Many-Core Timing Analysis of Real-Time Systems

and its application to an industrial processor

Hamza Rihani

Université Grenoble Alpes / Verimag

December 1, 2017

Jury:

Pr. Jan Reineke	Saarland University	*Reviewer*
Pr. Christine Rochange	Université de Toulouse	*Reviewer*
Dr. Robert I. Davis	University of York	*Examiner*
Dr. Benoît de Dinechin	Kalray SA	*Examiner*
Dr. Claire Maïza	Université Grenoble Alpes	*Supervisor*
Dr. Matthieu Moy	Université Claude Bernard - Lyon 1	*Advisor*

Introduction: Real-Time Systems

Many-Core Timing Analysis of **Real-Time Systems**

Definition (Real-Time Systems)

A system that must produce **valid** outputs before a **deadline**.

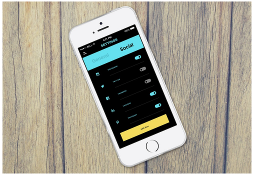
Introduction: Real-Time Systems

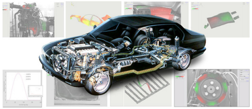
Many-Core Timing Analysis of **Real-Time Systems**

Definition (Real-Time Systems)

A system that must produce **valid** outputs before a **deadline**.

- **Soft Real-Time**
  - Global Positioning System device
  - Smartphones
- **Hard Real-Time**
  - Automatic Braking System
  - Flight Management System





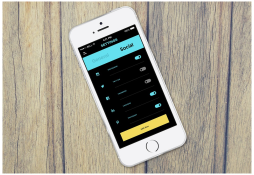
Introduction: Real-Time Systems

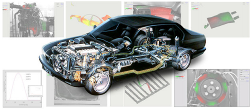
Many-Core Timing Analysis of **Real-Time Systems**

Definition (Real-Time Systems)

A system that must produce **valid** outputs before a **deadline**.

- **Soft Real-Time**
  - Global Positioning System device
  - Smartphones
- **Hard Real-Time** ★
  - Automatic Braking System
  - Flight Management System

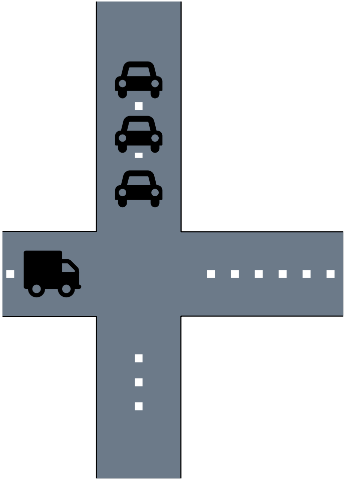




Introduction: Timing Analysis of Arbitration Policies

Many-Core **Timing Analysis** of Real-Time Systems

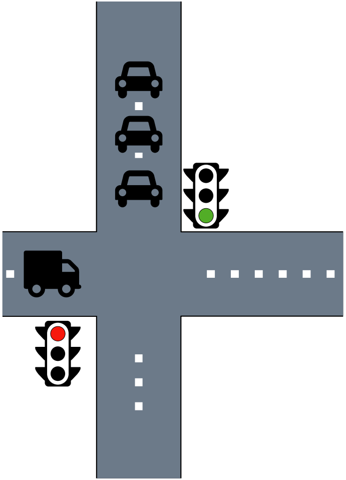
How long will the truck wait to cross the road?



Introduction: Timing Analysis of Arbitration Policies

Many-Core **Timing Analysis** of Real-Time Systems

How long will the truck wait to cross the road?

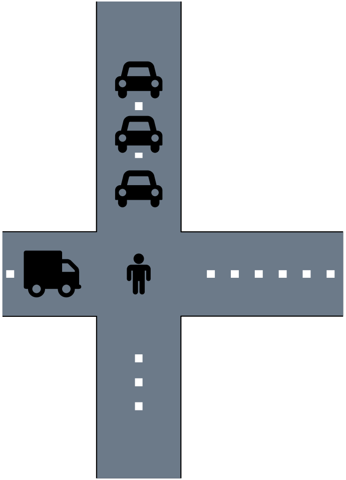


🚦 waits for the green light

Introduction: Timing Analysis of Arbitration Policies

Many-Core **Timing Analysis** of Real-Time Systems

How long will the truck wait to cross the road?



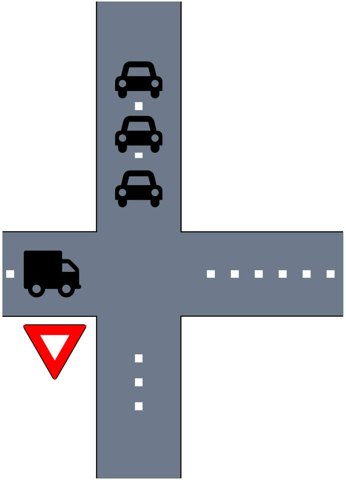
🚦 waits for the green light

🧍 grants each direction at a time

Introduction: Timing Analysis of Arbitration Policies

Many-Core **Timing Analysis** of Real-Time Systems

How long will the truck wait to cross the road?



🚦 waits for the green light

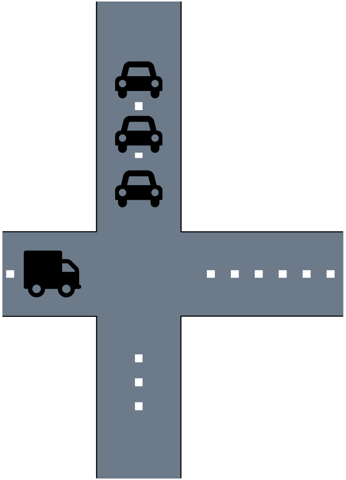
🧍 grants each direction at a time

▽ gives a priority to the cars

Introduction: Timing Analysis of Arbitration Policies

Many-Core **Timing Analysis** of Real-Time Systems

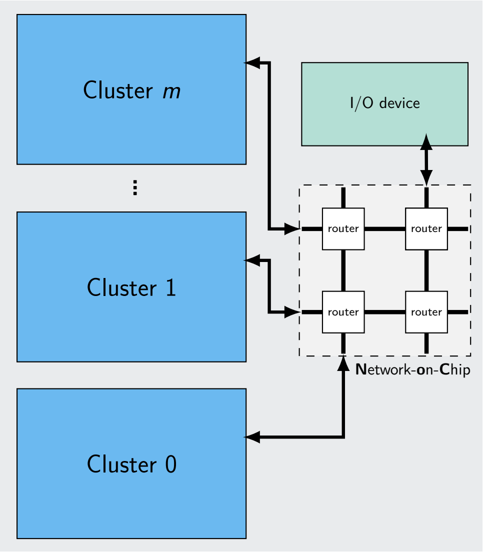
How long will the truck wait to cross the road?



- Crossroad is a shared resource
- Vehicles request accesses to pass
- **Arbitration Policies:**
  - 🚦 *Time Division Multiple Access*
  - 🚶 *Round Robin*
  - ▽ *Fixed Priority*

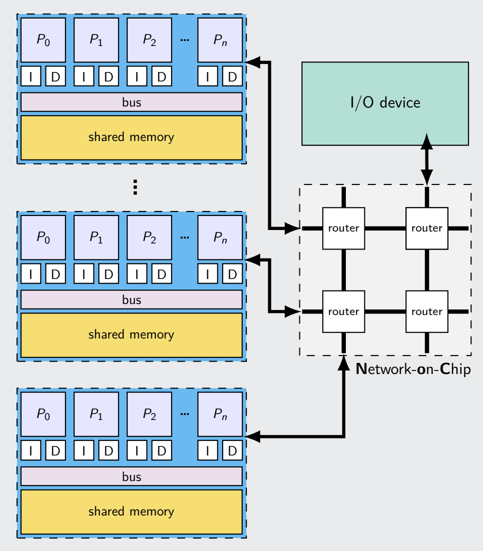
Introduction: Many-Cores in Real Time Systems

Many-Core Timing Analysis of Real-Time Systems



Introduction: Many-Cores in Real Time Systems

Many-Core Timing Analysis of Real-Time Systems

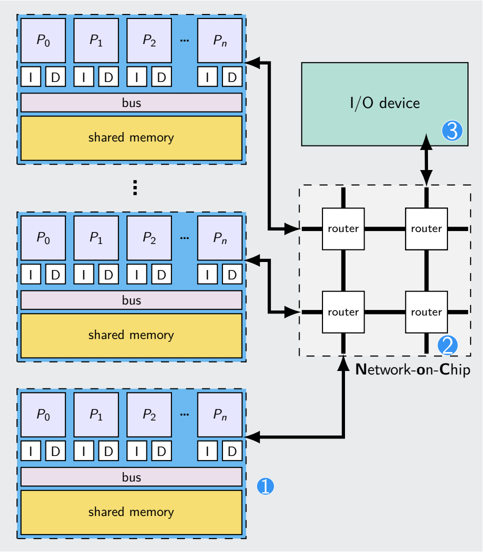


Timing analysis of cores

- Existing tools for pipeline and cache analyses

Introduction: Many-Cores in Real Time Systems

Many-Core Timing Analysis of Real-Time Systems



Timing analysis of cores

- Existing tools for pipeline and cache analyses

Where is the potential interference?

