Response Time Analysis of Synchronous Data Flow Programs on a Many-Core Processor

Hamza Rihani, Matthieu Moy, Claire Maiza, Robert I. Davis, Sebastian Altmeyer

RTNS'16, October 19, 2016

RTNS * Artifact * Evaluated *

Consistent * Complete * Well documented * Easy to reuse *

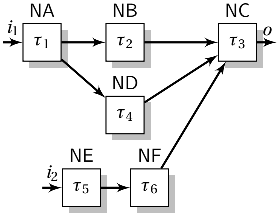
Verimag

UNIVERSITÉ Grenoble Alpes

RTS York

University of Amsterdam

Execution of Synchronous Data Flow Programs



High level representation

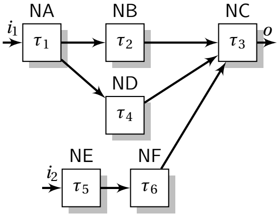
Single-core

code generation

```
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;
}
```

static non-preemptive scheduling

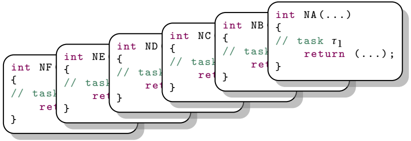
Execution of Synchronous Data Flow Programs



High level representation

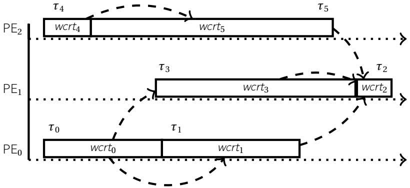
Multi/Many-core

code generation

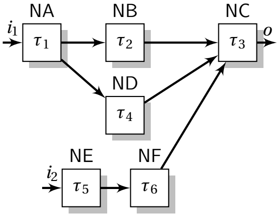

int NF
{
// tas
ret
}
int NE
{
// task
ret
}
int ND
{
// task
ret
}
int NC
{
// task
ret
}
int NB
{
// task
ret
}

```
int NA(...)
{
// task τ1
   return (...);
}
```

static non-preemptive scheduling



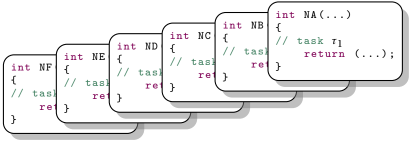
Execution of Synchronous Data Flow Programs



High level representation

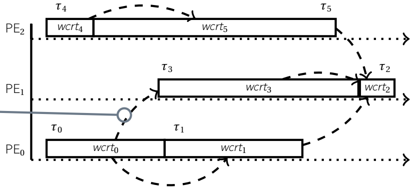
Multi/Many-core

code generation



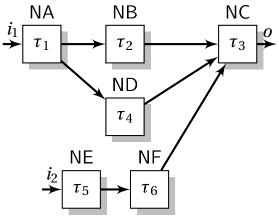
```
int NA(...)
{
// task τ1
    return (...);
}
```

static non-preemptive scheduling



✓ Respect the dependency constraints

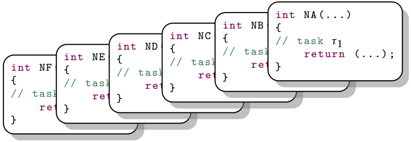
Execution of Synchronous Data Flow Programs



High level representation

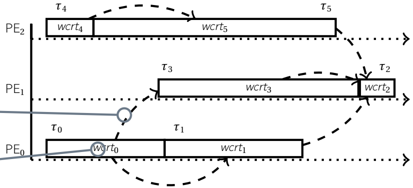
Multi/Many-core

code generation



```
int NA(...)
{
// task τ1
      return (...);
}
```

static non-preemptive scheduling

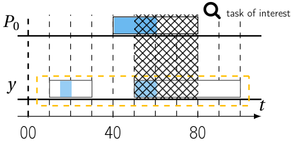


✓ Respect the dependency constraints

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

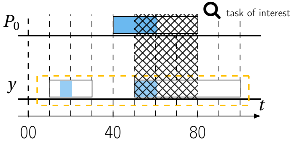
Contributions

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

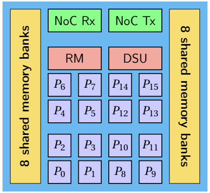


Contributions

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

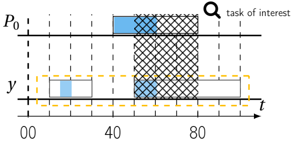


2. Model of a multi-level arbiter to the shared memory

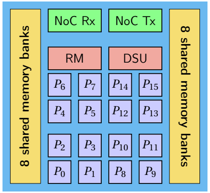


Contributions

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

1. Motivation and Context
2. Models Definition
   - Architecture Model
   - Execution Model
   - Application Model
3. Multicore Response Time Analysis of SDF Programs
4. Evaluation
5. Conclusion and Future Work

Outline

1. Motivation and Context

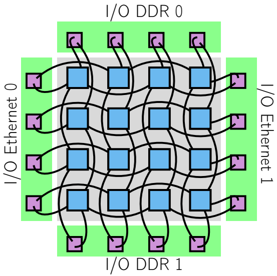
2. Models Definition
   - Architecture Model
   - Execution Model
   - Application Model

3. Multicore Response Time Analysis of SDF Programs

4. Evaluation

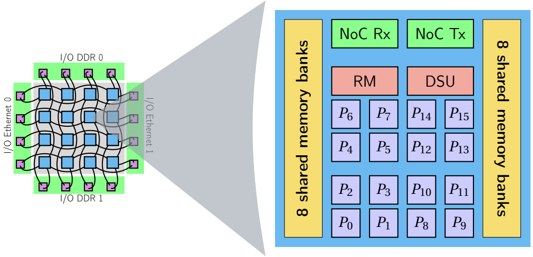
5. Conclusion and Future Work

Architecture Model



- Kalray MPPA 256 Bostan
- 16 compute clusters + 4 I/O clusters
- Dual NoC

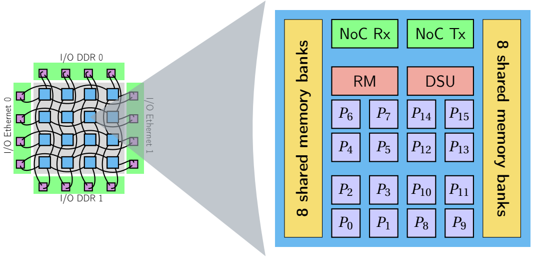
Architecture Model



Per cluster:

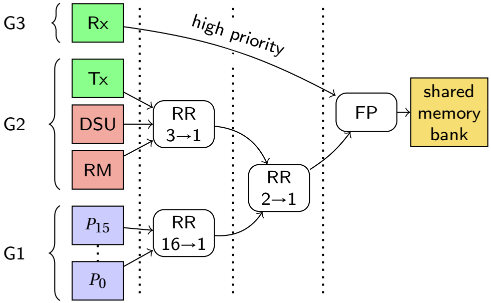
- 16 cores + 1 Resource Manager
- NoC Tx, NoC Rx, Debug Unit
- 16 shared memory banks (total size: 2 MB)

Architecture Model

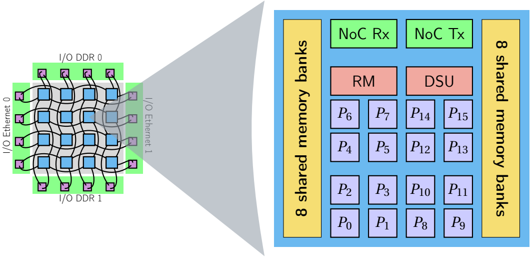


Per cluster:

- 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

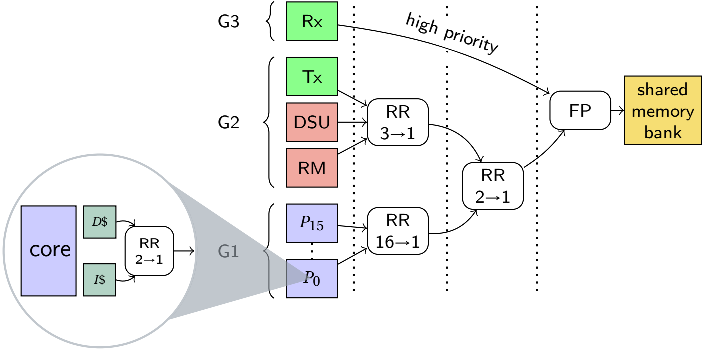


Architecture Model

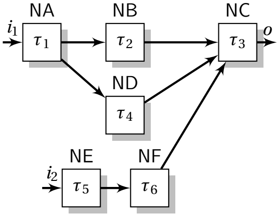


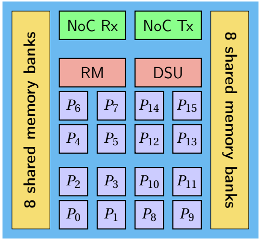
Per cluster:

- 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

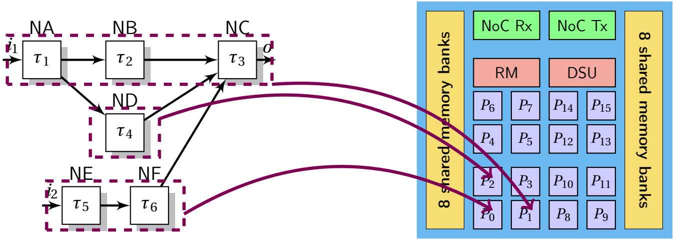


Execution Model



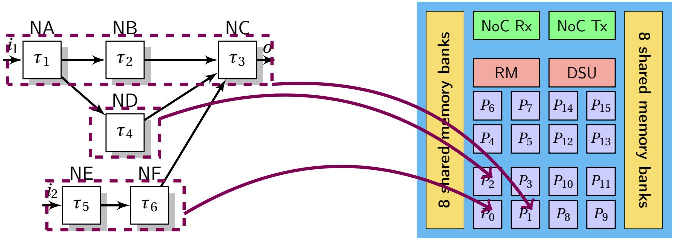

8 shared memory banks
NoC Rx
NoC Tx
RM
DSU
P_6 P_7 P_{14} P_{15}
P_4 P_5 P_{12} P_{13}
P_2 P_3 P_{10} P_{11}
P_0 P_1 P_8 P_9
8 shared memory banks

