
Model of the MPPA Bus



I_b^{BUS}: delay from all accesses + concurrent ones

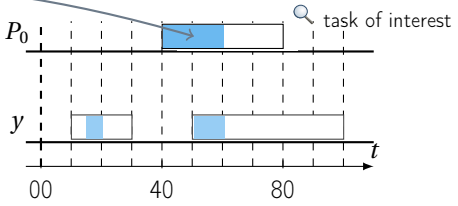


Model of the MPPA Bus

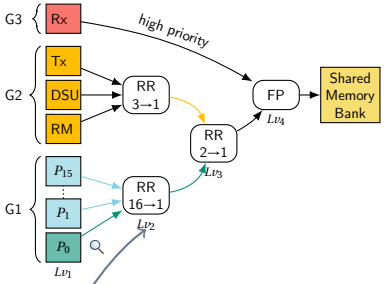


I_b^{BUS}: delay from all accesses + concurrent ones

S_i^b: number of accesses of task τ_i to bank b



Model of the MPPA Bus



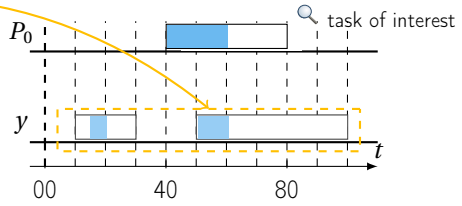
I_b^{BUS}: delay from all accesses + concurrent ones

S_i^b: number of accesses of task τ_i to bank b

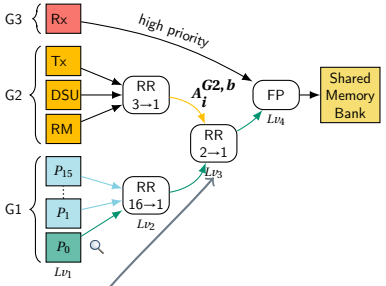
$A_i^{y,b}$: number of concurrent accesses from core y to bank b

$$Lv_1 = S_i^b$$

$$Lv_2 = Lv_1 + \sum_{y=1}^{15} \min(A_i^{y,b}, Lv_1)$$



Model of the MPPA Bus



$$Lv_1 = S_i^b$$

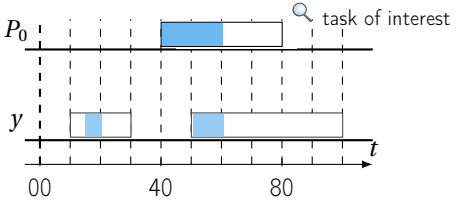
$$Lv_2 = Lv_1 + \sum_{y=1}^{15} \min(A_i^{y,b}, Lv_1)$$

$$Lv_3 = Lv_2 + \min(A_i^{G2,b}, Lv_2)$$

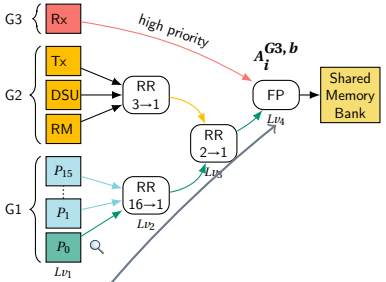
I_b^{BUS}: delay from all accesses + concurrent ones

S_i^b: number of accesses of task τ_i to bank b

$A_i^{y,b}$: number of concurrent accesses from core y to bank b



Model of the MPPA Bus



$$Lv_1 = S_i^b$$

$$Lv_2 = Lv_1 + \sum_{y=1}^{15} \min(A_i^{y,b}, Lv_1)$$

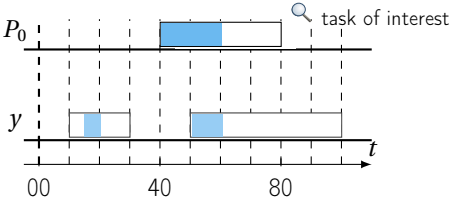
$$Lv_3 = Lv_2 + \min(A_i^{G2,b}, Lv_2)$$

$$Lv_4 = Lv_4 + A_i^{G3,b}$$

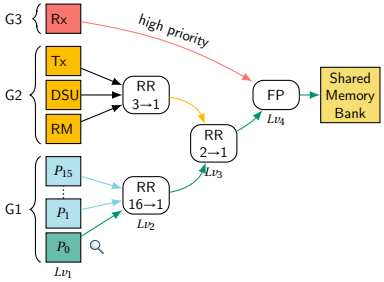
I_b^{BUS}: delay from all accesses + concurrent ones

S_i^b: number of accesses of task τ_i to bank b

$A_i^{y,b}$: number of concurrent accesses from core y to bank b



Model of the MPPA Bus



I_b^{BUS}: delay from all accesses + concurrent ones

S_i^b: number of accesses of task τ_i to bank b

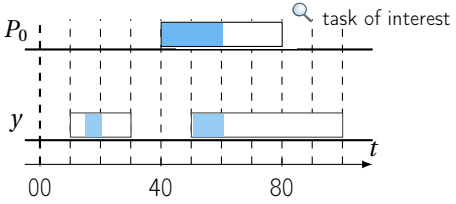
$A_i^{y,b}$: number of concurrent accesses from core y to bank b

$$Lv_1 = S_i^b$$

$$Lv_2 = Lv_1 + \sum_{y=1}^{15} \min(A_i^{y,b}, Lv_1)$$

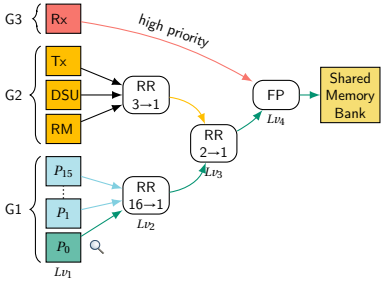
$$Lv_3 = Lv_2 + \min(A_i^{G2,b}, Lv_2)$$

$$Lv_4 = Lv_4 + A_i^{G3,b}$$



$$I_b^{BUS} = Lv_4 \times \text{Bus Delay}$$

Model of the MPPA Bus



I_b^{BUS}: delay from all accesses + concurrent ones

S_i^b: number of accesses of task τ_i to bank b

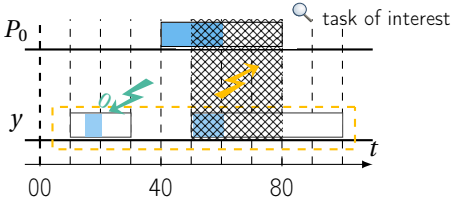
$A_i^{y,b}$: number of concurrent accesses from core y to bank b

$$Lv_1 = S_i^b$$

$$Lv_2 = Lv_1 + \sum_{y=1}^{15} \min(A_i^{y,b}, Lv_1)$$

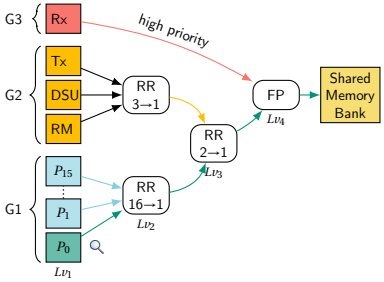
$$Lv_3 = Lv_2 + \min(A_i^{G2,b}, Lv_2)$$