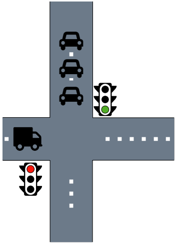
1. Shared buses and memory★
2. NoC routing
3. Shared I/O controllers

Contributions

Contribution 1

Analysis of Time Division Multiple Access policy

- Approach based on Satisfiability Modulo Theory

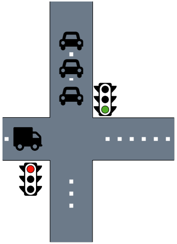


Contributions

Contribution 1

Analysis of Time Division Multiple Access policy

- Approach based on Satisfiability Modulo Theory



Contribution 2

Response time analysis of a many-core processor

- Synchronous Data Flow programs
- Model of the shared bus arbiter

The High Five, Dallas, Texas, USA

Outline

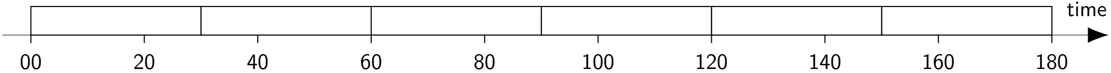
I TDMA Bus Timing Analysis

II Many-Core Response Time Analysis

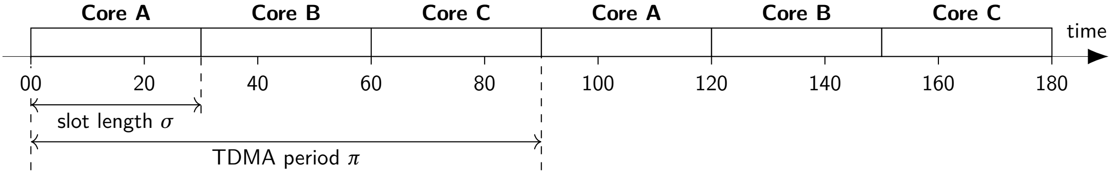
III Conclusion

TDMA Bus Timing Analysis

Definition: Time Division Multiple Access

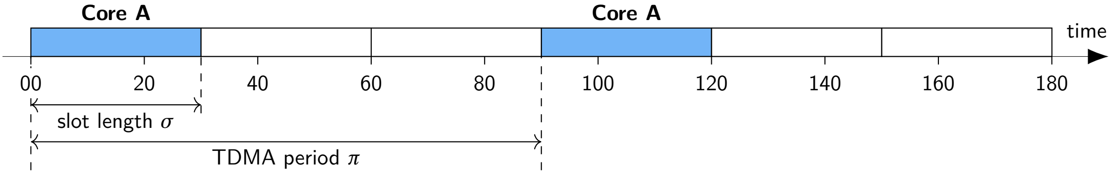


Definition: Time Division Multiple Access



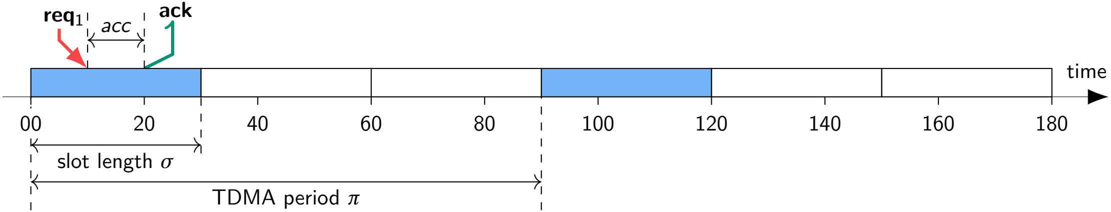
Definition: Time Division Multiple Access

🚦 Core A viewpoint:



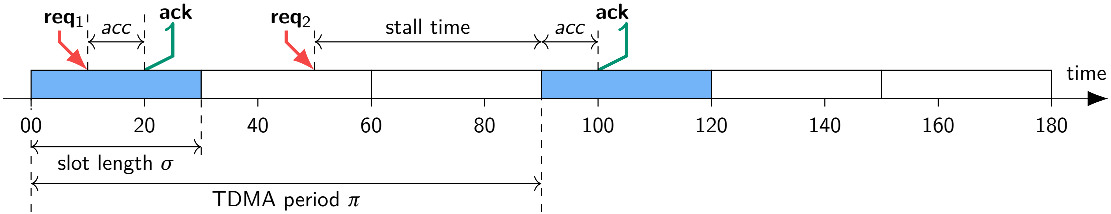
Definition: Time Division Multiple Access

🚦 Core A viewpoint:



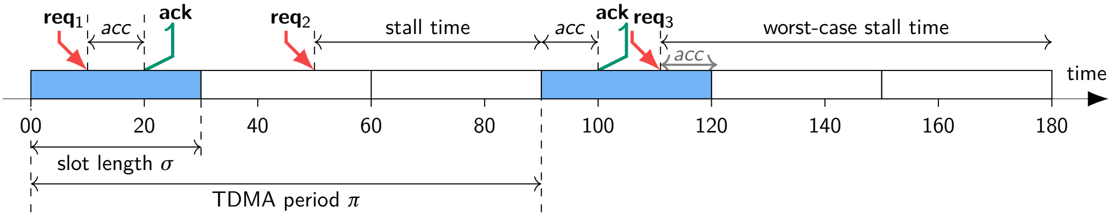
Definition: Time Division Multiple Access

🚦 Core A viewpoint:



Definition: Time Division Multiple Access

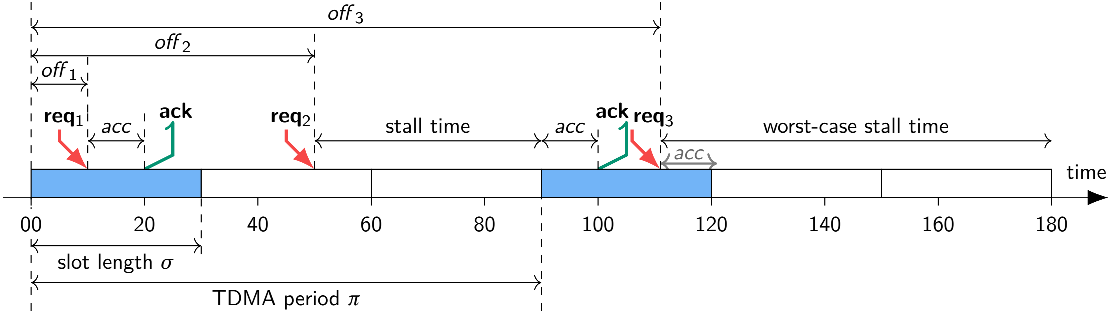
🚦 Core A viewpoint:



Worst-Case Stall Time $= \pi$ - (σ - *acc*)

Definition: Time Division Multiple Access

🚦 Core A viewpoint:



$$\textit{Worst-Case Stall Time} = \pi \text{ - } (\sigma \text{ - } acc)$$

- **Offsets** off_1, off_2, off_3 relative to the TDMA period:

$$off_{\{1,2,3\}} = time\_instant(req_{\{1,2,3\}}) \mod \pi$$

Outline: TDMA Bus Timing Analysis

1 Approaches in WCET Analysis of TDMA

2 WCET Analysis by SMT Encoding
- Naive SMT Approach
- Offset-based SMT Encoding

3 Experimental Evaluation

4 Summary and Future Work of Part I

Outline: TDMA Bus Timing Analysis

1. Approaches in WCET Analysis of TDMA

2. WCET Analysis by SMT Encoding
   - Naive SMT Approach
   - Offset-based SMT Encoding

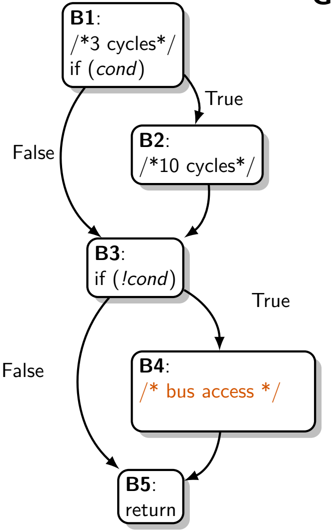
3. Experimental Evaluation

4. Summary and Future Work of Part I

Worst-Case Execution Time (WCET) Analysis of TDMA

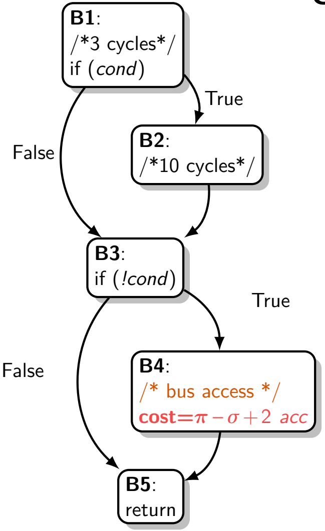
Goal: Estimate the WCET

```
int f(int x){
  /* 3 cycles */
  if(cond)
  {
    /* 10 cycles */
  }
  if(!cond)
  {
    /*bus access */
  }
  return;
}
```



Worst-Case Execution Time (WCET) Analysis of TDMA

```
int f(int x){
  /* 3 cycles */
  if(cond)
  {
    /* 10 cycles */
  }
  if(!cond)
  {
    /*bus access */
  }
  return;
}
```



Goal: Estimate the WCET

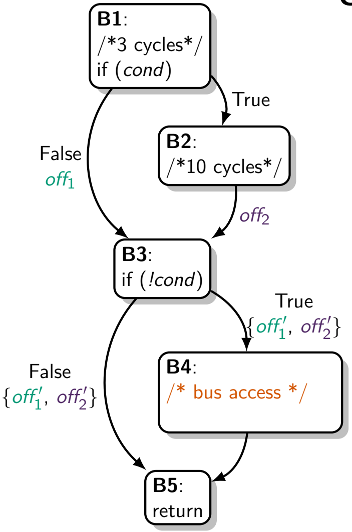
↪ **Existing approaches:**

❶ Worst-case everywhere

[Altmeyer et al., 2015; Rosèn et al., 2007...]

Worst-Case Execution Time (WCET) Analysis of TDMA

```
int f(int x){
  /* 3 cycles */
  if(cond)
  {
    /* 10 cycles */
  }
  if(!cond)
  {
    /*bus access */
  }
  return;
}
```



Goal: Estimate the WCET

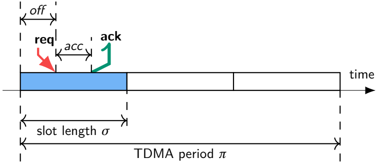
↪ **Existing approaches:**

❶ Worst-case everywhere

[Altmeyer et al., 2015; Rosèn et al., 2007...]

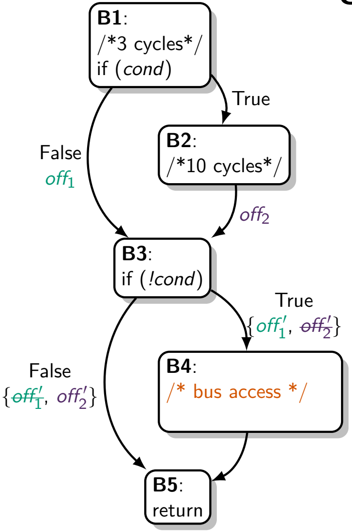
❷ Capture all possible offsets

[Chattopadhyay et al., 2010; Kelter et al., 2014...]