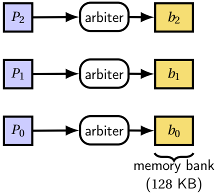
Execution Model



- Tasks mapping on cores
- Static non-preemptive scheduling

Execution Model

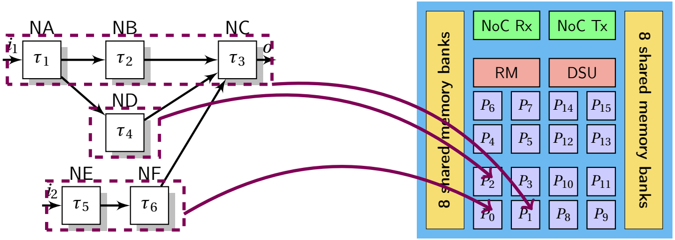


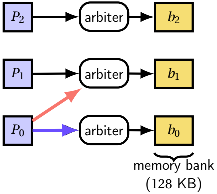


- Tasks mapping on cores
- Static non-preemptive scheduling
- Spatial Isolation

  different tasks go to different memory banks

Execution Model

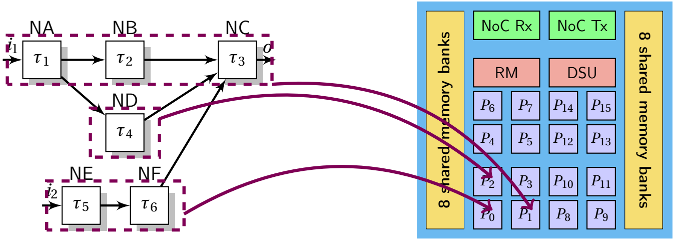


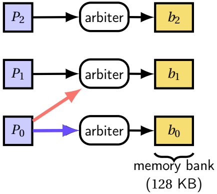


- Tasks mapping on cores
- Static non-preemptive scheduling
- Spatial Isolation

  different tasks go to different memory banks
- Interference from communications

Execution Model



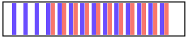


- Tasks mapping on cores
- Static non-preemptive scheduling
- Spatial Isolation

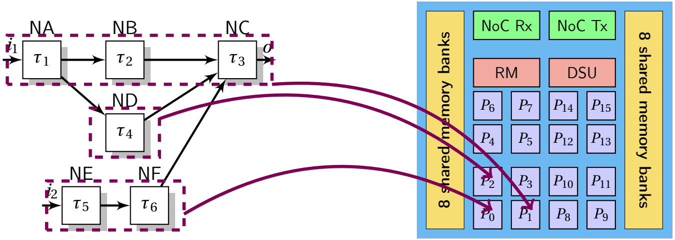
  different tasks go to different memory banks
- Interference from communications
- Execution model:
  - execute in a “local” bank
  - write to a “remote” bank

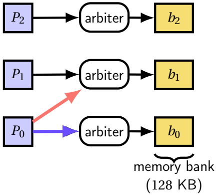
Single phase: execute *and* write data.

memory access pattern



Execution Model





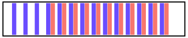
- Tasks mapping on cores
- Static non-preemptive scheduling
- Spatial Isolation

  different tasks go to different memory banks

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

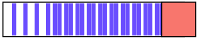
Single phase: execute *and* write data.

memory access pattern

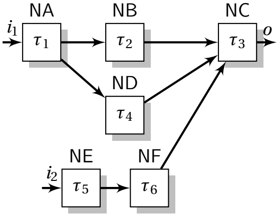


Two phases: execute *then* write data.

memory access pattern

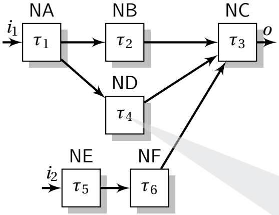


Application Model



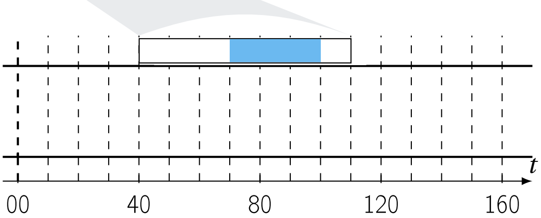
- Direct Acyclic Task Graph
- Mono-rate (or at least harmonic rates)
- Fixed mapping and execution order

Application Model

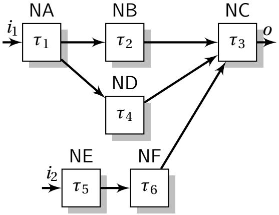


- Direct Acyclic Task Graph
- Mono-rate (or at least harmonic rates)
- Fixed mapping and execution order

Each task τ_i:



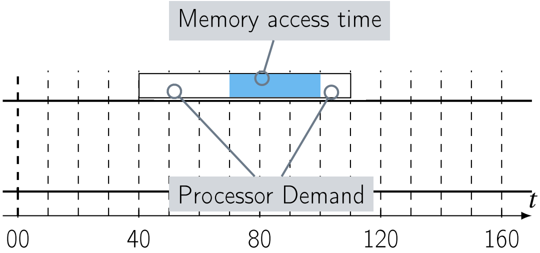
Application Model



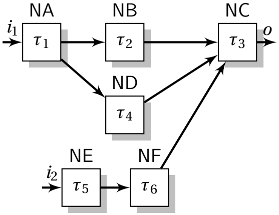
- Direct Acyclic Task Graph
- Mono-rate (or at least harmonic rates)
- Fixed mapping and execution order

Each task τ_i:

- Processor Demand, Memory Demand



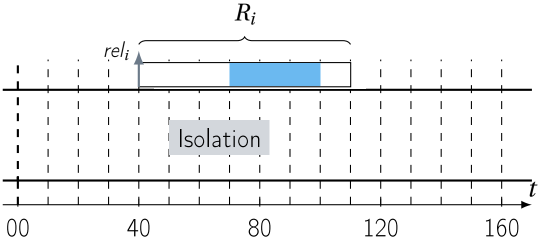
Application Model



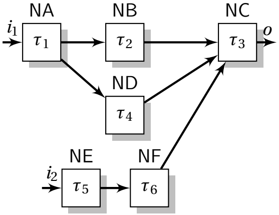
- Direct Acyclic Task Graph
- Mono-rate (or at least harmonic rates)
- Fixed mapping and execution order

Each task τ_i:

- Processor Demand, Memory Demand
- Release date (*rel*$_i$), response time (R_i)



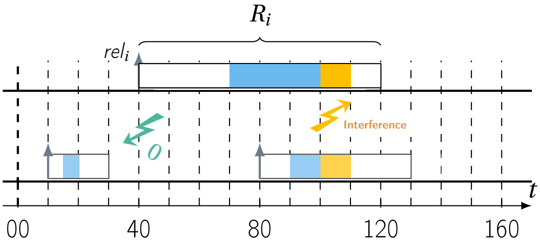
Application Model



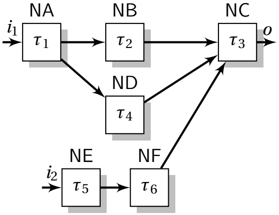
- Direct Acyclic Task Graph
- Mono-rate (or at least harmonic rates)
- Fixed mapping and execution order

Each task τ_i:

- Processor Demand, Memory Demand
- Release date (rel_i), response time (R_i)



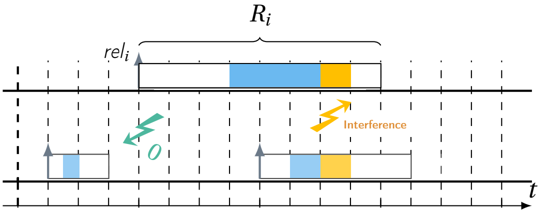
Application Model



- Direct Acyclic Task Graph
- Mono-rate (or at least harmonic rates)
- Fixed mapping and execution order

Each task τ_i:

- Processor Demand, Memory Demand
- Release date (*rel*$_i$), response time (R_i)



🔍 Find R_i (including the interference)

🔍 Find *rel*$_i$ respecting precedence constraints

Outline

1 Motivation and Context

2 Models Definition

- Architecture Model
- Execution Model
- Application Model

3 Multicore Response Time Analysis of SDF Programs

4 Evaluation

5 Conclusion and Future Work

Response Time Analysis

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

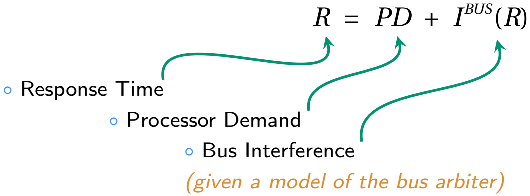
- Response Time

Response Time Analysis

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

- Response Time
  - Processor Demand

Response Time Analysis

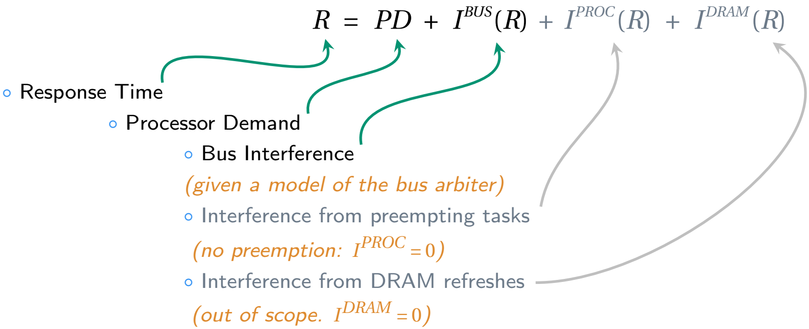


$$R = PD + I^{BUS}(R)$$

- Response Time
- Processor Demand
- Bus Interference

(given a model of the bus arbiter)