$$Lv_4 = Lv_4 + A_i^{G3,b}$$



$I_b^{BUS} = Lv_4 \times$ Bus Delay

Model of the MPPA Bus



I_b^{BUS}: delay from all accesses + concurrent ones

S_i^b: number of accesses of task τ_i to bank b

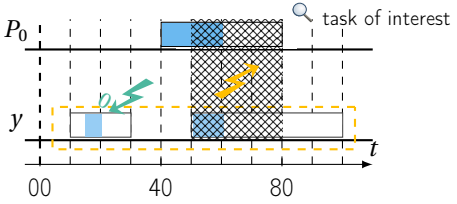
$A_i^{y,b}$: number of concurrent accesses from core y to bank b

$$Lv_1 = S_i^b$$

$$Lv_2 = Lv_1 + \sum_{y=1}^{15} \min(A_i^{y,b}, Lv_1)$$

$$Lv_3 = Lv_2 + \min(A_i^{G2,b}, Lv_2)$$

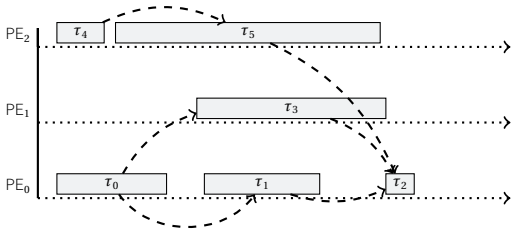
$$Lv_4 = Lv_4 + A_i^{G3,b}$$



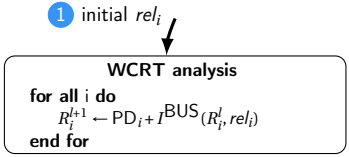
⚠ $A_i^{y,b}$ depends on rel_i and R_i

$I_b^{BUS} = Lv_4 \times$ Bus Delay

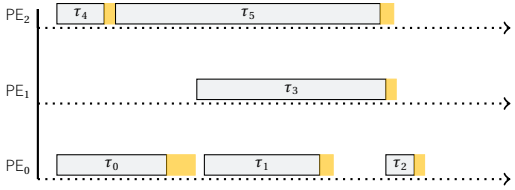
Response Time Analysis with Dependencies

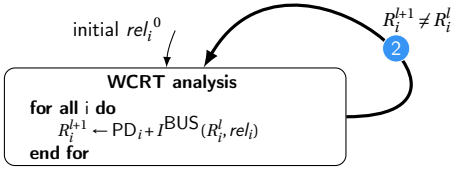


1 Start with initial release dates.



Response Time Analysis with Dependencies

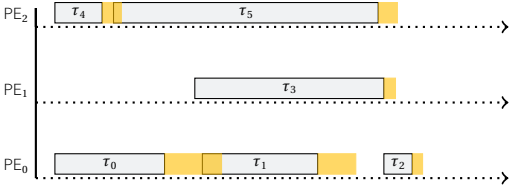


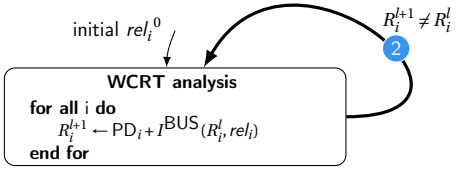


1. Start with initial release dates.
2. Compute response times

...

Response Time Analysis with Dependencies

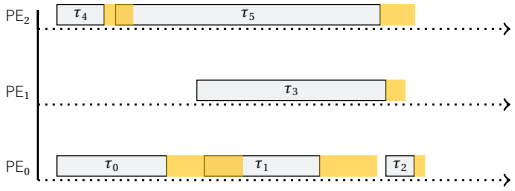




1. Start with initial release dates.
2. Compute response times

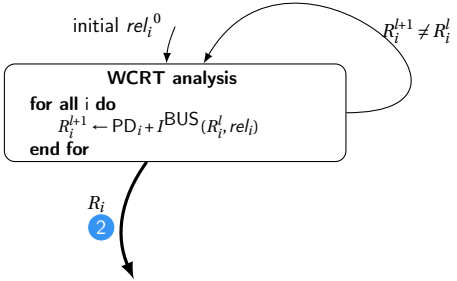
... ...

Response Time Analysis with Dependencies

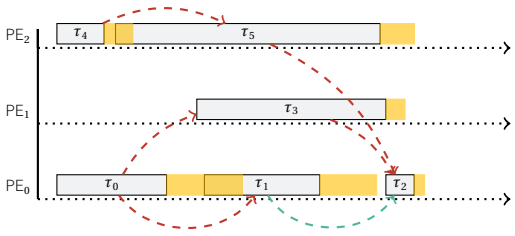


1. Start with initial release dates.
2. Compute response times

   ... ... a fixed-point is reached!

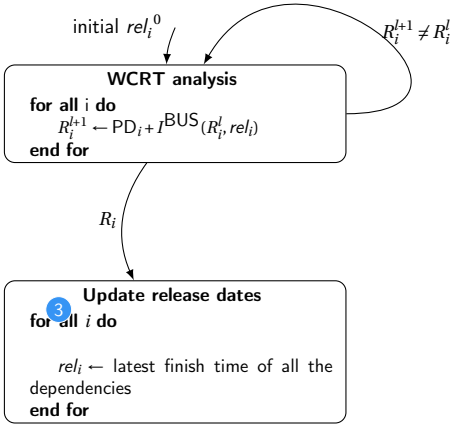


Response Time Analysis with Dependencies

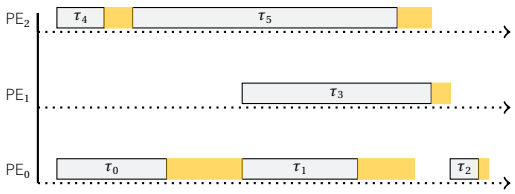


1. Start with initial release dates.
2. Compute response times

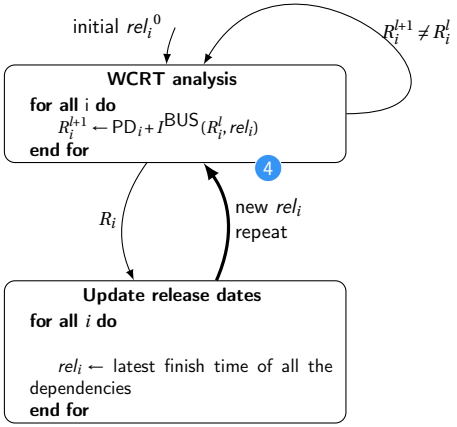
   ... ... a fixed-point is reached!
3. Update the release dates.