Worst-Case Execution Time (WCET) Analysis of TDMA

```
int f(int x){
  /* 3 cycles */
  if(cond)
  {
    /* 10 cycles */
  }
  if(!cond)
  {
    /*bus access */
  }
  return;
}
```



B1: /*3 cycles*/ if (*cond*)

True

B2: /*10 cycles*/

False off_1

off_2

B3: if (*!cond*)

True $\{off'_1, \cancel{off'_2}\}$

B4: /* bus access */

False $\{\cancel{off'_1}, off'_2\}$

B5: return

Goal: Estimate the WCET

↪ **Existing approaches:**

❶ Worst-case everywhere

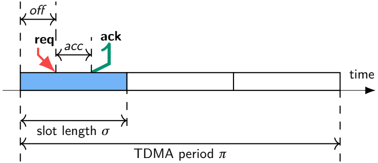
[Altmeyer et al., 2015; Rosèn et al., 2007...]

❷ Capture all possible offsets

[Chattopadhyay et al., 2010; Kelter et al., 2014...]

↪ **Combined with:**

❸ Feasible Path Analysis



Approaches in WCET Analysis of TDMA

❷

Capture all possible offsets

[Kelter et al., 2014]

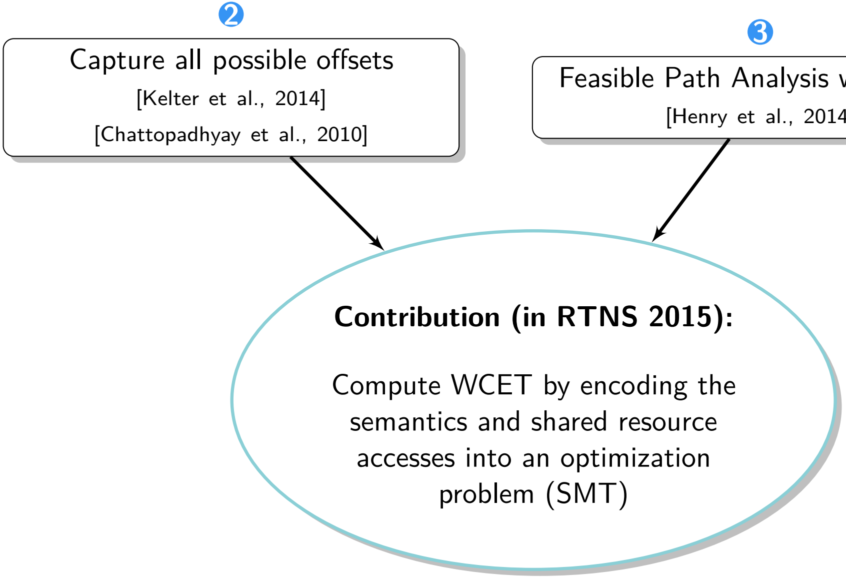
[Chattopadhyay et al., 2010]

❸

Feasible Path Analysis with SMT

[Henry et al., 2014]

Approaches in WCET Analysis of TDMA



❷

Capture all possible offsets

[Kelter et al., 2014]

[Chattopadhyay et al., 2010]

❸

Feasible Path Analysis with SMT

[Henry et al., 2014]

Contribution (in RTNS 2015):

Compute WCET by encoding the semantics and shared resource accesses into an optimization problem (SMT)

Outline: TDMA Bus Timing Analysis

1 Approaches in WCET Analysis of TDMA

2 WCET Analysis by SMT Encoding

- Naive SMT Approach
- Offset-based SMT Encoding

3 Experimental Evaluation

4 Summary and Future Work of Part I

WCET Analysis by SMT Encoding

- Bounded Model Checking
    - Encode the semantics into a Satisfiability Modulo Theory problem

WCET Analysis by SMT Encoding

- Bounded Model Checking
  - Encode the semantics into a Satisfiability Modulo Theory problem

$$\underbrace{\text{SMT query}}_{\text{assert}(\wedge\,\text{expr})} = \text{``Is there a trace with a feasible path?''}$$

- SMT-solver response:
  - SAT: There is a feasible execution path
  - UNSAT: There is no feasible execution path

WCET Analysis by SMT Encoding

- Bounded Model Checking
    - Encode the semantics into a Satisfiability Modulo Theory problem
- **Add execution times on the paths**

$$\underbrace{\text{SMT query}}_{\text{assert}(\wedge\text{expr})} = \text{“Is there a trace with a feasible path}$$

...such that the execution time is greater than X?”

- SMT-solver response:
    - SAT: **There is a feasible path with an execution time $> X$**
    - UNSAT: **X is an upper-bound on WCET**

WCET Analysis by SMT Encoding

- Bounded Model Checking
  - Encode the semantics into a Satisfiability Modulo Theory problem
- **Add execution times on the paths**

$$\underbrace{\text{SMT query}}_{\text{assert}(\bigwedge \text{expr})} = \text{“Is there a trace with a feasible path}$$

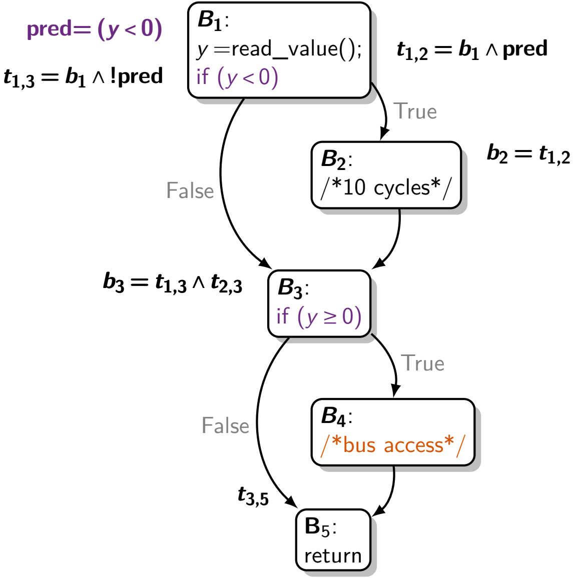
...such that the execution time is greater than X?”

- SMT-solver response:
  - SAT: **There is a feasible path with an execution time** $> X$
  - UNSAT: X **is an upper-bound on WCET**

Goal

Find the smallest X, such that Execution Time $> X$ is UNSAT

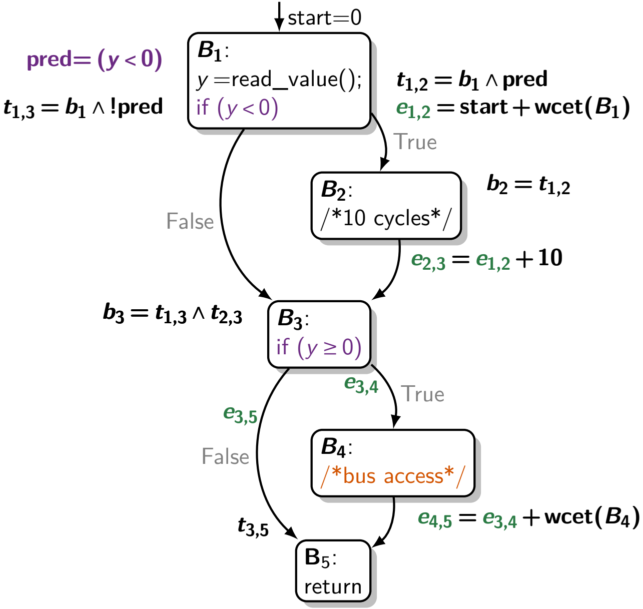
Example: Semantics and Timing Encoding



Previous work in [Henry et al., 2014]

- b_i "true" $\overset{\text{def}}{\Longleftrightarrow}$ B_i executed
- $t_{i,j}$ "true" $\overset{\text{def}}{\Longleftrightarrow}$ $B_i \rightarrow B_j$ taken

Example: Semantics and Timing Encoding



start=0

B_1: y =read_value(); if $(y<0)$

B_2: /*10 cycles*/

B_3: if $(y \geq 0)$

B_4: /*bus access*/

B_5: return

True

False

True

False

$\mathbf{pred} = (y < 0)$

$t_{1,3} = b_1 \wedge !\mathbf{pred}$

$t_{1,2} = b_1 \wedge \mathbf{pred}$

$e_{1,2} = \mathbf{start} + \mathbf{wcet}(B_1)$

$b_2 = t_{1,2}$

$e_{2,3} = e_{1,2} + 10$

$b_3 = t_{1,3} \wedge t_{2,3}$

$e_{3,4}$

$e_{3,5}$

$t_{3,5}$

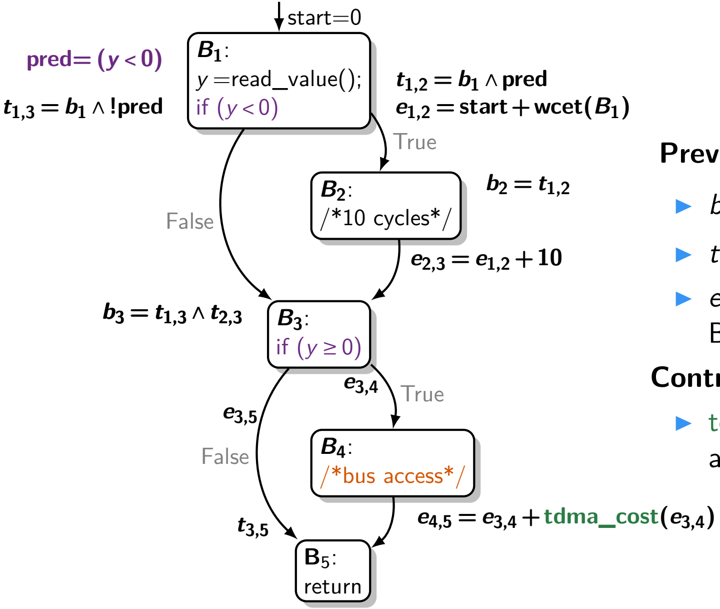
$e_{4,5} = e_{3,4} + \mathbf{wcet}(B_4)$

execution time = if $t_{3,5}$ then $e_{3,5}$ else $e_{4,5}$

Previous work in [Henry et al., 2014]