Response Time Analysis



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

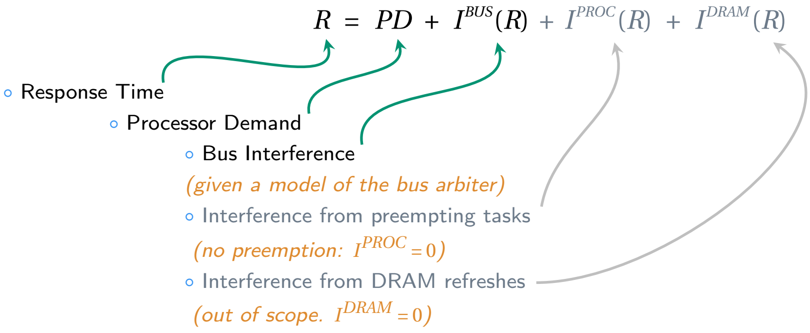
- Response Time
  - Processor Demand
    - Bus Interference

      (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

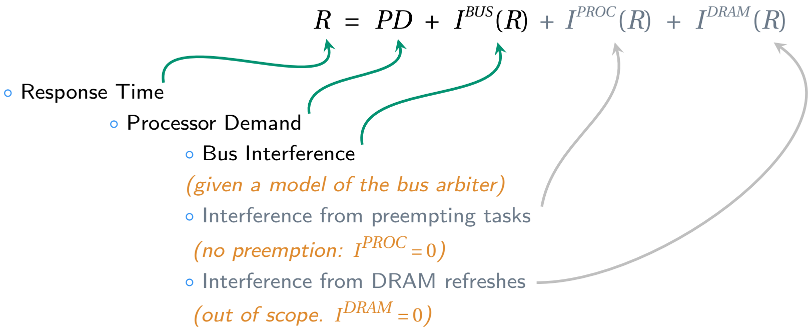


$$R = PD + I^{BUS}(R) + I^{PROC}(R) + I^{DRAM}(R)$$

- 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. $I^{DRAM} = 0$)

- Recursive formula ⇒ fixed-point algorithm.

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

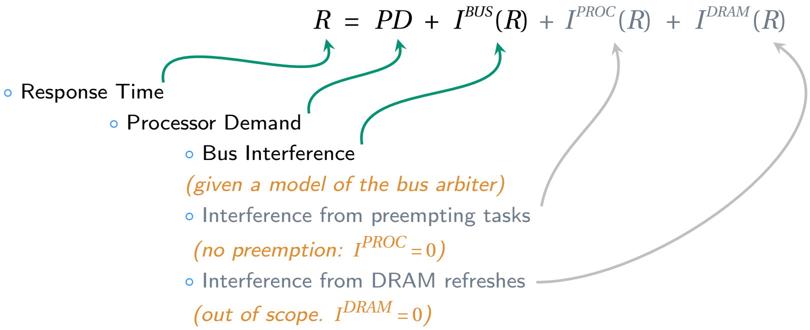
(no preemption: $I^{PROC} = 0$)

- Interference from DRAM refreshes

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

- Recursive formula ⇒ fixed-point 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. $I^{DRAM} = 0$)

- Recursive formula ⇒ fixed-point 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. $I^{DRAM} = 0$)

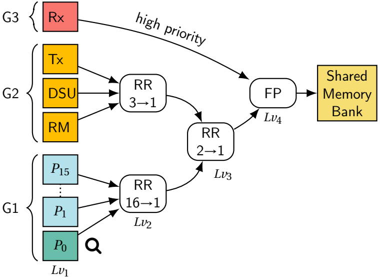
- Recursive formula ⇒ fixed-point algorithm.
- Multiple shared resources (memory banks)

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

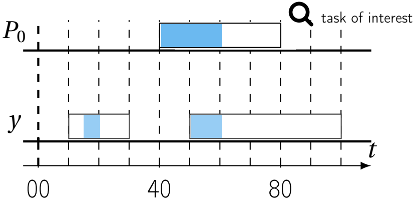
where B: a set of memory banks

🔍 Requires a model of the bus arbiter

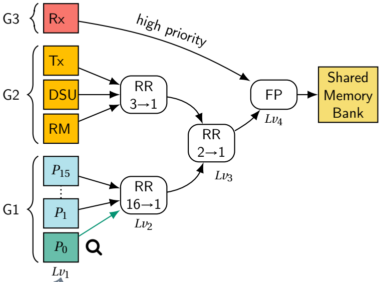
Model of the MPPA Bus



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



Model of the MPPA Bus

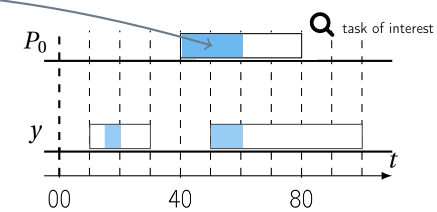


I_b^{BUS}: delay from all accesses + concurrent ones

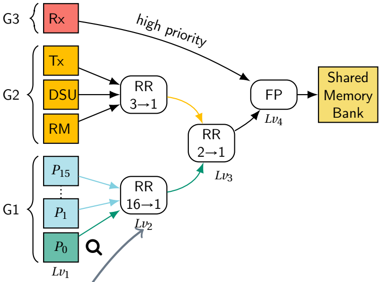
S_i^b: number of accesses of task τ_i to bank b

S_i^b = Memory Demand to bank b

$Lv_1 = S_i^b$



Model of the MPPA Bus



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

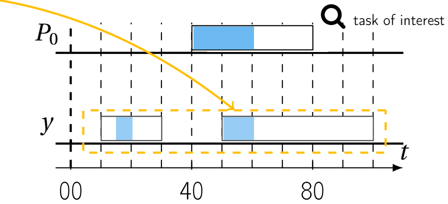
S_i^b: number of accesses of task τ_i to bank b

S_i^b = Memory Demand to bank b

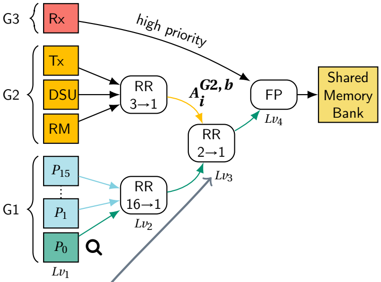
$A_i^{y,b}$: number of concurrent accesses from core y to bank b

$$Lv_1 = S_i^b$$

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



Model of the MPPA Bus



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

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

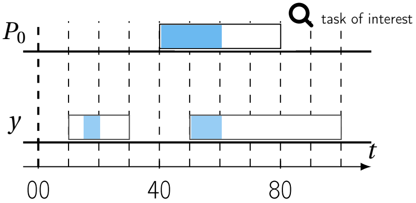
S_i^b = Memory Demand to bank b

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

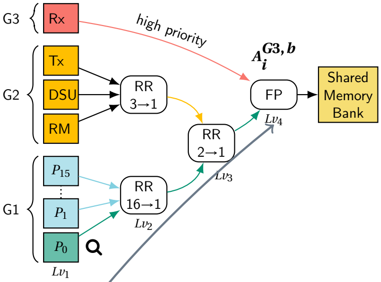
$$Lv_1 = S_i^b$$

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

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



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

S_i^b = Memory Demand to bank b

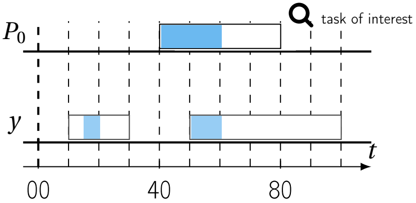
$A_i^{y,b}$: number of concurrent accesses from core y to bank b

$$Lv_1 = S_i^b$$

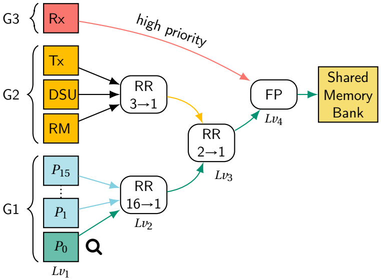
$$Lv_2 = Lv_1 + \sum_{y=1}^{15} \min(A_i^{y,b}, Lv_1)$$

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

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



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

S_i^b = Memory Demand to bank b

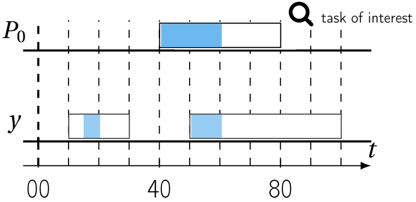
$A_i^{y,b}$: number of concurrent accesses from core y to bank b

$$Lv_1 = S_i^b$$

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

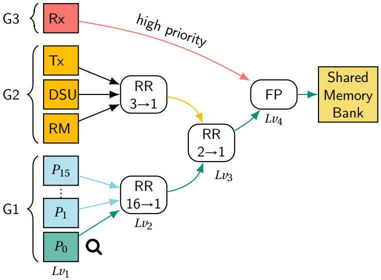
$$Lv_3 = Lv_2 + \min(A_i^{G2,b} , Lv_2)$$

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



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

Model of the MPPA Bus



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

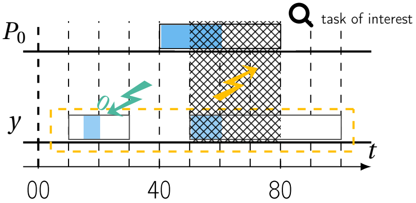
S_i^b: number of accesses of task τ_i to bank b