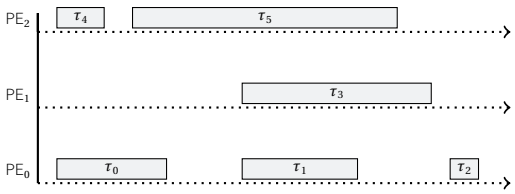
Response Time Analysis with Dependencies



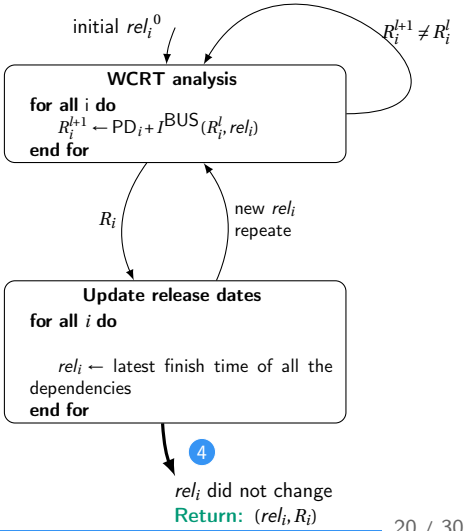
1. Start with initial release dates.
2. Compute response times
   ... ... a fixed-point is reached!
3. Update the release dates.
4. Repeat until no release date changes
   (another fixed-point iteration).



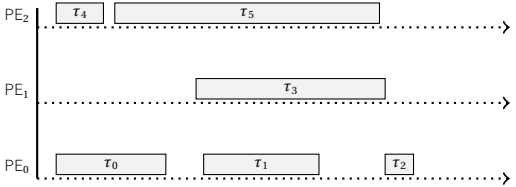
Response Time Analysis with Dependencies



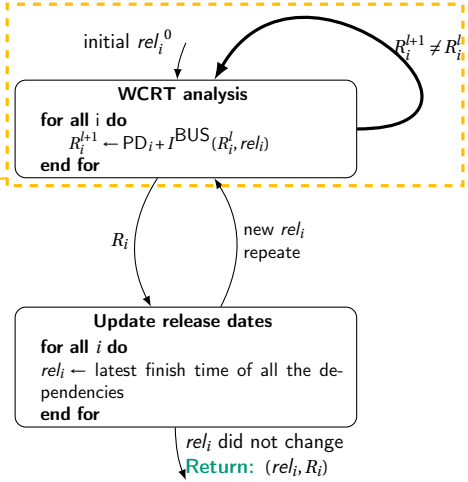
1. Start with initial release dates.
2. Compute response times
   ... ... a fixed-point is reached!
3. Update the release dates.
4. Repeat until no release date changes
   (another fixed-point iteration).



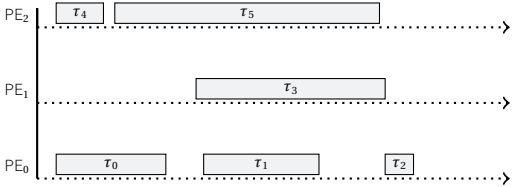
Proof of Convergence Toward a Fixed-point



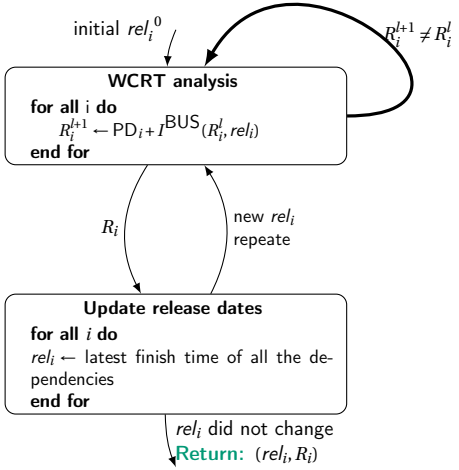
- Convergence of the 1^{st} fixed-point iteration:



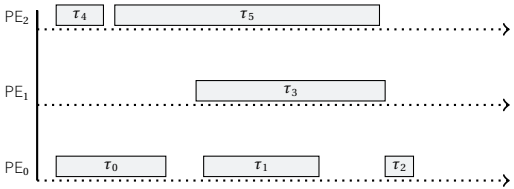
Proof of Convergence Toward a Fixed-point



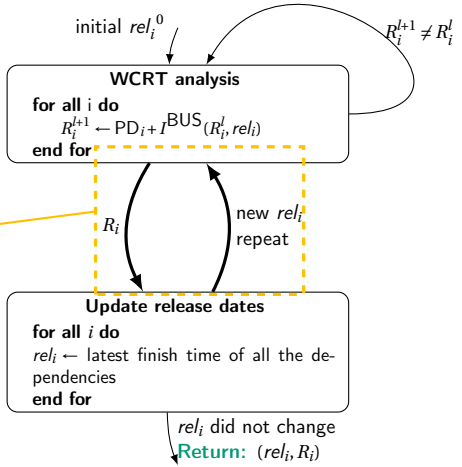
- Convergence of the 1^{st} fixed-point iteration:
  - Monotonic and bounded ✓



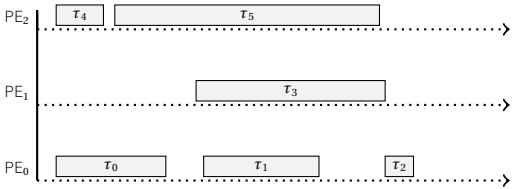
Proof of Convergence Toward a Fixed-point



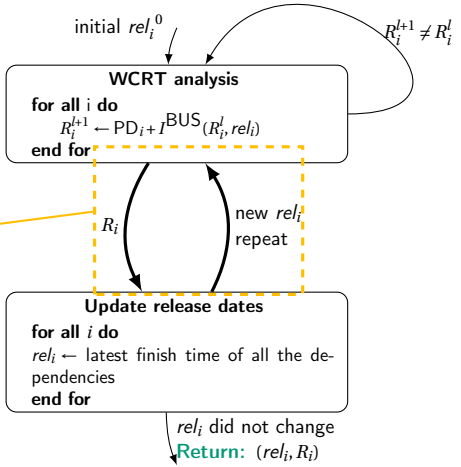
- Convergence of the 1^{st} fixed-point iteration:
  - Monotonic and bounded ✓
- Convergence of the 2^{nd} fixed-point iteration:



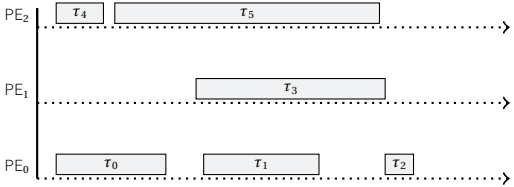
Proof of Convergence Toward a Fixed-point



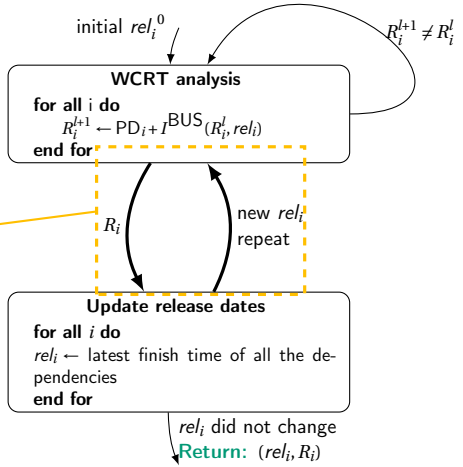
- Convergence of the 1^{st} fixed-point iteration:
  - Monotonic and bounded ✓
- Convergence of the 2^{nd} fixed-point iteration:
  - no monotonicity: R_i and rel_i may grow or shrink at each iteration. ?