- b_i *"true"* $\overset{\text{def}}{\Longleftrightarrow}$ B_i executed
- $t_{i,j}$ *"true"* $\overset{\text{def}}{\Longleftrightarrow}$ $B_i \rightarrow B_j$ taken
- $e_{i,j}$ *execution time* at transition $B_i \rightarrow B_j$

Example: Semantics and Timing Encoding



execution time = if $t_{3,5}$ then $e_{3,5}$ else $e_{4,5}$

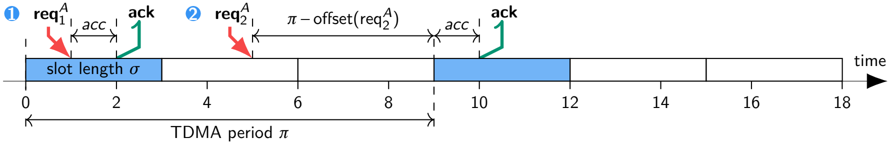
Previous work in [Henry et al., 2014]

- b_i *"true"* $\overset{\text{def}}{\Longleftrightarrow}$ B_i executed
- $t_{i,j}$ *"true"* $\overset{\text{def}}{\Longleftrightarrow}$ $B_i \rightarrow B_j$ taken
- $e_{i,j}$ *execution time* at transition $\mathrm{B}_i \rightarrow \mathrm{B}_j$

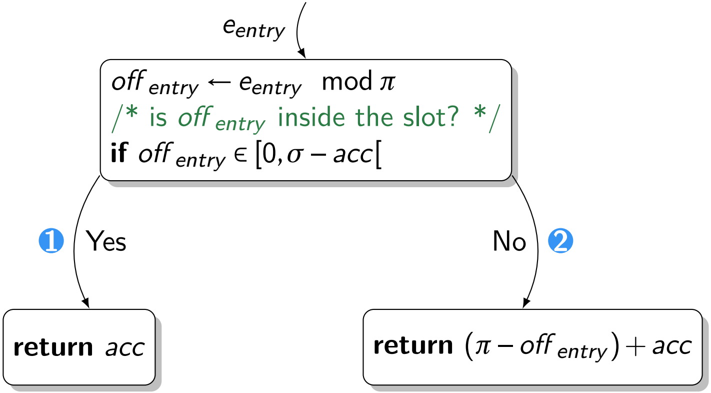
Contribution

- tdma_cost() execution time of a bus access

Naive SMT Encoding



tdma_cost(e_{entry}): returns the execution time of a bus access



e_{entry}

$off_{entry} \leftarrow e_{entry} \mod \pi$

/* is off_{entry} inside the slot? */

if $off_{entry} \in [0, \sigma - acc[$

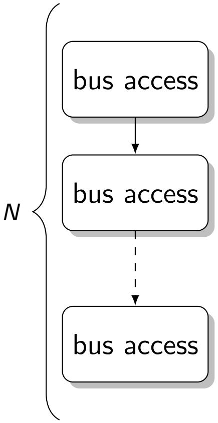
1 Yes

return acc

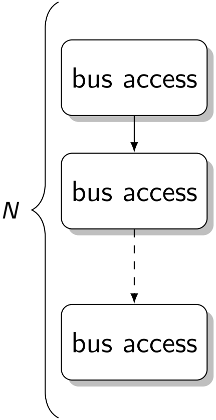
No 2

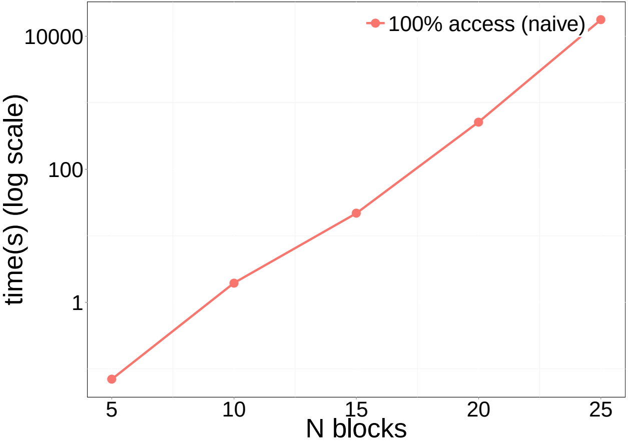
return $(\pi - off_{entry}) + acc$

Performance of the Naive Encoding

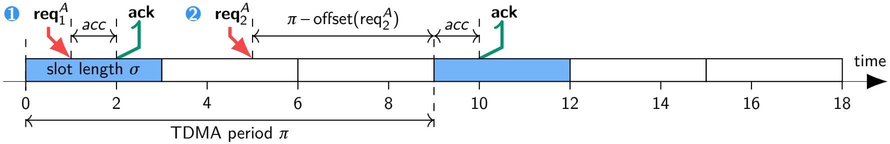


Performance of the Naive Encoding

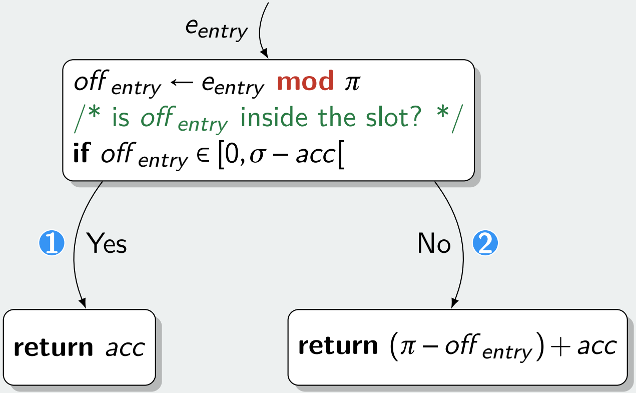




Naive SMT Encoding



tdma_cost: returns the execution time of a bus access



e_{entry}

$off_{entry} \leftarrow e_{entry}$ **mod** π

/* is off_{entry} inside the slot? */

if $off_{entry} \in [0, \sigma - acc[$

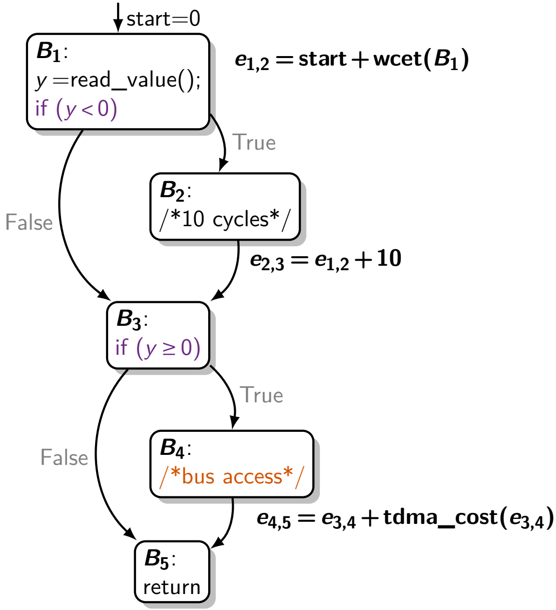
1 Yes

No 2

return acc

return $(\pi - off_{entry}) + acc$

Offset-based SMT Encoding



$\boldsymbol{e_{1,2} = \text{start} + \text{wcet}(B_1)}$

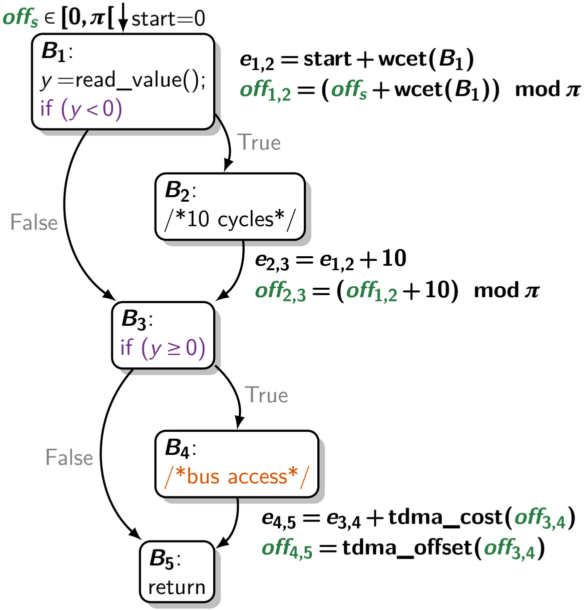
$\boldsymbol{e_{2,3} = e_{1,2} + 10}$

$\boldsymbol{e_{4,5} = e_{3,4} + \text{tdma\_cost}(e_{3,4})}$

- $\text{off}_{i,j} = e_{i,j} \mod \pi$

execution time = if $t_{3,5}$ then $e_{3,5}$ else $e_{4,5}$

Offset-based SMT Encoding



$off_s \in [0, \pi[$ start=0

B_1:
y =read_value();
if $(y < 0)$

$e_{1,2} = \text{start} + \text{wcet}(B_1)$

$off_{1,2} = (off_s + \text{wcet}(B_1)) \mod \pi$

True

False

B_2:
/*10 cycles*/

$e_{2,3} = e_{1,2} + 10$

$off_{2,3} = (off_{1,2} + 10) \mod \pi$

B_3:
if $(y \geq 0)$

True

False

B_4:
/*bus access*/

$e_{4,5} = e_{3,4} + \text{tdma\_cost}(off_{3,4})$

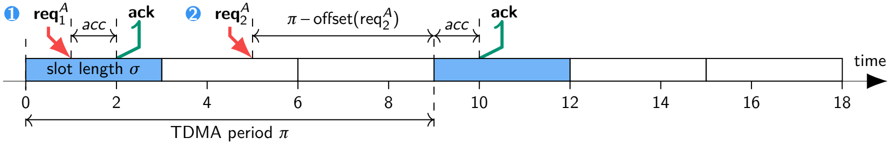
$off_{4,5} = \text{tdma\_offset}(off_{3,4})$

B_5:
return

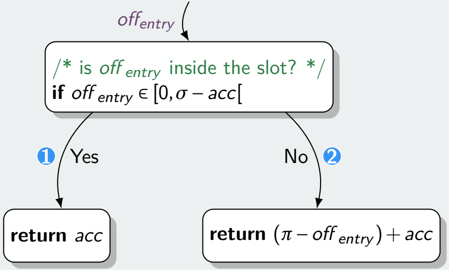
execution time = if $t_{3,5}$ then $e_{3,5}$ else $e_{4,5}$