S_i^b = Memory Demand to bank b

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

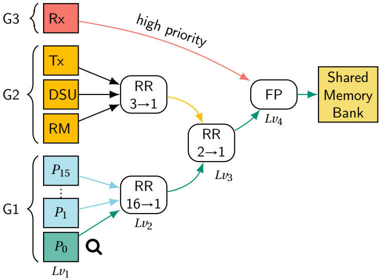
$A_i^{y,b} = \sum$ overlapping concurrent accesses

$$
\begin{aligned}
Lv_1 &= S_i^b \\
Lv_2 &= Lv_1 + \sum_{y=1}^{15} \min(A_i^{y,b} , Lv_1) \\
Lv_3 &= Lv_2 + \min(A_i^{G2,b} , Lv_2) \\
Lv_4 &= Lv_4 + A_i^{G3,b}
\end{aligned}
$$



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

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

S_i^b = Memory Demand to bank b

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

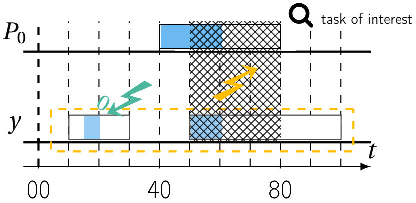
$A_i^{y,b} = \sum$ overlapping concurrent accesses

$$Lv_1 = S_i^b$$

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

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

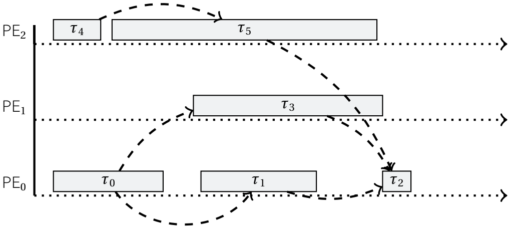
$$Lv_4 = Lv_4 + A_i^{G3,b}$$



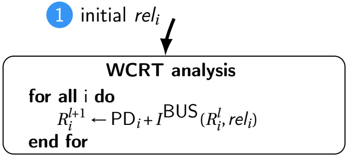
⚠ $A_i^{y,b}$ depends on rel_i and R_i

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

Response Time Analysis with Dependencies



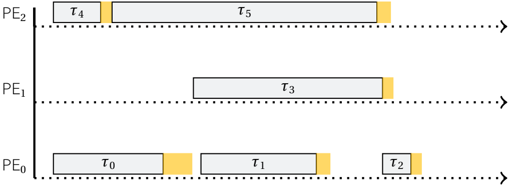
1 Start with initial release dates.

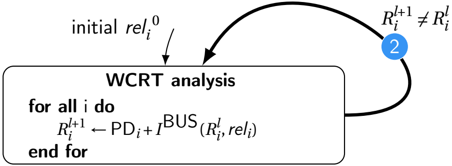


WCRT analysis

```
for all i do
    $R_i^{l+1} \leftarrow \mathrm{PD}_i + I^{\mathrm{BUS}}(R_i^l, rel_i)$
end for
```

Response Time Analysis with Dependencies





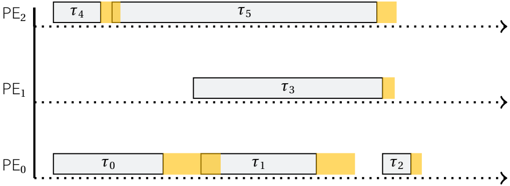
WCRT analysis

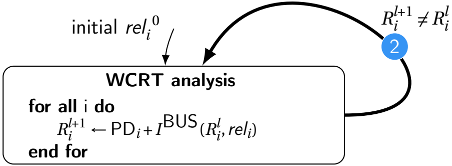
```
for all i do
    $R_i^{l+1} \leftarrow \mathrm{PD}_i + I^{\mathrm{BUS}}(R_i^l, rel_i)$
end for
```

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

   …

Response Time Analysis with Dependencies





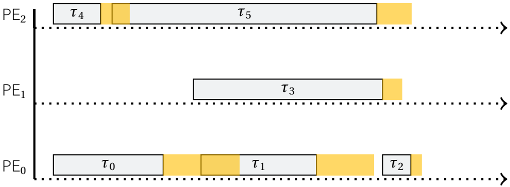
WCRT analysis

```
for all i do
    $R_i^{l+1} \leftarrow \mathrm{PD}_i + I^{\mathrm{BUS}}(R_i^l, rel_i)$
end for
```

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

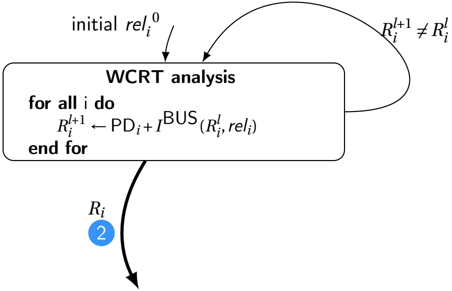
… …

Response Time Analysis with Dependencies



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

   … … … a fixed-point is reached!



initial rel_i^0

$R_i^{l+1} \neq R_i^l$

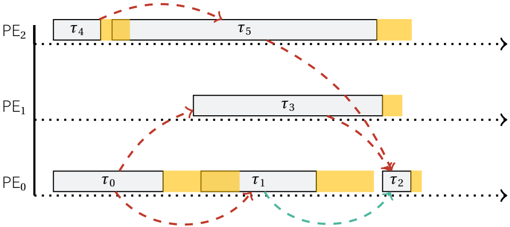
WCRT analysis

```
for all i do
    $R_i^{l+1} \leftarrow \mathrm{PD}_i + I^{\mathrm{BUS}}(R_i^l, rel_i)$
end for
```

R_i

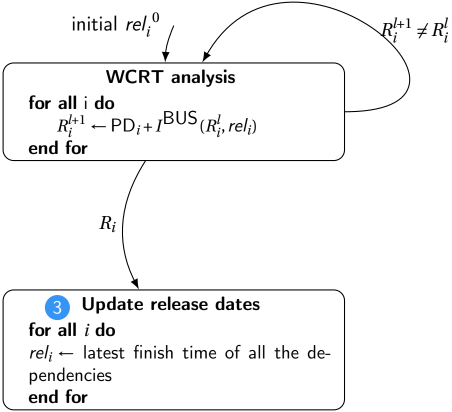
2

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.



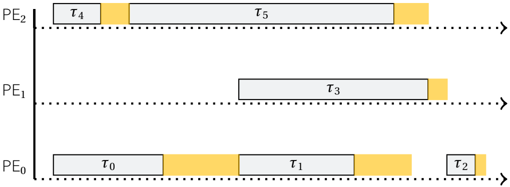
WCRT analysis

```
for all i do
    $R_i^{l+1} \leftarrow \mathrm{PD}_i + I^{\mathrm{BUS}}(R_i^l, rel_i)$
end for
```

3 **Update release dates**

```
for all i do
$rel_i \leftarrow$ latest finish time of all the dependencies
end for
```

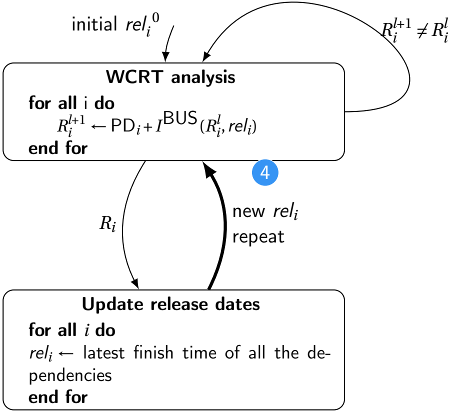
Response Time Analysis with Dependencies



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).



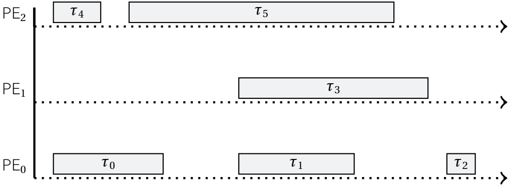
WCRT analysis

```
for all i do
    $R_i^{l+1} \leftarrow \mathrm{PD}_i + I^{\mathrm{BUS}}(R_i^l, rel_i)$
end for
```

Update release dates

```
for all $i$ do
$rel_i \leftarrow$ latest finish time of all the dependencies
end for
```

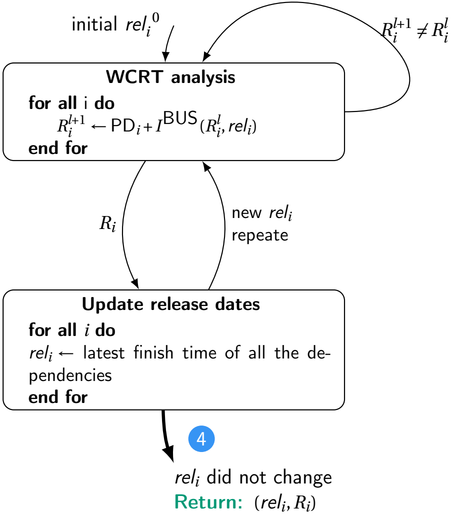
Response Time Analysis with Dependencies



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).



initial $rel_i{}^0$

$R_i^{l+1} \neq R_i^l$

WCRT analysis

```
for all i do
    $R_i^{l+1} \leftarrow \mathsf{PD}_i + I^{\mathsf{BUS}}(R_i^l, rel_i)$
end for
```

R_i

new rel_i repeate

Update release dates

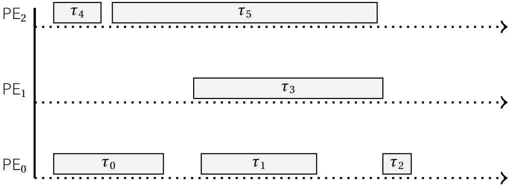
```
for all i do
$rel_i \leftarrow$ latest finish time of all the dependencies
end for
```