Proof of Convergence Toward a Fixed-point



- Convergence of the 1^{st} fixed-point iteration:
  - Monotonic and bounded ✓
- Convergence of the 2^{nd} fixed-point iteration:
  - no monotonicity: R_i and rel_i may grow or shrink at each iteration. ?

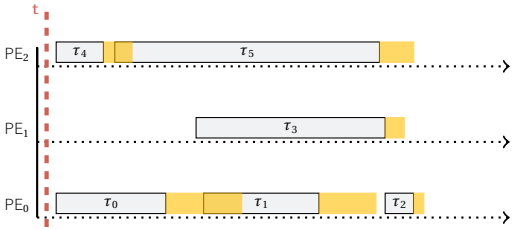


Theorem

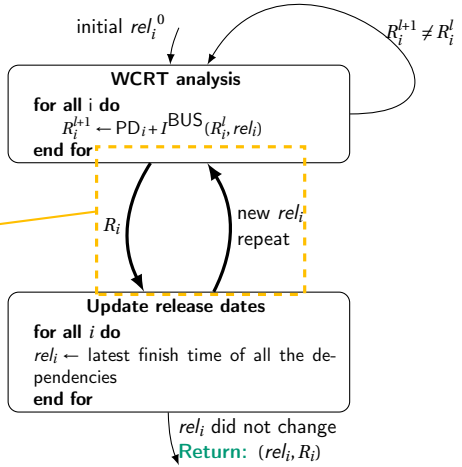
At each iteration, at least one task finds its final release date.

Full proof in our technical report:
http://www-verimag.imag.fr/TR/TR-2016-1.pdf

Proof of Convergence Toward a Fixed-point



- Convergence of the 1^{st} fixed-point iteration:
  - Monotonic and bounded ✓
- Convergence of the 2^{nd} fixed-point iteration:
  - no monotonicity: R_i and rel_i may grow or shrink at each iteration. ?

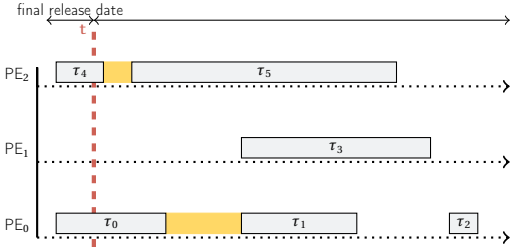


Theorem

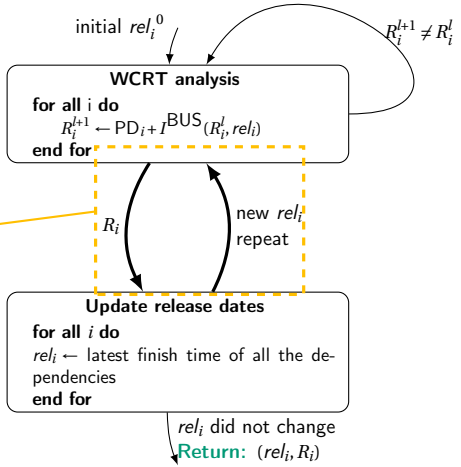
At each iteration, at least one task finds its final release date.

Full proof in our technical report:
http://www-verimag.imag.fr/TR/TR-2016-1.pdf

Proof of Convergence Toward a Fixed-point



- Convergence of the 1^{st} fixed-point iteration:
  - Monotonic and bounded ✓
- Convergence of the 2^{nd} fixed-point iteration:
  - no monotonicity: R_i and rel_i may grow or shrink at each iteration. ?

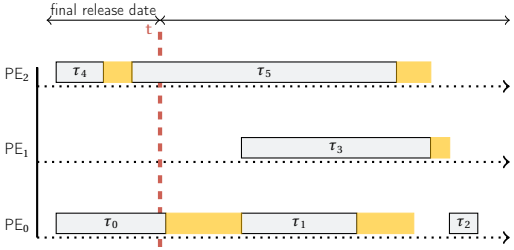


Theorem

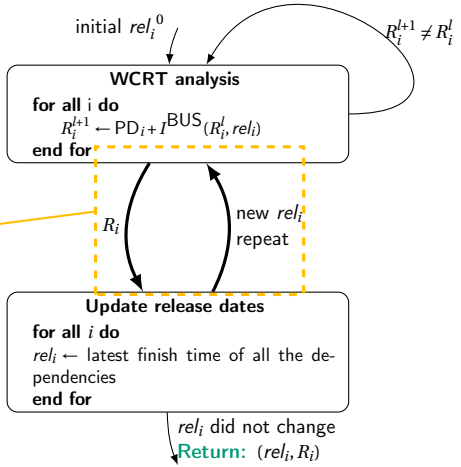
At each iteration, at least one task finds its final release date.

Full proof in our technical report:
http://www-verimag.imag.fr/TR/TR-2016-1.pdf

Proof of Convergence Toward a Fixed-point



- Convergence of the 1^{st} fixed-point iteration:
  - Monotonic and bounded ✓
- Convergence of the 2^{nd} fixed-point iteration:
  - no monotonicity: R_i and rel_i may grow or shrink at each iteration. ?

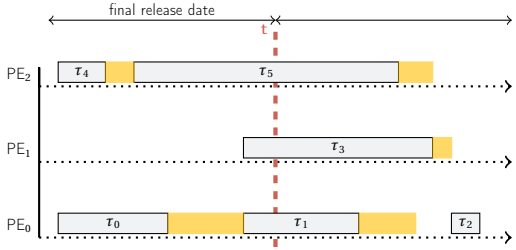


Theorem

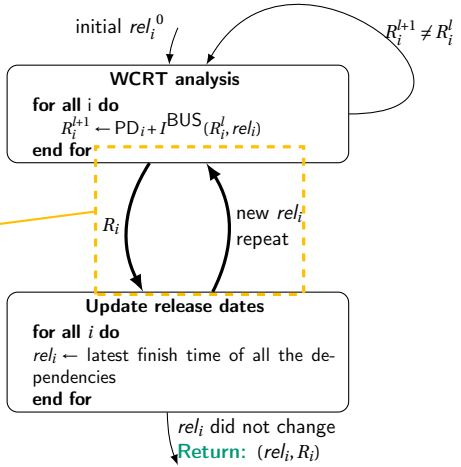
At each iteration, at least one task finds its final release date.