- $\text{off}_{i,j} = e_{i,j} \mod \pi$
- $\text{off}_{i,j}$ offset at transition $B_i \rightarrow B_j$

Offset-based SMT Encoding



tdma_cost: returns the time after a bus access



off_{entry}

/* is off_{entry} inside the slot? */
if $off_{entry} \in [0, \sigma - acc[$

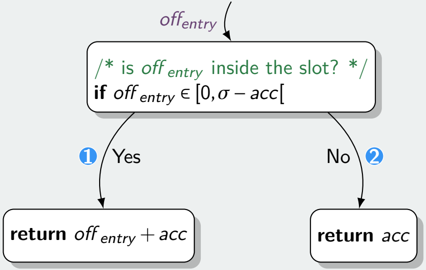
❶ Yes

No ❷

return acc

return $(\pi - off_{entry}) + acc$

tdma_offset: returns the offset after a bus access



off_{entry}

/* is off_{entry} inside the slot? */
if $off_{entry} \in [0, \sigma - acc[$

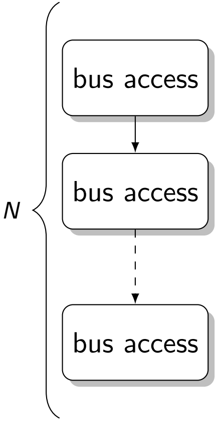
❶ Yes

No ❷

return $off_{entry} + acc$

return acc

Performance of the Offset-based Encoding

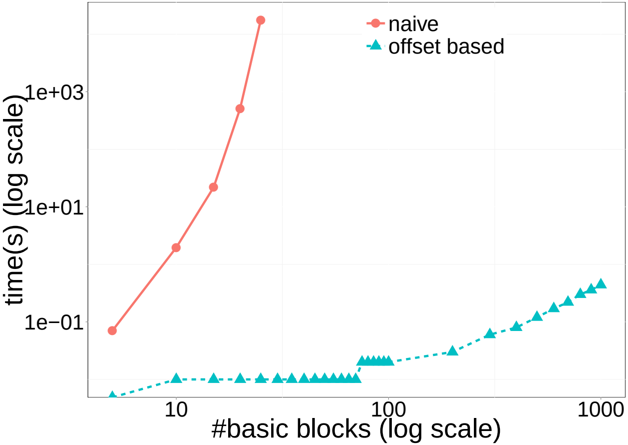


N

bus access

bus access

bus access



Outline: TDMA Bus Timing Analysis

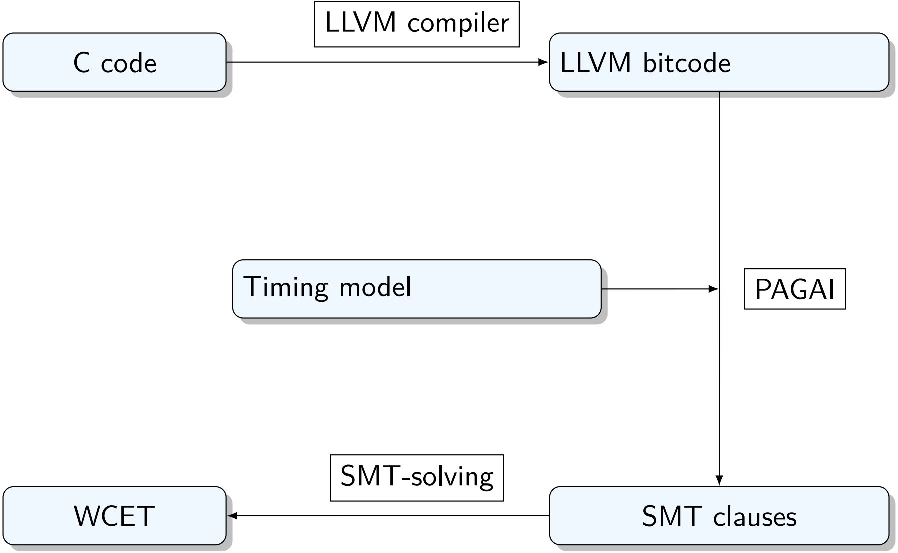
1 Approaches in WCET Analysis of TDMA

2 WCET Analysis by SMT Encoding
- Naive SMT Approach
- Offset-based SMT Encoding

3 Experimental Evaluation

4 Summary and Future Work of Part I

Proof-of-Concept Implementation



C code

LLVM compiler

LLVM bitcode

Timing model

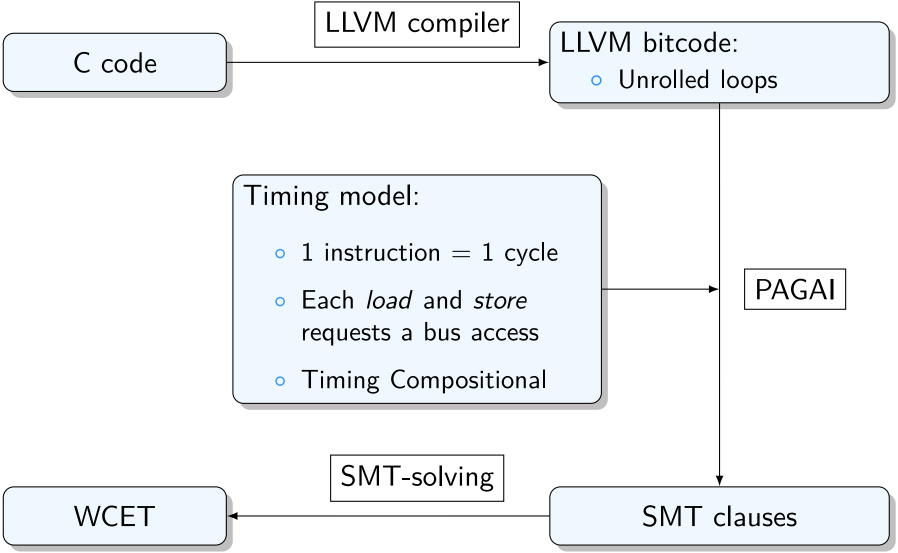
PAGAI

SMT clauses

SMT-solving

WCET

Proof-of-Concept Implementation



C code

LLVM compiler

LLVM bitcode:

- Unrolled loops

Timing model:

- 1 instruction = 1 cycle
- Each *load* and *store* requests a bus access
- Timing Compositional

PAGAI

SMT clauses

SMT-solving

WCET

Evaluation: Benchmark Descriptions

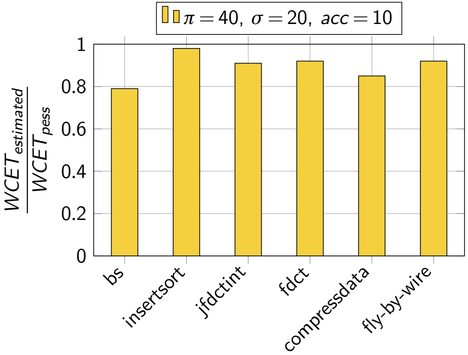
Benchmark from TACLEBench suite [1]

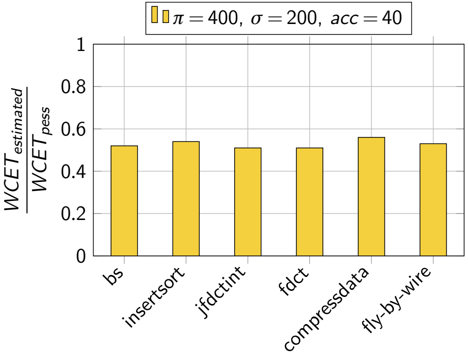
Name	Description	#LLVM instr.	#bus access
bs	Binary search	231	12
insertsort	Insertion sort on a reversed array	493	65
jfdctint	Discrete Cosine Transformation	2334	448
fdct	Fast Discrete Cosine Transform	2502	385
compressdata	Data compression program adopted from SPEC95	674	131
fly-by-wire	UAV fly-by-wire software	2815	515

[1] `https://github.com/tacle/tacle-bench`

Evaluation: Experiments

Comparison between estimated WCET and pessimistic WCET





Best-case of gain: All requests are within the TDMA slots
Worst-case of gain: All requests have worst-case delay

Outline: TDMA Bus Timing Analysis

1 Approaches in WCET Analysis of TDMA

2 WCET Analysis by SMT Encoding
- Naive SMT Approach
- Offset-based SMT Encoding

3 Experimental Evaluation

4 Summary and Future Work of Part I

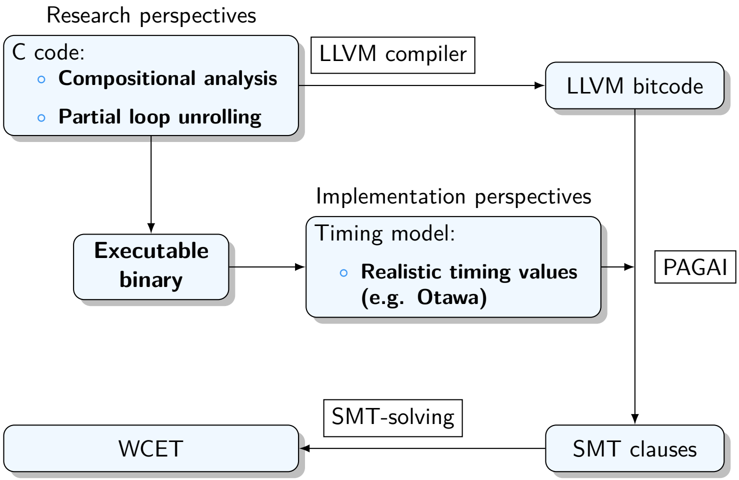
Summary of Part I

- SMT encodings for TDMA access
- Feasible path analysis combined with the WCET computation
- Comparison between different encodings
- Validation with small but relevant benchmarks