4

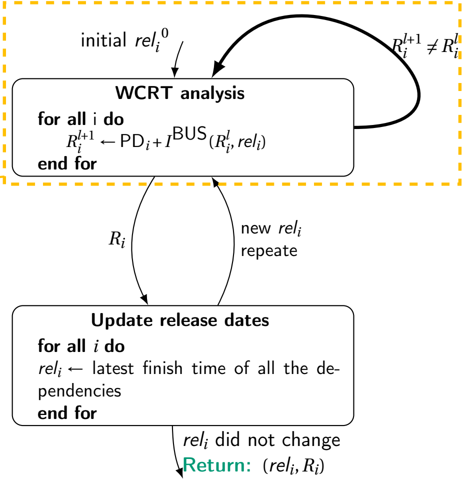
rel_i did not change

Return: (rel_i, R_i)

Convergence Toward a Fixed-point



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



initial $rel_i{}^0$

$R_i^{l+1} \neq R_i^l$

WCRT analysis

```
for all i do
    $R_i^{l+1} \leftarrow PD_i + I^{BUS}(R_i^l, rel_i)$
end for
```

R_i

new rel_i repeate

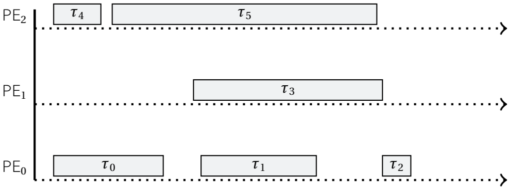
Update release dates

```
for all $i$ do
$rel_i \leftarrow$ latest finish time of all the dependencies
end for
```

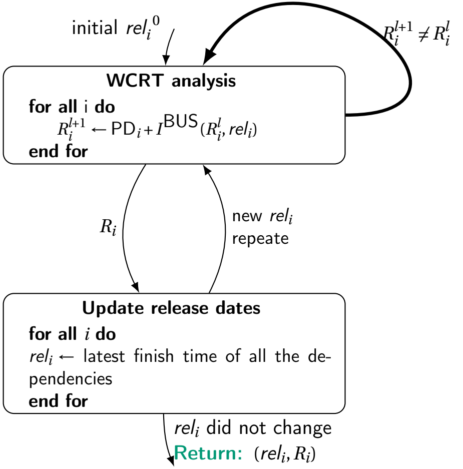
rel_i did not change

Return: (rel_i, R_i)

Convergence Toward a Fixed-point



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



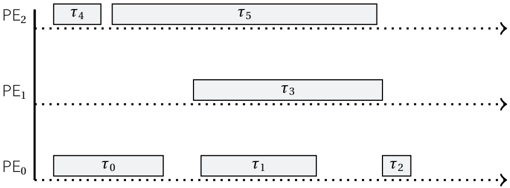
WCRT analysis

```
for all i do
    $R_i^{l+1} \leftarrow \mathrm{PD}_i + I^{\mathrm{BUS}}(R_i^l, rel_i)$
end for
```

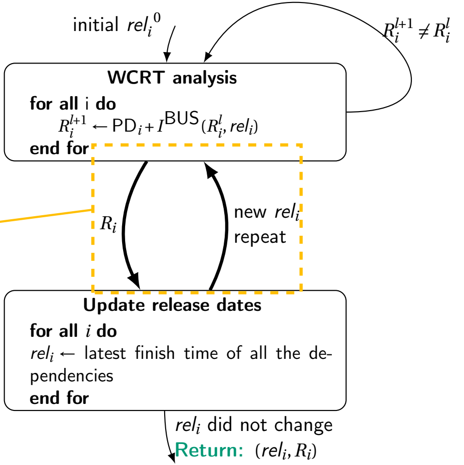
Update release dates

```
for all $i$ do
$rel_i \leftarrow$ latest finish time of all the dependencies
end for
```

Convergence Toward a Fixed-point



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



initial rel_i^0

$R_i^{l+1} \neq R_i^l$

WCRT analysis

```
for all i do
    $R_i^{l+1} \leftarrow \mathrm{PD}_i + I^{\mathrm{BUS}}(R_i^l, rel_i)$
end for
```

R_i

new rel_i repeat

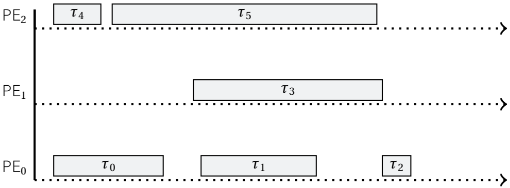
Update release dates

```
for all $i$ do
$rel_i \leftarrow$ latest finish time of all the dependencies
end for
```

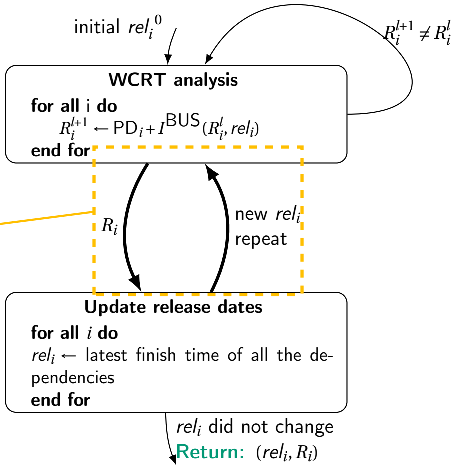
rel_i did not change

Return: (rel_i, R_i)

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. ?



initial rel_i^0

$R_i^{l+1} \neq R_i^l$

WCRT analysis

```
for all i do
    $R_i^{l+1} \leftarrow \mathrm{PD}_i + I^{\mathrm{BUS}}(R_i^l, rel_i)$
end for
```

R_i

new rel_i repeat

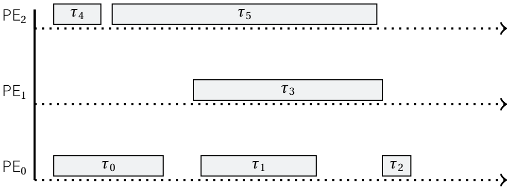
Update release dates

```
for all $i$ do
$rel_i \leftarrow$ latest finish time of all the dependencies
end for
```

rel_i did not change

Return: (rel_i, R_i)

Convergence Toward a Fixed-point

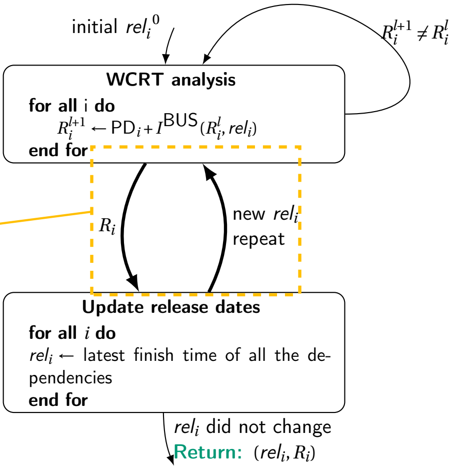


- Convergence of the 1^{st} fixed-point iteration:
  - Monotonic and bounded ✓
- Convergence of the 2^{nd} fixed-point iteration:
  - no monotonicity: R_i and *rel*$_i$ may grow or shrink at each iteration. ?

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



initial rel_i^0

$R_i^{l+1} \neq R_i^l$

WCRT analysis

```
for all i do
    $R_i^{l+1} \leftarrow \mathrm{PD}_i + I^{\mathrm{BUS}}(R_i^l, rel_i)$
end for
```

R_i

new *rel*$_i$ repeat

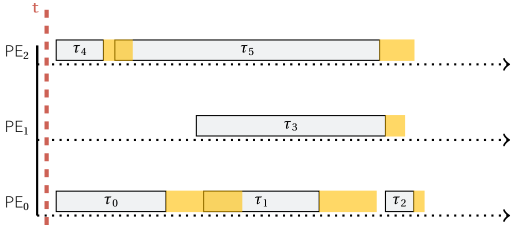
Update release dates

```
for all i do
    $rel_i \leftarrow$ latest finish time of all the dependencies
end for
```

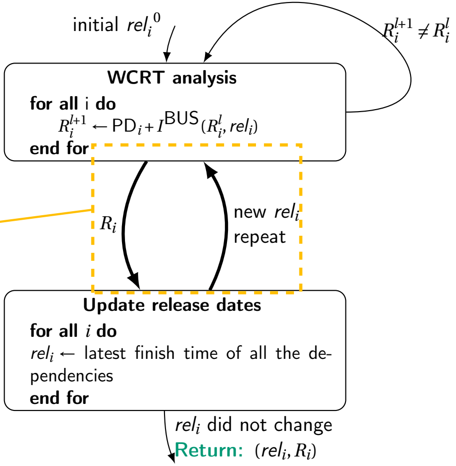
rel$_i$ did not change

Return: (rel_i, R_i)

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. ?



WCRT analysis

```
for all i do
    $R_i^{l+1} \leftarrow \mathrm{PD}_i + I^{\mathrm{BUS}}(R_i^l, rel_i)$
end for
```

Update release dates

```
for all $i$ do
$rel_i \leftarrow$ latest finish time of all the dependencies
end for
```

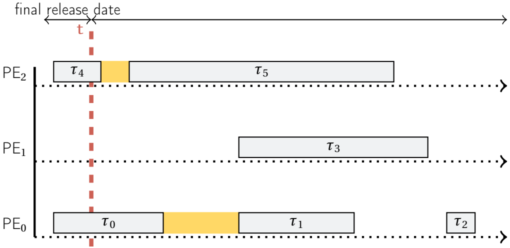
Return: (rel_i, R_i)

Theorem

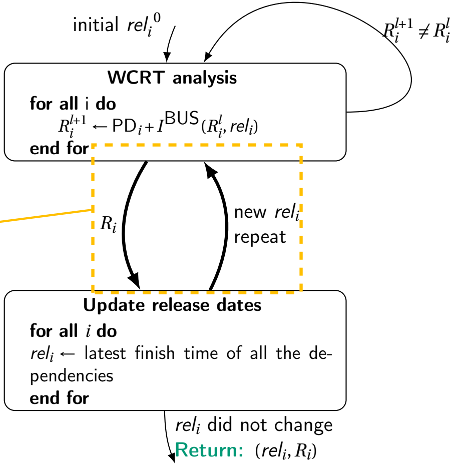
At each iteration, at least one task finds its final release date.