Full proof in our technical report:
http://www-verimag.imag.fr/TR/TR-2016-1.pdf

Proof of Convergence Toward a Fixed-point



- Convergence of the 1^{st} fixed-point iteration:
  - Monotonic and bounded ✓
- Convergence of the 2^{nd} fixed-point iteration:
  - no monotonicity: R_i and rel_i may grow or shrink at each iteration. ?

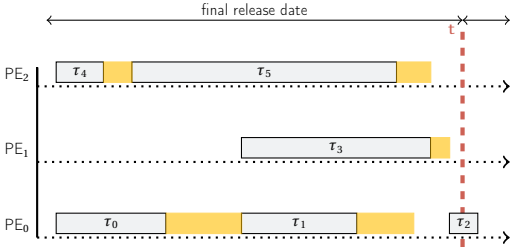


Theorem

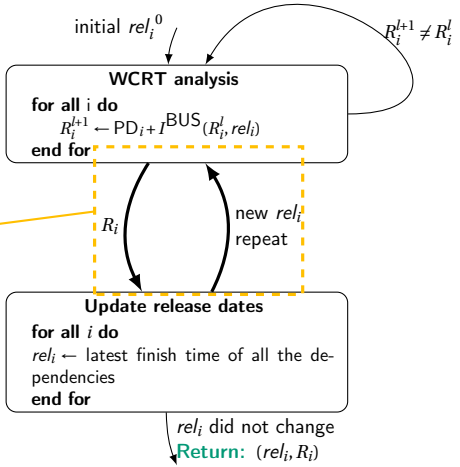
At each iteration, at least one task finds its final release date.

Full proof in our technical report:
http://www-verimag.imag.fr/TR/TR-2016-1.pdf

Proof of Convergence Toward a Fixed-point



- Convergence of the 1^{st} fixed-point iteration:
  - Monotonic and bounded ✓
- Convergence of the 2^{nd} fixed-point iteration:
  - no monotonicity: R_i and rel_i may grow or shrink at each iteration. ?



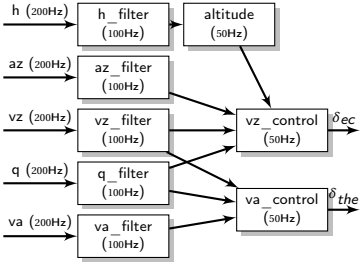
Theorem

At each iteration, at least one task finds its final release date.

Full proof in our technical report:
http://www-verimag.imag.fr/TR/TR-2016-1.pdf

Outline

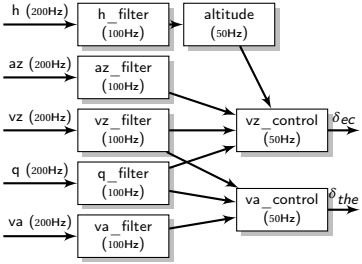
Evaluation: ROSACE Case Study[1]



- Flight management system controller

[1] Pagetti et al., RTAS 2014

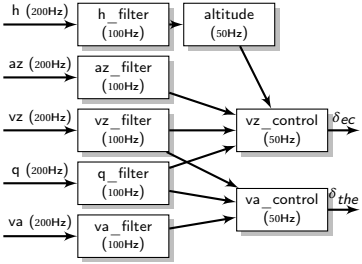
Evaluation: ROSACE Case Study[1]

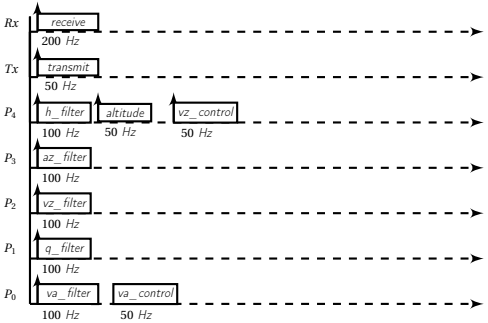


- Flight management system controller
- Receive from sensors and transmit to actuators

[1] Pagetti et al., RTAS 2014

Evaluation: ROSACE Case Study[1]



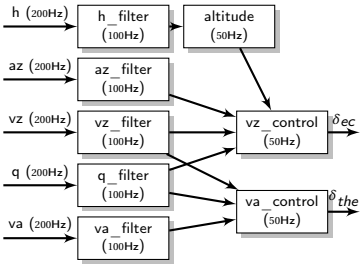


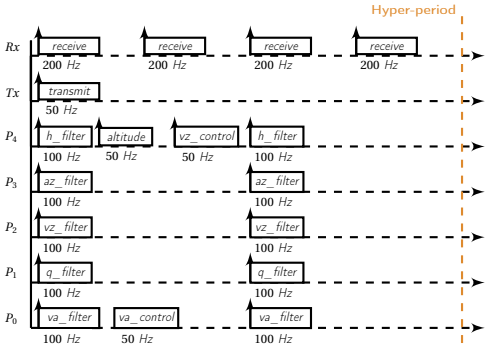
- Flight management system controller
- Receive from sensors and transmit to actuators
- **Assumptions:**

  Tasks are mapped on 5 cores
  Debug Support Unit is disabled
  Context switches are over-approximated constants

[1] Pagetti et al., RTAS 2014

Evaluation: ROSACE Case Study[1]





- Flight management system controller
- Receive from sensors and transmit to actuators
- **Assumptions:**

  Tasks are mapped on 5 cores
  Debug Support Unit is disabled
  Context switches are over-approximated constants

[1] Pagetti et al., RTAS 2014

Evaluation: ROSACE Case Study

Task	Processor Demand (cycles)	Memory Demand (accesses)
altitude	275	22
az_filter	274	22
h_filter	326	24
va_control	303	24
va_filter	301	23
vz_control	320	25
vz_filter	334	25

Table: Task profiles of the FMS controller

- Profile obtained from measurements

Evaluation: ROSACE Case Study

Task	Processor Demand (cycles)	Memory Demand (accesses)
altitude	275	22
az_filter	274	22
h_filter	326	24
va_control	303	24
va_filter	301	23
vz_control	320	25
vz_filter	334	25

Table: Task profiles of the FMS controller

- Profile obtained from measurements
- Memory Demand: data and instruction cache misses + communications

Evaluation: ROSACE Case Study

Task	Processor Demand (cycles)	Memory Demand (accesses)
altitude	275	22
az_filter	274	22
h_filter	326	24
va_control	303	24
va_filter	301	23
vz_control	320	25
vz_filter	334	25

Table: Task profiles of the FMS controller

- Profile obtained from measurements
- Memory Demand: data and instruction cache misses + communications
- Moreover:
  - *NoC Rx*: writes 5 words
  - *NoC Tx*: reads 2 words