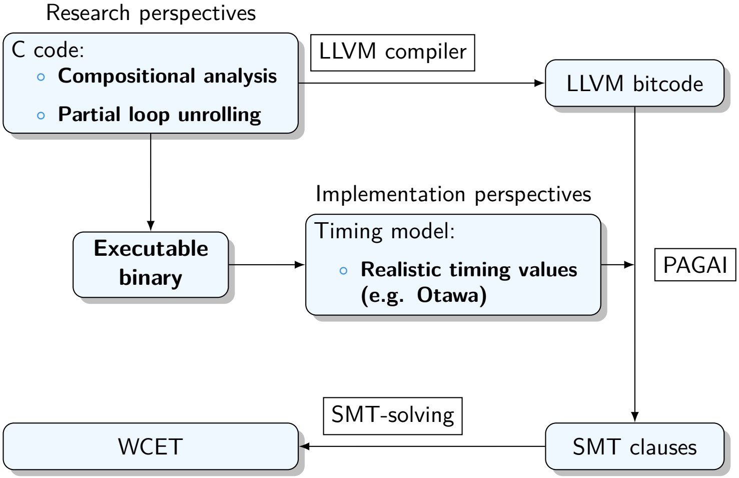
Find in the manuscript:

- Linearization of SMT encoding (modulo operators)
- Other possible SMT encodings

Future Work of Part I


Research perspectives
C code:
◦ **Compositional analysis**
◦ **Partial loop unrolling**
LLVM compiler
LLVM bitcode
Implementation perspectives
Executable binary
Timing model:
◦ **Realistic timing values (e.g. Otawa)**
PAGAI
SMT-solving
WCET
SMT clauses

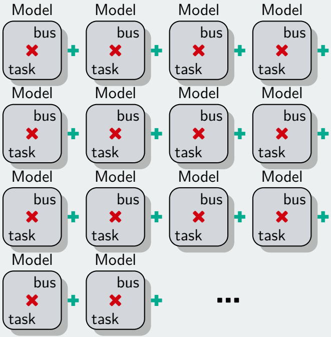
Future Work of Part I



⇝ **SMT is an interesting research direction for WCET Analysis**

From TDMA to Other Arbitration Policy

Full isolation with TDMA 🚦



Model bus ✖ task + Model bus ✖ task + Model bus ✖ task + Model bus ✖ task +

Model bus ✖ task + Model bus ✖ task + Model bus ✖ task + Model bus ✖ task +

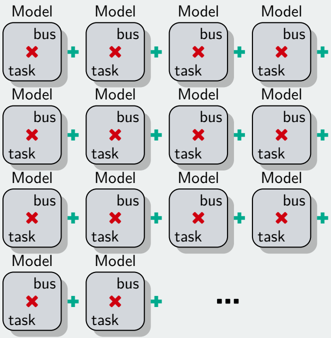
Model bus ✖ task + Model bus ✖ task + Model bus ✖ task + Model bus ✖ task +

Model bus ✖ task + Model bus ✖ task + ...

Linear complexity
in the number of tasks

From TDMA to Other Arbitration Policy

Full isolation with TDMA



Model bus ✖ task + Model bus ✖ task + Model bus ✖ task + Model bus ✖ task +

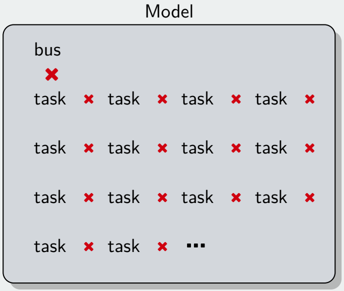
Model bus ✖ task + Model bus ✖ task + Model bus ✖ task + Model bus ✖ task +

Model bus ✖ task + Model bus ✖ task + Model bus ✖ task + Model bus ✖ task +

Model bus ✖ task + Model bus ✖ task + ...

Linear complexity
in the number of tasks

No isolation



Model

bus ✖

task ✖ task ✖ task ✖ task ✖

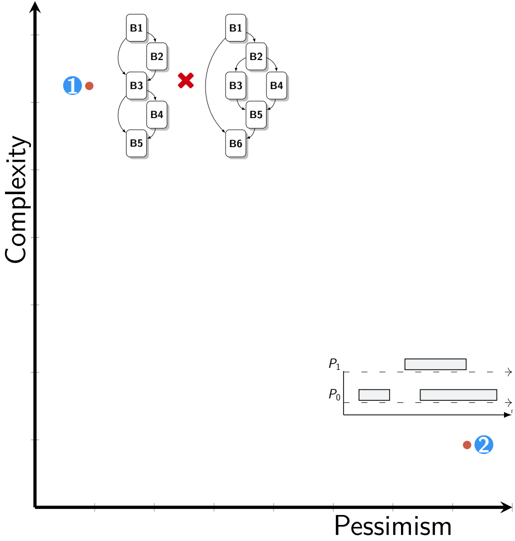
task ✖ task ✖ task ✖ task ✖

task ✖ task ✖ task ✖ task ✖

task ✖ task ✖ ...

States explosion!

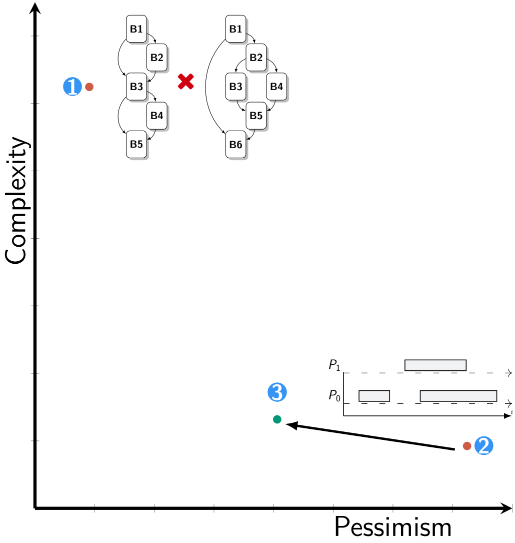
Analysis of Large Multi and Many-Cores



➊ Exact analysis

➋ Account for any interference **globally** during the task's execution

Analysis of Large Multi and Many-Cores



❶ Exact analysis

❷ Account for any interference **globally** during the task's execution

❸ Exploit any information about:

- The target architecture Kalray MPPA2
  - Reduce the interference
  - Model precisely the shared resources
- The target application model Synchronous Data Flow

Many-Core Response Time Analysis

Outline: Many-Core Response Time Analysis

5 Implementation Choices of Synchronous Data Flow Programs

6 Multicore Response Time Analysis of SDF Programs

7 Target Many-Core: Kalray MPPA2

8 Evaluation

9 Summary and Future Work of Part II

Outline: Many-Core Response Time Analysis

5 Implementation Choices of Synchronous Data Flow Programs

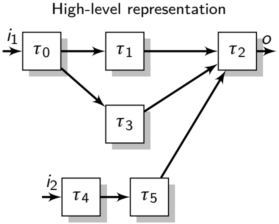
6 Multicore Response Time Analysis of SDF Programs

7 Target Many-Core: Kalray MPPA2

8 Evaluation

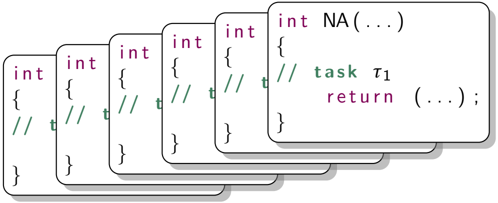
9 Summary and Future Work of Part II

Implementation Choices: SDF on Multi/Many-Cores



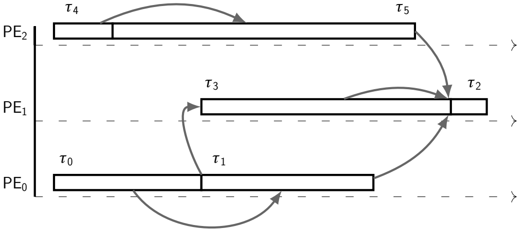
Multi/Many-Core

code generation



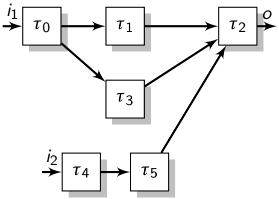
```
int NA(...)
{
// task τ1
    return (...);
}
```

Static, time triggered, non-preemptive scheduling



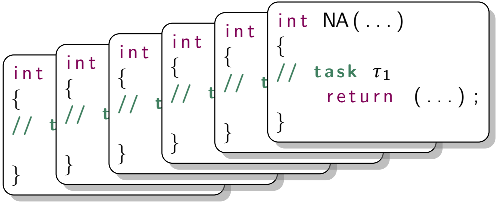
Implementation Choices: SDF on Multi/Many-Cores

High-level representation



Multi/Many-Core

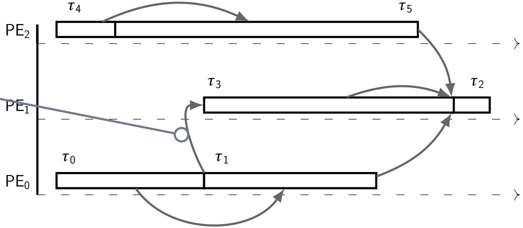
code generation



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

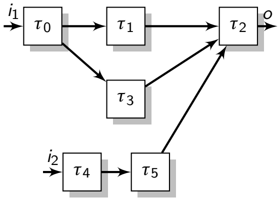
Static, time triggered, non-preemptive scheduling

✓ Respect the dependency constraints



Implementation Choices: SDF on Multi/Many-Cores

High-level representation



Multi/Many-Core

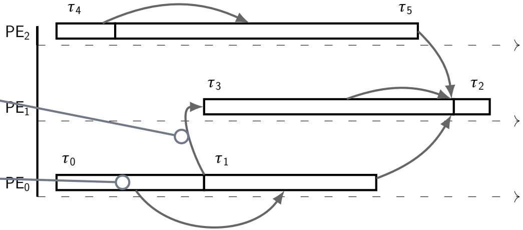
code generation

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

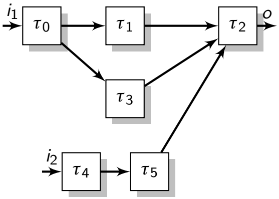
Static, time triggered, non-preemptive scheduling

✓ Respect the dependency constraints

✓ account for precise upper bounds on the interference



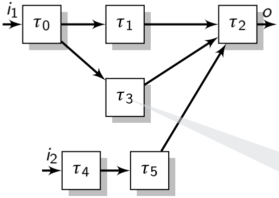
Model of the Application



An execution instance is:

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

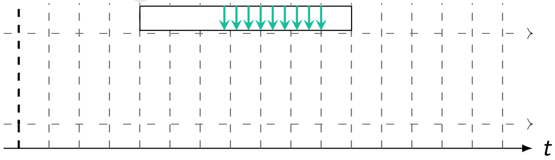
Model of the Application



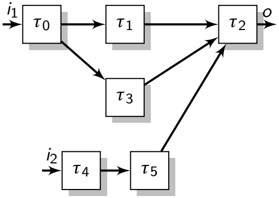
An execution instance is:

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

Each task τ_i:



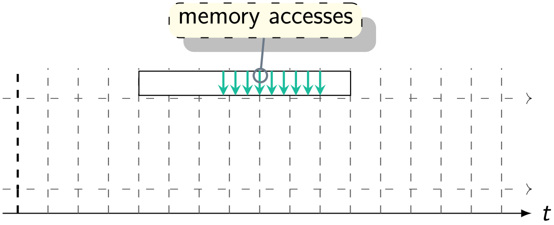
Model of the Application



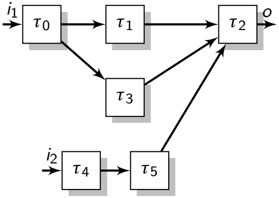
An execution instance is:

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

Each task τ_i:



Model of the Application

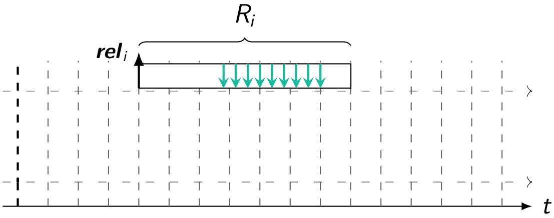


An execution instance is:

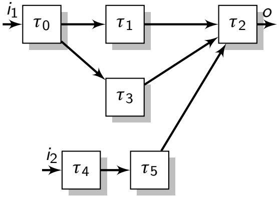
- Direct Acyclic Task Graph
- Mono-rate (or at least harmonic rates)
- Fixed mapping and execution order

Each task τ_i:

- **Release date** (rel_i). **Response time** (R_i)



Model of the Application

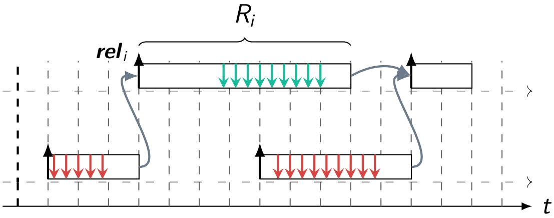


An execution instance is:

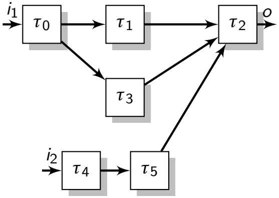
- Direct Acyclic Task Graph
- Mono-rate (or at least harmonic rates)
- Fixed mapping and execution order

Each task τ_i:

- **Release date** (rel_i). **Response time** (R_i)



Model of the Application

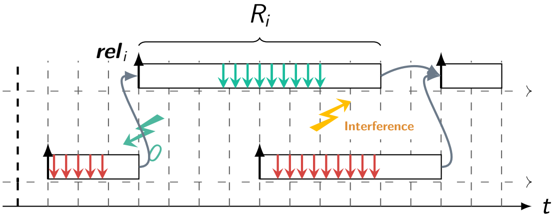


An execution instance is:

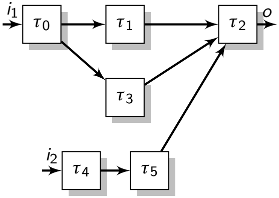
- Direct Acyclic Task Graph
- Mono-rate (or at least harmonic rates)
- Fixed mapping and execution order

Each task τ_i:

- **Release date** (rel_i). **Response time** (R_i)



Model of the Application

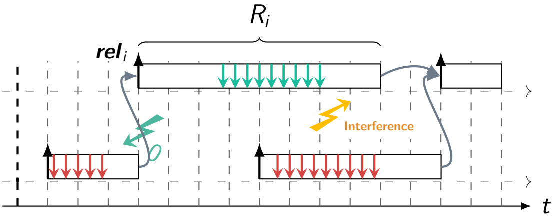


An execution instance is:

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

Each task τ_i:

- **Release date** (rel_i). **Response time** (R_i)



Static Non-Preemptive Scheduling

- Find R_i including interference
- Find rel_i respecting dependencies

Outline: Many-Core Response Time Analysis

5 Implementation Choices of Synchronous Data Flow Programs

6 Multicore Response Time Analysis of SDF Programs

7 Target Many-Core: Kalray MPPA2

8 Evaluation

9 Summary and Future Work of Part II

Response Time Analysis

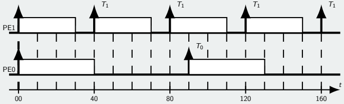
Existing work [Altmeyer et al., 2015]

$$\forall i: \; R_i \;=\; WCET_i \;+\; I^{BUS}(R_i) \;+ \ldots$$

For each task i:

- Response Time
- WCET in isolation
- Bus Interference

Independent tasks



Response Time Analysis

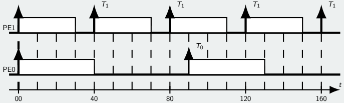
Existing work [Altmeyer et al., 2015]

$$\forall i:\ R_i = WCET_i + I^{BUS}(R_i) + \dots$$

For each task i:

- Response Time
- WCET in isolation
- Bus Interference

Independent tasks

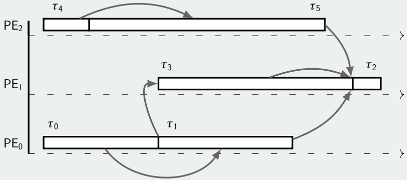


Contribution (in RTNS 2016)

$$\forall i:\ R_i = \mathrm{WCET}_i + I^{BUS}(R_i, \Theta) + \dots$$

- WCET in isolation
- Set of release dates of all tasks
- Bounded interference

Dependent tasks



Response Time Analysis

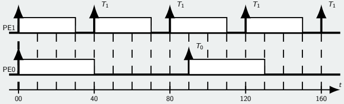
Existing work [Altmeyer et al., 2015]

$$\forall i: \; R_i = WCET_i + I^{BUS}(R_i) + \ldots$$

For each task i:

- Response Time
- WCET in isolation
- Bus Interference

Independent tasks

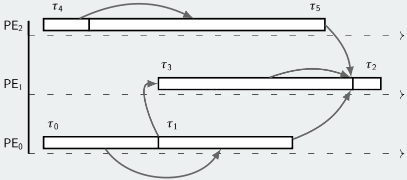


Contribution (in RTNS 2016)

$$\forall i: \; R_i = \text{WCET}_i + I^{BUS}(R_i, \Theta) + \ldots$$

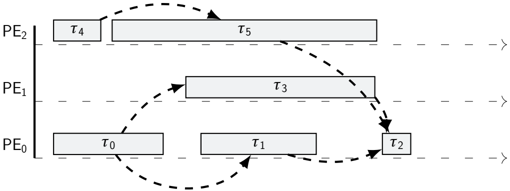
- WCET in isolation
- Set of release dates of all tasks
- Bounded interference

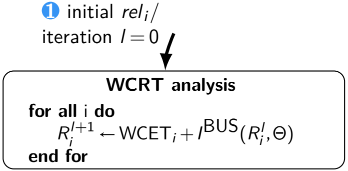
Dependent tasks



Recursive formula ⇒ iterative algorithm.

Response Time Analysis with Dependencies





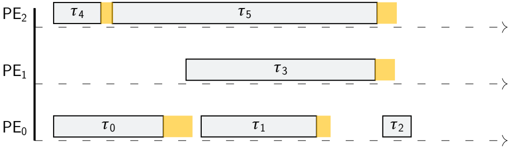
❶ initial rel_i/ iteration $l = 0$

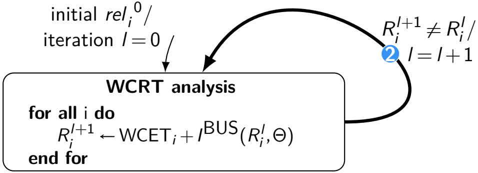
WCRT analysis

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

1. Start with initial release dates

Response Time Analysis with Dependencies



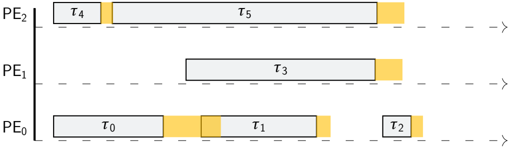


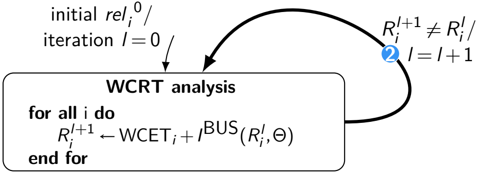
```
for all i do
    $R_i^{l+1} \leftarrow \text{WCET}_i + I^{\text{BUS}}(R_i^l, \Theta)$
end for
```

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

   ...

Response Time Analysis with Dependencies



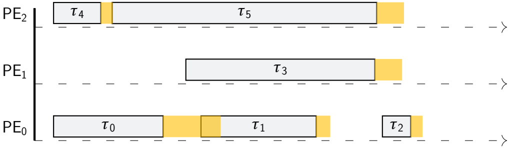


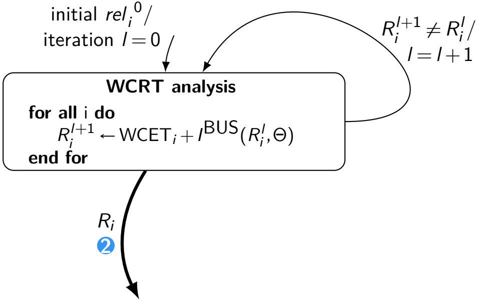
```
WCRT analysis
for all i do
    $R_i^{l+1} \leftarrow \text{WCET}_i + I^{\text{BUS}}(R_i^l, \Theta)$
end for
```

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

   ... ...