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

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. ?



initial rel_i^0

$R_i^{l+1} \neq R_i^l$

WCRT analysis

```
for all i do
    $R_i^{l+1} \leftarrow \mathrm{PD}_i + I^{\mathrm{BUS}}(R_i^l, rel_i)$
end for
```

R_i

new rel_i repeat

Update release dates

```
for all $i$ do
$rel_i \leftarrow$ latest finish time of all the dependencies
end for
```

rel_i did not change

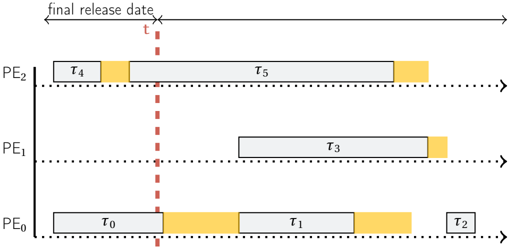
Return: (rel_i, R_i)

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

Convergence Toward a Fixed-point



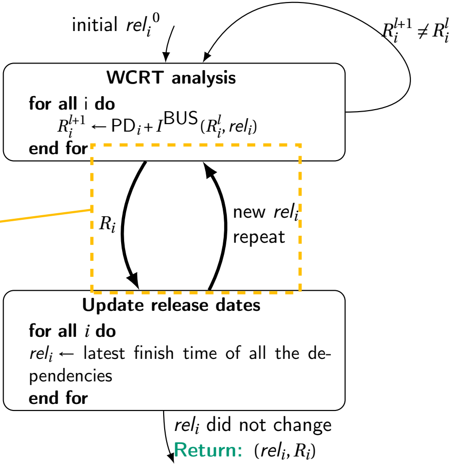
- Convergence of the 1^{st} fixed-point iteration:
  - Monotonic and bounded ✓
- Convergence of the 2^{nd} fixed-point iteration:
  - no monotonicity: R_i and *rel*$_i$ may grow or shrink at each iteration. ?

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



WCRT analysis

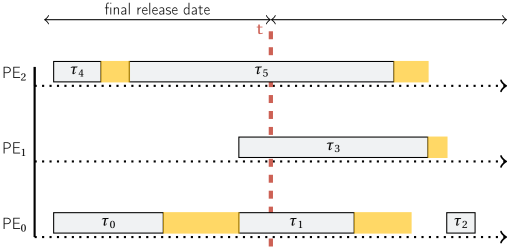
```
for all i do
    $R_i^{l+1} \leftarrow \mathrm{PD}_i + I^{\mathrm{BUS}}(R_i^l, rel_i)$
end for
```

Update release dates

```
for all $i$ do
$rel_i \leftarrow$ latest finish time of all the dependencies
end for
```

Return: (rel_i, R_i)

Convergence Toward a Fixed-point



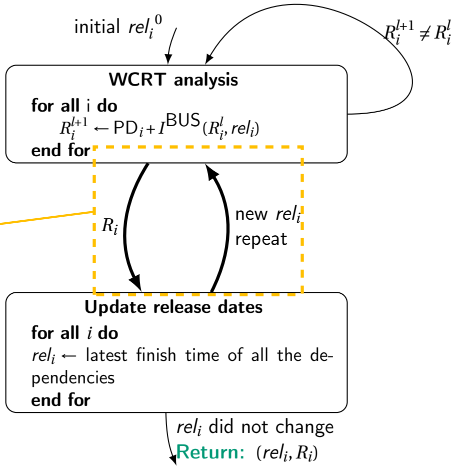
- Convergence of the 1^{st} fixed-point iteration:
  - Monotonic and bounded ✓
- Convergence of the 2^{nd} fixed-point iteration:
  - no monotonicity: R_i and rel_i may grow or shrink at each iteration. ?

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



WCRT analysis

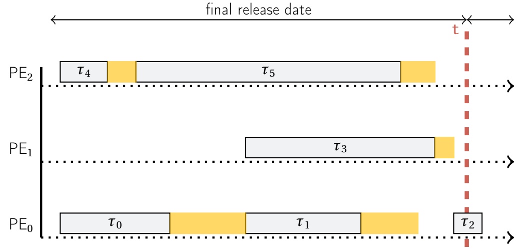
```
for all i do
    $R_i^{l+1} \leftarrow \mathrm{PD}_i + I^{\mathrm{BUS}}(R_i^l, rel_i)$
end for
```

Update release dates

```
for all $i$ do
    $rel_i \leftarrow$ latest finish time of all the dependencies
end for
```

Return: (rel_i, R_i)

Convergence Toward a Fixed-point



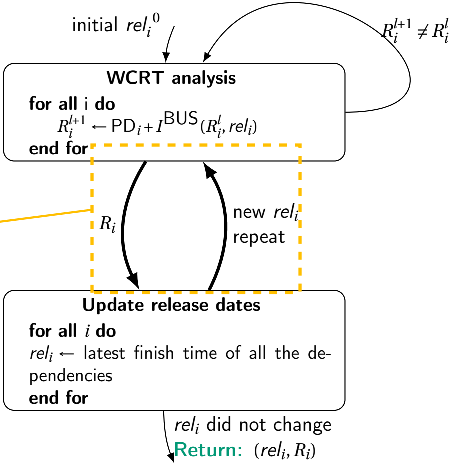
- Convergence of the 1^{st} fixed-point iteration:
  - Monotonic and bounded ✓
- Convergence of the 2^{nd} fixed-point iteration:
  - no monotonicity: R_i and rel_i may grow or shrink at each iteration. ?

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



initial rel_i^0

$R_i^{l+1} \neq R_i^l$

WCRT analysis

```
for all i do
    $R_i^{l+1} \leftarrow \mathrm{PD}_i + I^{\mathrm{BUS}}(R_i^l, rel_i)$
end for
```

R_i

new rel_i repeat

Update release dates

```
for all i do
$rel_i \leftarrow$ latest finish time of all the dependencies
end for
```

rel_i did not change

Return: (rel_i, R_i)

Outline

1 Motivation and Context

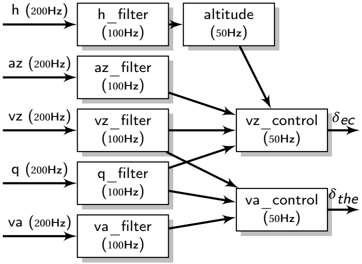
2 Models Definition
- Architecture Model
- Execution Model
- Application Model

3 Multicore Response Time Analysis of SDF Programs

4 Evaluation

5 Conclusion and Future Work

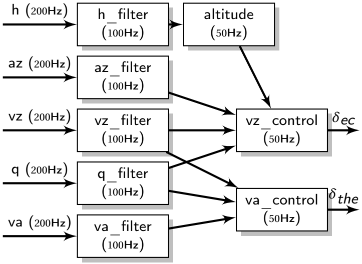
Evaluation: ROSACE Case Study [1]



- Flight management system controller

[1] Pagetti et al., RTAS 2014

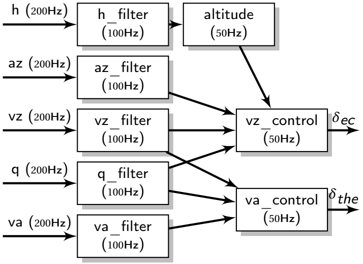
Evaluation: ROSACE Case Study [1]

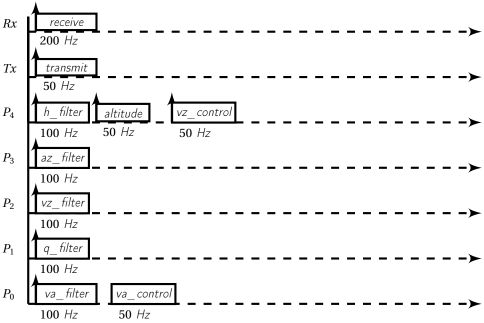


- Flight management system controller
- Receive from sensors and transmit to actuators

[1] Pagetti et al., RTAS 2014

Evaluation: ROSACE Case Study [1]



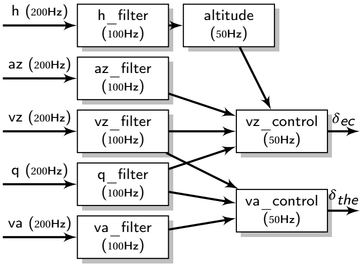


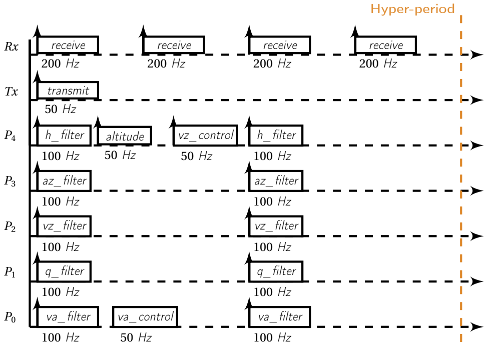
- Flight management system controller
- Receive from sensors and transmit to actuators
- **Assumptions:**

  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

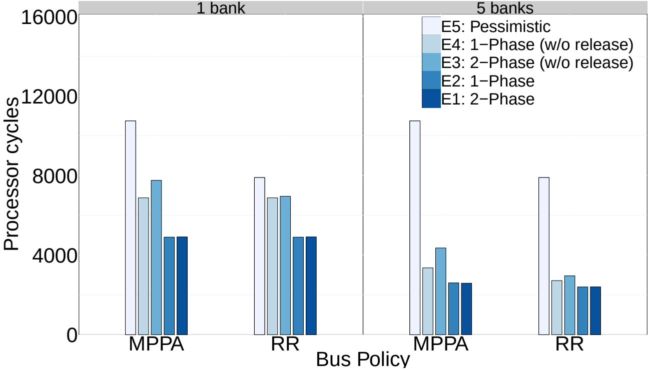
Task	Processor Demand (cycles)	Memory Demand (accesses)
altitude	275	22
az_filter	274	22
h_filter	326	24
va_control	303	24
va_filter	301	23
vz_control	320	25
vz_filter	334	25