Evaluation: ROSACE Case Study

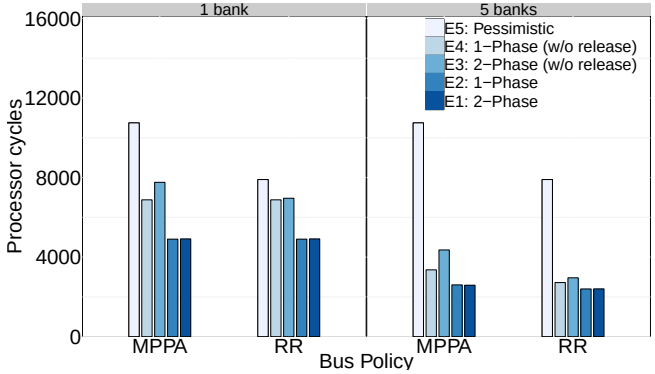
Task	Processor Demand (cycles)	Memory Demand (accesses)
altitude	275	22
az_filter	274	22
h_filter	326	24
va_control	303	24
va_filter	301	23
vz_control	320	25
vz_filter	334	25

Table: Task profiles of the FMS controller

- Profile obtained from measurements
- Memory Demand: data and instruction cache misses + communications
- Moreover:
  - *NoC Rx*: writes 5 words
  - *NoC Tx*: reads 2 words

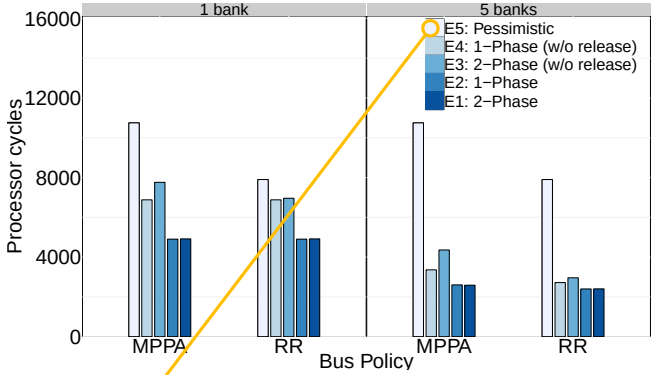
Experiments: Find the smallest schedulable hyper-period

Evaluation: Experiments



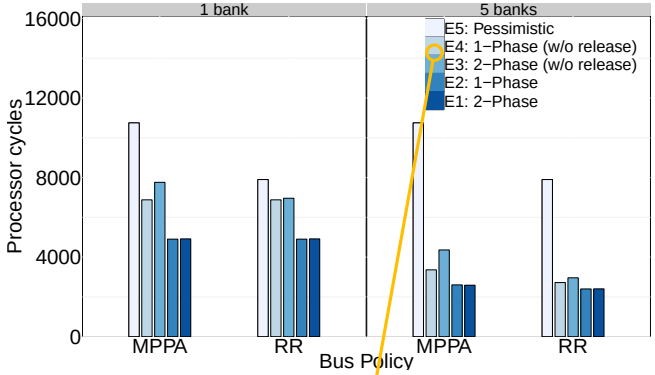
Smallest schedulable hyper-period

Evaluation: Experiments



E5: All accesses interfere

Evaluation: Experiments

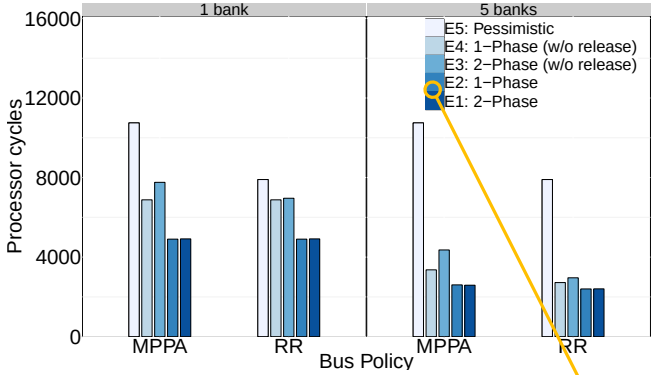


Smallest schedulable hyper-period

E4, E3: We don't use the release dates

E5: All accesses interfere

Evaluation: Experiments



Smallest schedulable hyper-period

E5: All accesses interfere

E4, E3: We don't use the release dates

E2, E1: Our approach. We use the release dates

Evaluation: Experiments

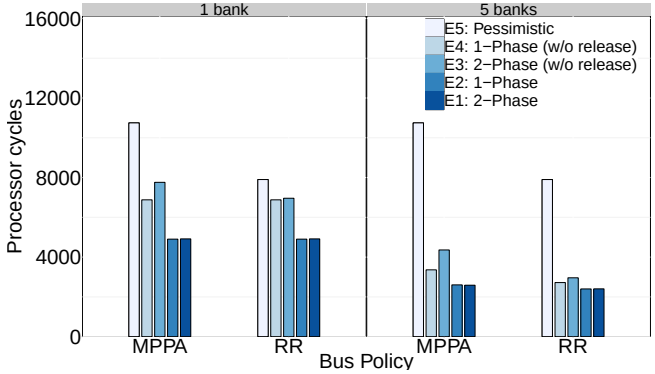
1-Phase model

memory access pattern 

2-Phase model

memory access pattern 

- Phases are modeled as sub-tasks



Smallest schedulable hyper-period

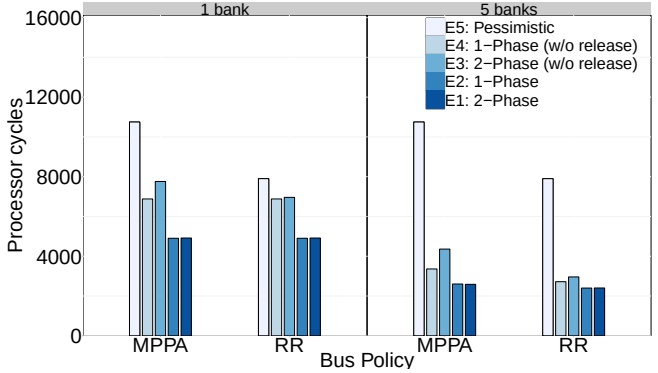
E5: All accesses interfere

E4, E3: We don't use the release dates

E2, E1: Our approach. We use the release dates

Evaluation: Experiments

Taking into account the memory banks improves the analysis with a factor in $[1.77, 2.52]$



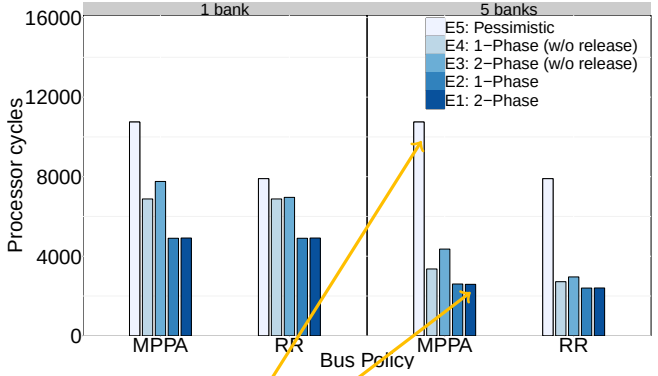
Smallest schedulable hyper-period

	E5/E1	E5/E2	E3/E1	E4/E2	E2/E1	E4/E3
MPPA	4.15	4.12	1.68	1.29	~1.01	0.77
RR	3.3	3.29	1.24	1.13	~1.01	0.91

Speedup factors

Evaluation: Experiments

Taking into account the memory banks improves the analysis with a factor in $[1.77, 2.52]$