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

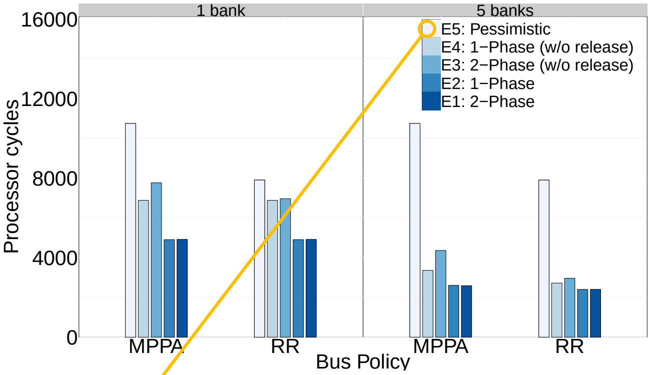
Experiments: Find the smallest schedulable hyper-period

Evaluation: Experiments



Smallest schedulable hyper-period

Evaluation: Experiments

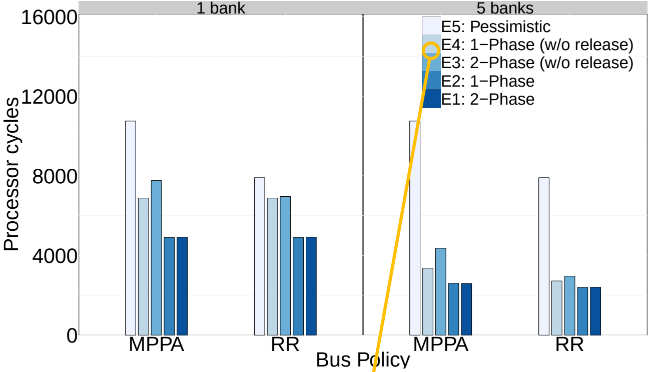


Smallest schedulable hyper-period

- Pessimistic assumption: High priority tasks are bounded by 1 access per bank

E5: All accesses interfere

Evaluation: Experiments



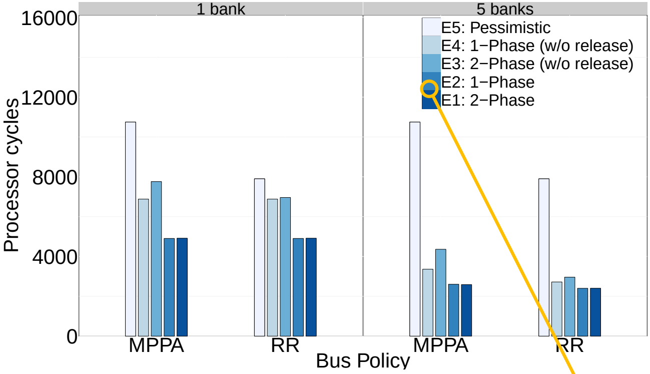
- Pessimistic assumption: High priority tasks are bounded by 1 access per bank

Smallest schedulable hyper-period

E5: All accesses interfere

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

Evaluation: Experiments



Smallest schedulable hyper-period

- Pessimistic assumption: High priority tasks are bounded by 1 access per bank

E5: All accesses interfere

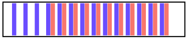
E4, E3: We don't use the release dates

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

Evaluation: Experiments

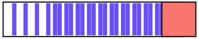
1-Phase model

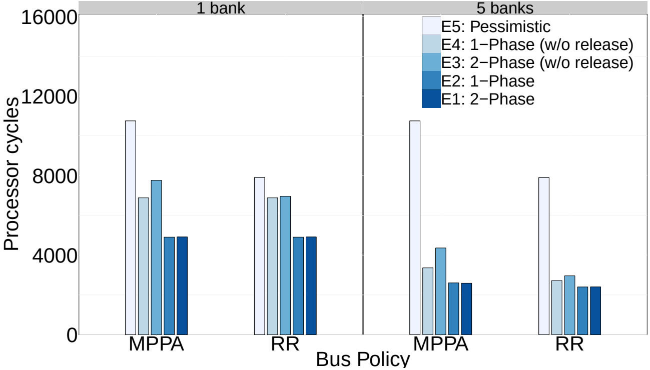
memory access pattern



2-Phase model

memory access pattern





- Pessimistic assumption: High priority tasks are bounded by 1 access per bank
- Phases are modeled as sub-tasks

Smallest schedulable hyper-period

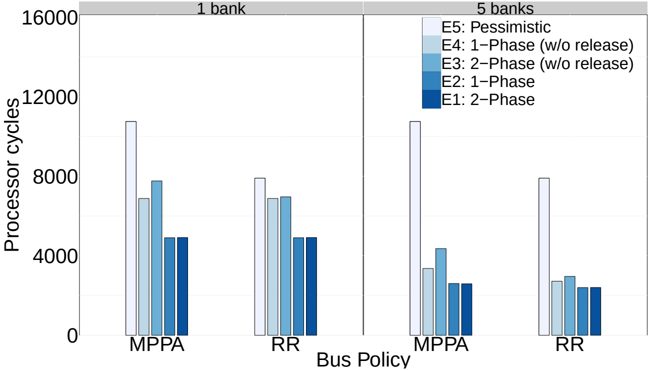
E5: All accesses interfere

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 $[1.77, 2.52]$



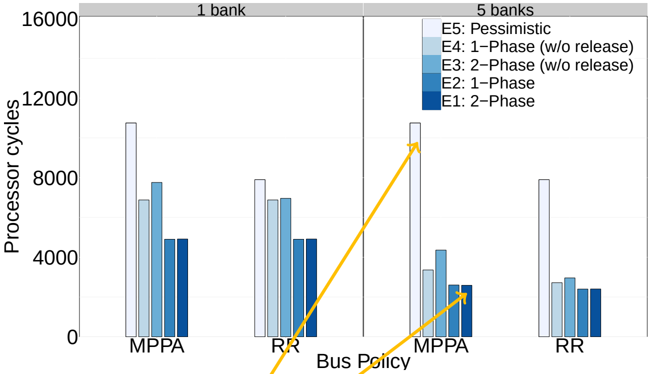
Smallest schedulable hyper-period

	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 $[1.77, 2.52]$



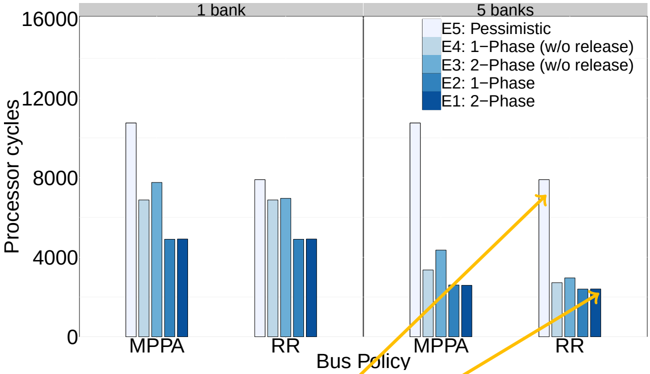
Smallest schedulable hyper-period

	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 $[1.77, 2.52]$



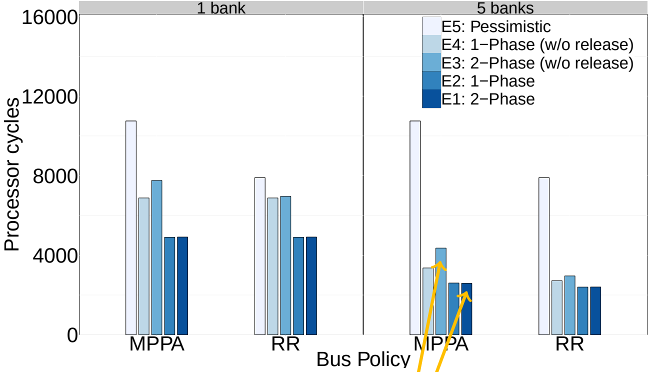
Smallest schedulable hyper-period

	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 $[1.77, 2.52]$



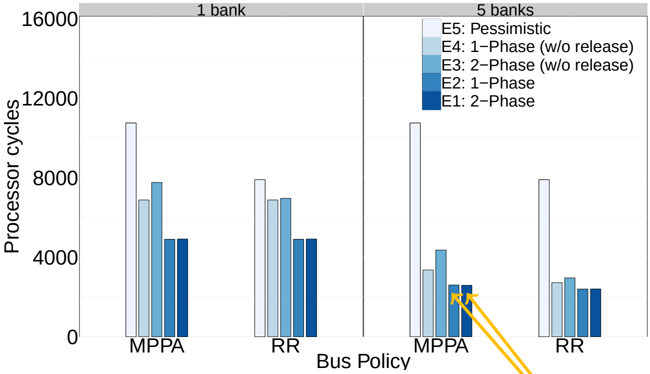
Smallest schedulable hyper-period

	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 $[1.77, 2.52]$



Smallest schedulable hyper-period

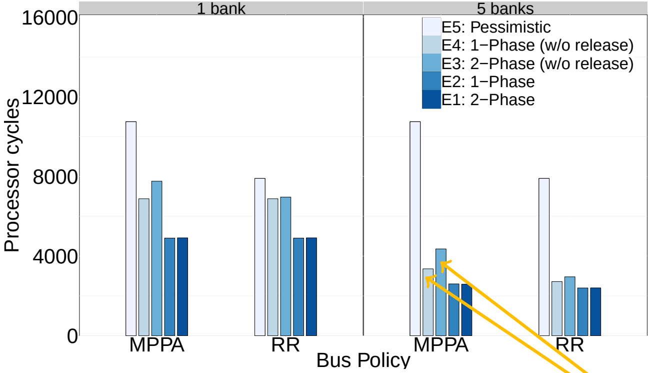
RTNS * Artifact * Evaluated
Consistent * Complete * Well documented * Easy to reuse *

	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 $[1.77, 2.52]$



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

Outline

1 Motivation and Context

2 Models Definition
- Architecture Model
- Execution Model
- Application Model

3 Multicore Response Time Analysis of SDF Programs

4 Evaluation

5 Conclusion and Future Work

Conclusion

- A response time analysis of SDF on the Kalray MPPA 256

Conclusion

- A response time analysis of SDF on the Kalray MPPA 256
- Given:
  - Task profile
  - Mapping of Tasks
  - Execution Order

Conclusion

- A response time analysis of SDF on the Kalray MPPA 256
- Given:
  - Task profile
  - Mapping of Tasks
  - Execution Order
- We compute:
  - Tight response times taking into account the interference.
  - Release dates respecting the dependency constraints.