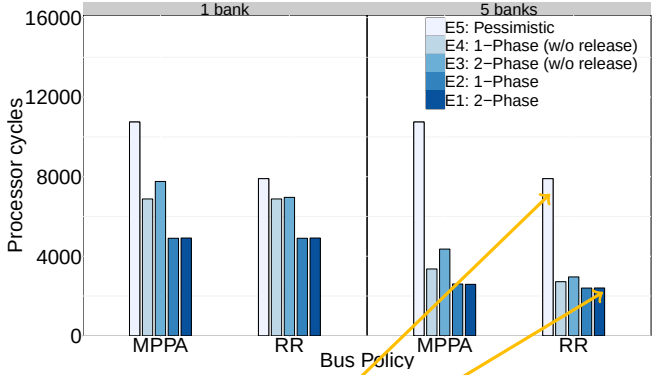


	E5/E1	E5/E2	E3/E1	E4/E2	E2/E1	E4/E3
MPPA	4.15	4.12	1.68	1.29	~1.01	0.77
RR	3.3	3.29	1.24	1.13	~1.01	0.91

Speedup factors

Evaluation: Experiments

Taking into account the memory banks improves the analysis with a factor in $[1.77, 2.52]$

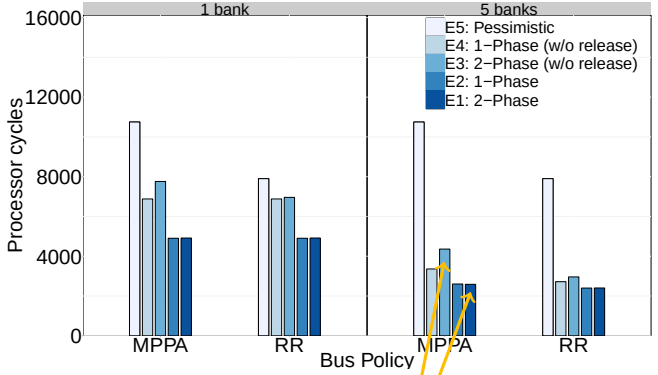


	E5/E1	E5/E2	E3/E1	E4/E2	E2/E1	E4/E3
MPPA	4.15	4.12	1.68	1.29	~1.01	0.77
RR	3.3	3.29	1.24	1.13	~1.01	0.91

Speedup factors

Evaluation: Experiments

Taking into account the memory banks improves the analysis with a factor in $[1.77, 2.52]$

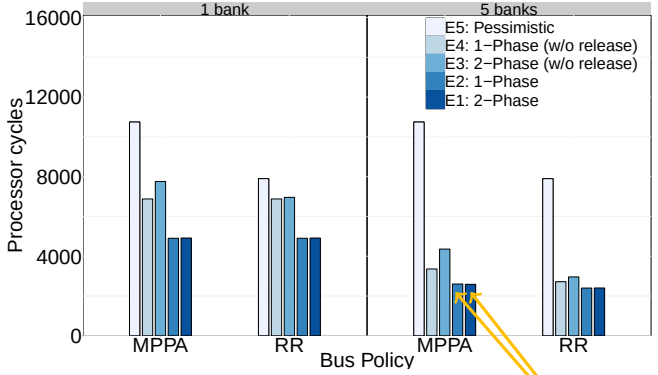


	E5/E1	E5/E2	E3/E1	E4/E2	E2/E1	E4/E3
MPPA	4.15	4.12	1.68	1.29	~1.01	0.77
RR	3.3	3.29	1.24	1.13	~1.01	0.91

Speedup factors

Evaluation: Experiments

Taking into account the memory banks improves the analysis with a factor in $[1.77, 2.52]$

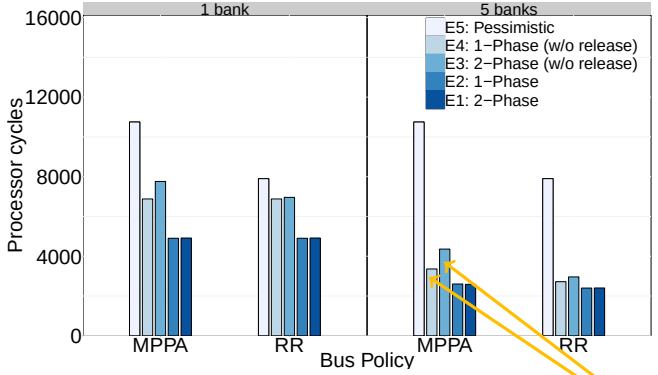


	E5/E1	E5/E2	E3/E1	E4/E2	E2/E1	E4/E3
MPPA	4.15	4.12	1.68	1.29	~1.01	0.77
RR	3.3	3.29	1.24	1.13	~1.01	0.91

Speedup factors

Evaluation: Experiments

Taking into account the memory banks improves the analysis with a factor in $[1.77, 2.52]$



	E5/E1	E5/E2	E3/E1	E4/E2	E2/E1	E4/E3
MPPA	4.15	4.12	1.68	1.29	~1.01	0.77
RR	3.3	3.29	1.24	1.13	~1.01	0.91

Speedup factors

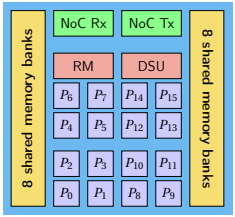
Outline

Conclusion

- 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
  - Non-trivial proof of termination

Future Work

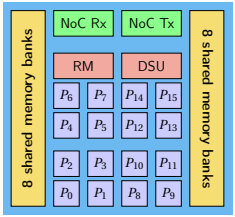
- Model of the Resource Manager.



Future Work

- Model of the Resource Manager.

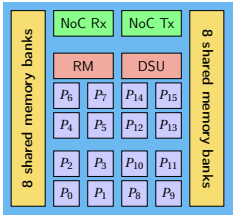
tighter estimation of context switches and other interrupts



Future Work

- Model of the Resource Manager.
- Model of the NoC accesses.

tighter estimation of context switches and other interrupts

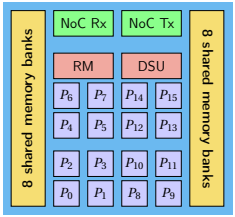


Future Work

- Model of the Resource Manager.
- Model of the NoC accesses.

tighter estimation of context switches and other interrupts

use the output of any NoC analysis

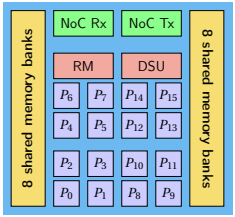


Future Work

- Model of the Resource Manager.
- Model of the NoC accesses.
- Memory access pipelining.

tighter estimation of context switches and other interrupts

use the output of any NoC analysis



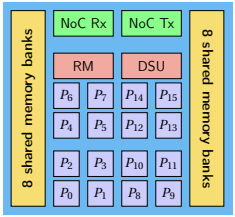
Future Work

- Model of the Resource Manager.
- Model of the NoC accesses.
- Memory access pipelining.

tighter estimation of context switches and other interrupts

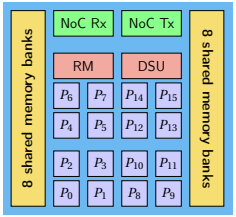
use the output of any NoC analysis

current assumption: bus delay is 10 cycles



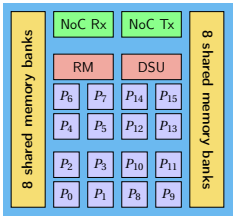
Future Work

- Model of the Resource Manager.
  - tighter estimation of context switches and other interrupts
- Model of the NoC accesses.
  - use the output of any NoC analysis
- Memory access pipelining.
  - current assumption: bus delay is 10 cycles
- Model Blocking and non-blocking accesses.



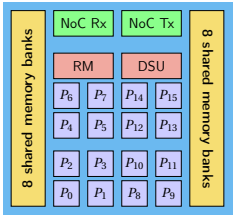
Future Work

- Model of the Resource Manager.
  - tighter estimation of context switches and other interrupts
- Model of the NoC accesses.
  - use the output of any NoC analysis
- Memory access pipelining.
  - current assumption: bus delay is 10 cycles
- Model Blocking and non-blocking accesses.
  - reads are blocking
    writes are non-blocking



Future Work

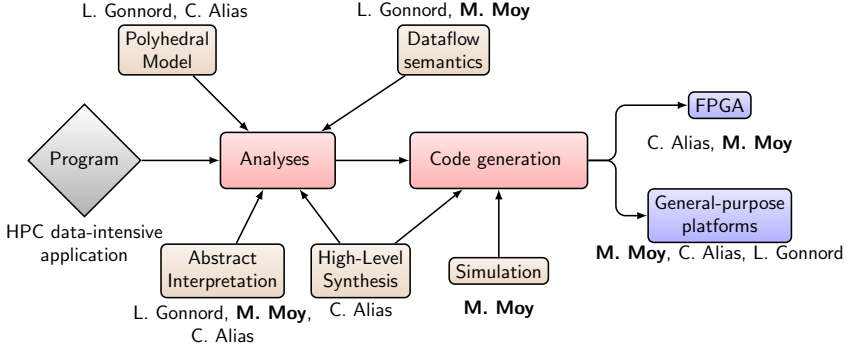
- Model of the Resource Manager.
  - tighter estimation of context switches and other interrupts
- Model of the NoC accesses.
  - use the output of any NoC analysis
- Memory access pipelining.
  - current assumption: bus delay is 10 cycles
- Model Blocking and non-blocking accesses.
  - reads are blocking
    writes are non-blocking



Questions?

Outline

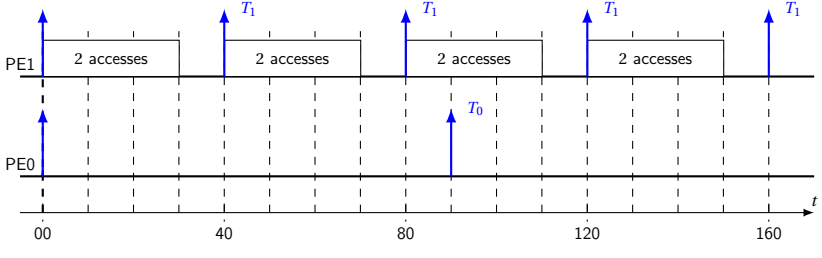
Compilation and
Analysis for
Software and
Hardware



BACKUP

Multicore Response Time Analysis

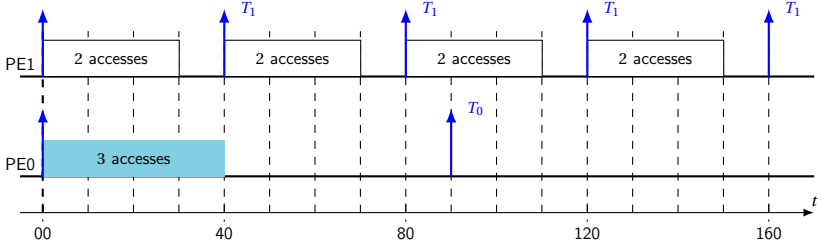
Example: Fixed Priority bus arbiter, PE1 > PE0
Bus access delay = 10



[1] Altmeyer et al., RTNS 2015

Multicore Response Time Analysis

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



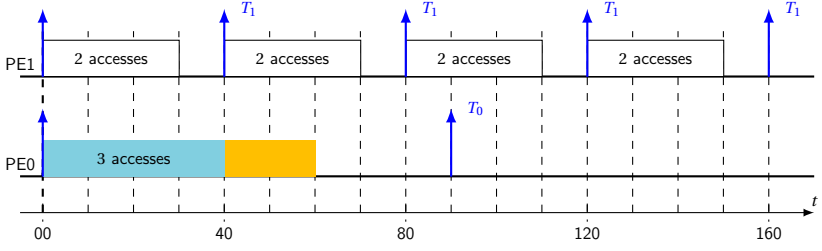
- Task of interest running on $PE0$:

  $R_0 = 10 + 3 \times 10$ (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 $PE0$:

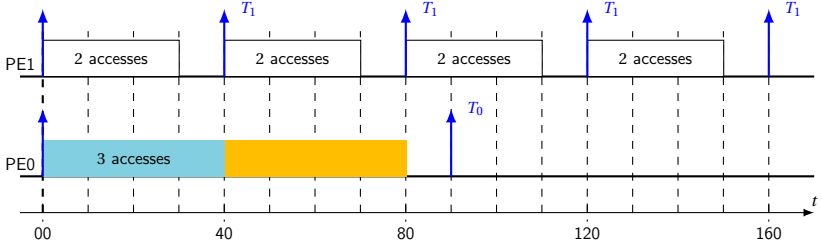
  $R_0 = 10 + 3 \times 10$ (response time in isolation)

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

[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)

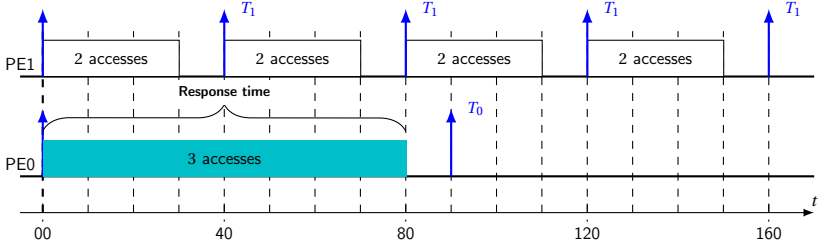
  $R_1 = 10 + 3 \times 10 + 2 \times 10 = 60$

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

[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