Conclusion

- A response time analysis of SDF on the Kalray MPPA 256
- Given:
  - Task profile
  - Mapping of Tasks
  - Execution Order
- We compute:
  - Tight response times taking into account the interference.
  - Release dates respecting the dependency constraints.

model of
the multi-level arbiter

Conclusion

- A response time analysis of SDF on the Kalray MPPA 256
- Given:
  - Task profile
  - Mapping of Tasks
  - Execution Order
- We compute:
  - Tight response times taking into account the interference.
  - Release dates respecting the dependency constraints.

model of
the multi-level arbiter

double fixed-point
algorithm

Conclusion

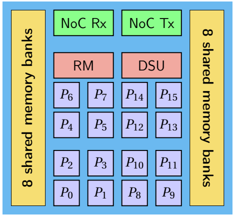
- A response time analysis of SDF on the Kalray MPPA 256
- Given:
  - Task profile
  - Mapping of Tasks
  - Execution Order
- We compute:
  - Tight response times taking into account the interference.
  - Release dates respecting the dependency constraints.
- Not restricted to SDF

model of
the multi-level arbiter

double fixed-point
algorithm

Future Work

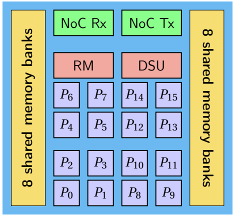
- Model of the Resource Manager.



Future Work

- Model of the Resource Manager.

tighter estimation of context switches and other interrupts

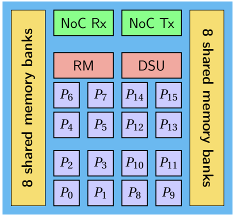


Future Work

- Model of the Resource Manager.

tighter estimation of context switches and other interrupts

- Model of the NoC accesses.



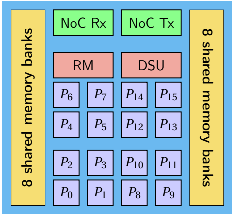
Future Work

- 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

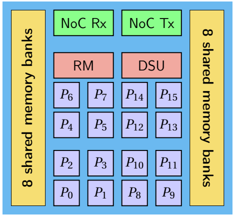


Future Work

- 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



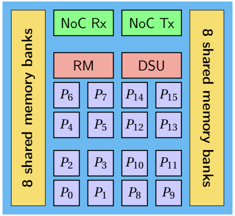
Future Work

- 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

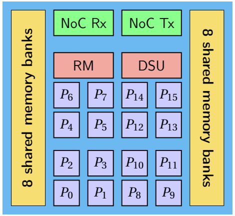
- Model of the NoC accesses.

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.

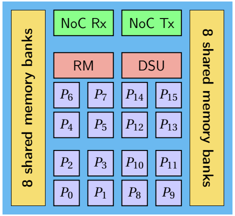
use the output of any NoC analysis

- 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.

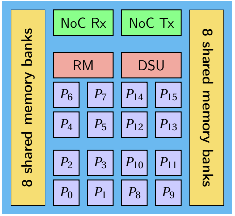
use the output of any NoC analysis

- Memory access pipelining.

current assumption: bus delay is 10 cycles

- Model Blocking and non-blocking accesses.

reads are blocking
writes are non-blocking

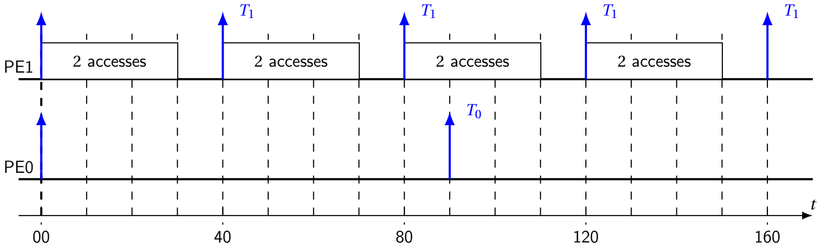


Questions?

BACKUP

Multicore Response Time Analysis

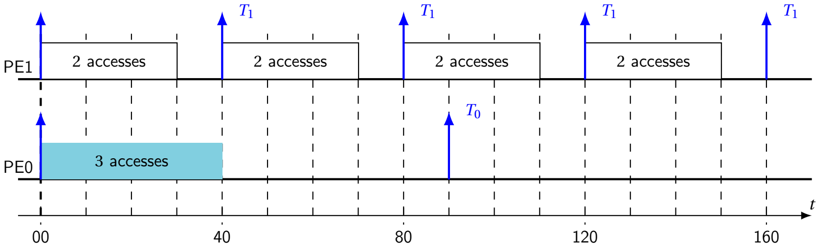
Example: Fixed Priority bus arbiter, PE1 > PE0
Bus access delay = 10



[1] Altmeyer et al., RTNS 2015

Multicore Response Time Analysis

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



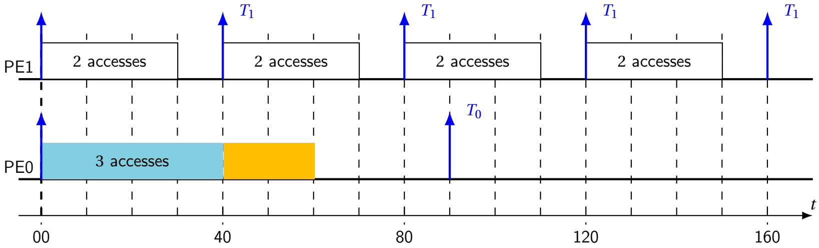
- Task of interest running on $PE0$:

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

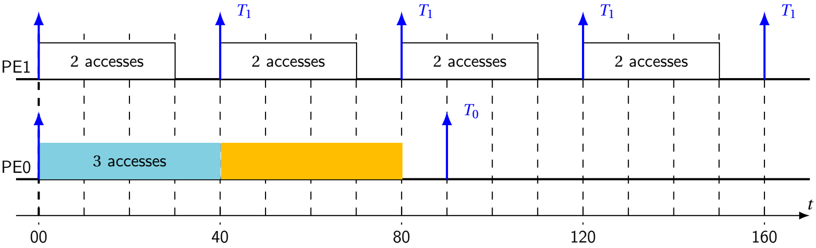
  $R_0 = 10 + 3 \times 10$ (response time in isolation)

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

[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)

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

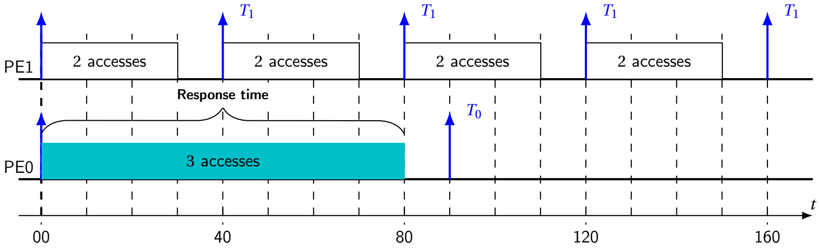
  $R_2 = 10 + 3 \times 10 + 2 \times 10 + 2 \times 10 = 80$

[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)

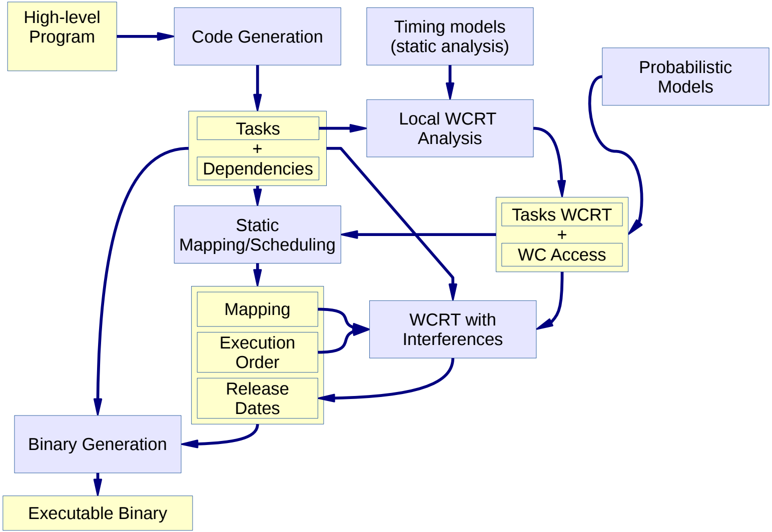
  $R_1 = 10 + 3 \times 10 + 2 \times 10 = 60$

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

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

[1] Altmeyer et al., RTNS 2015

The Global Picture


High-level Program
Code Generation
Timing models (static analysis)
Probabilistic Models
Tasks
+
Dependencies
Local WCRT Analysis
Static Mapping/Scheduling
Tasks WCRT
+
WC Access
Mapping
Execution Order
Release Dates
WCRT with Interferences
Binary Generation
Executable Binary