Response Time Analysis with Dependencies





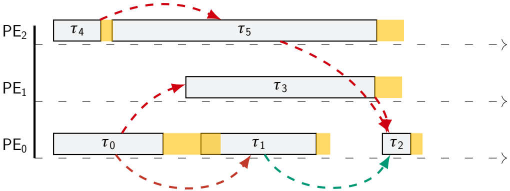
WCRT analysis

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

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

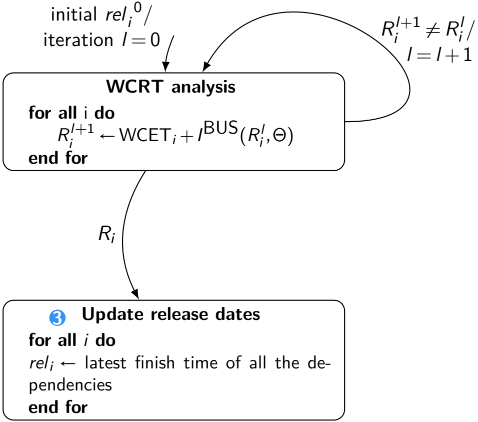
   … … … a fixed-point is reached!

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



initial rel_i^0 / iteration $l = 0$

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

WCRT analysis

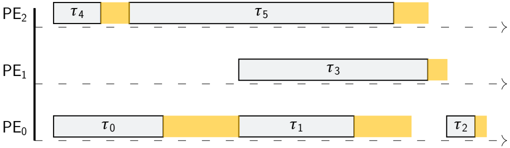
```
for all i do
    $R_i^{l+1} \leftarrow \mathrm{WCET}_i + I^{\mathrm{BUS}}(R_i^l, \Theta)$
end for
```

R_i

3 Update release dates

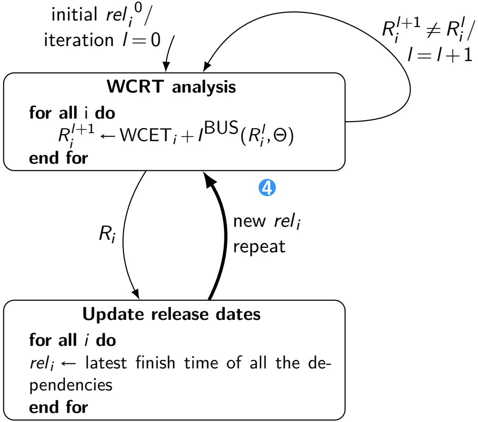
```
for all $i$ do
$rel_i \leftarrow$ latest finish time of all the dependencies
end for
```

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



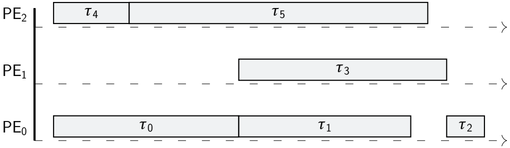
WCRT analysis

```
for all i do
    $R_i^{l+1} \leftarrow \text{WCET}_i + I^{\text{BUS}}(R_i^l, \Theta)$
end for
```

Update release dates

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

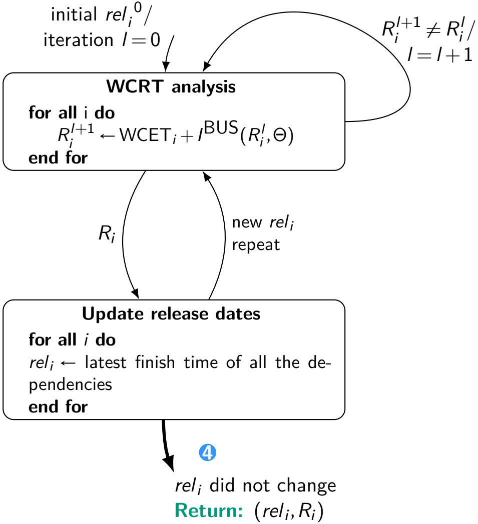
Response Time Analysis with Dependencies



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 / iteration $l = 0$

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

WCRT analysis

```
for all i do
    $R_i^{l+1} \leftarrow \text{WCET}_i + I^{\text{BUS}}(R_i^l, \Theta)$
end for
```

R_i

new rel_i repeat

Update release dates

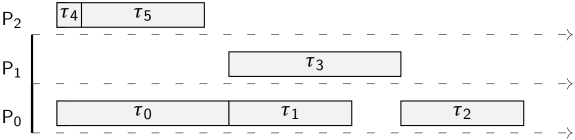
```
for all $i$ do
    $rel_i$ ← latest finish time of all the dependencies
end for
```

❹

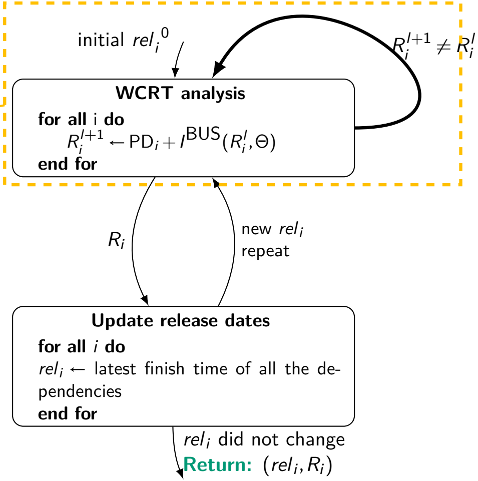
rel_i did not change

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

R_i

new rel_i
repeat

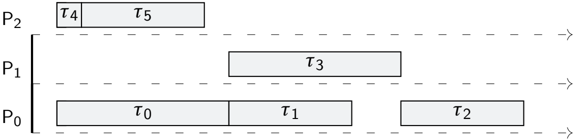
Update release dates

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

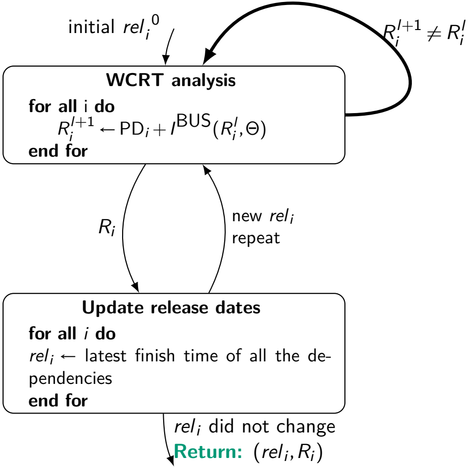
rel_i did not change

Return: (rel_i, R_i)

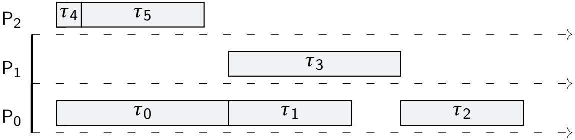
Convergence Toward a Fixed-point



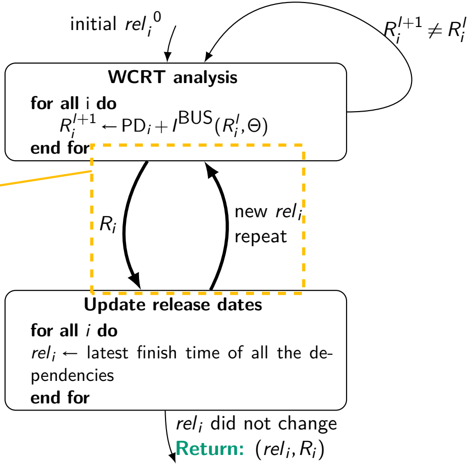
- Convergence of the 1^{st} fixed-point iteration:
  - Monotonic and bounded ✓



Convergence Toward a Fixed-point



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

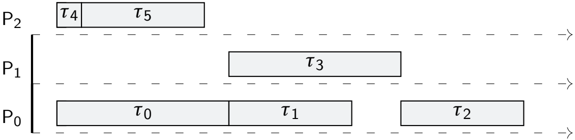


```
for all i do
    $R_i^{l+1} \leftarrow \mathrm{PD}_i + I^{\mathrm{BUS}}(R_i^l, \Theta)$
end for
```

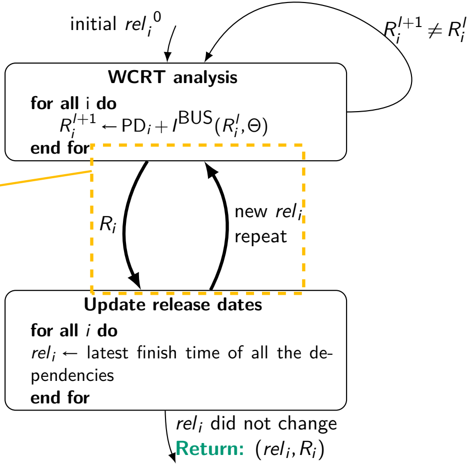
```
for all $i$ do
$rel_i \leftarrow$ latest finish time of all the dependencies
end for
```

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. ?


initial $rel_i^{\,0}$

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

WCRT analysis

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

R_i

new rel_i repeat

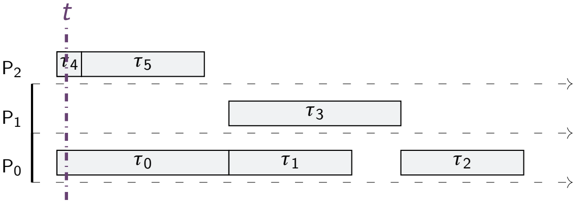
Update release dates

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

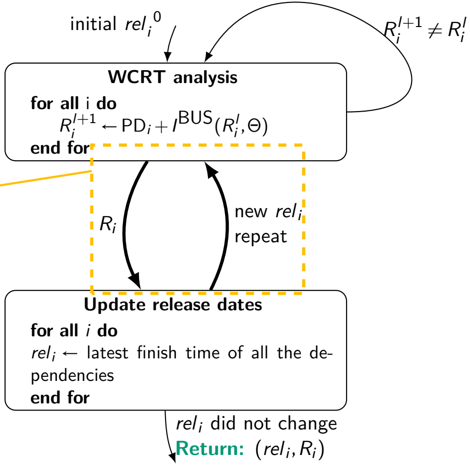
rel_i did not change

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

Update release dates

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

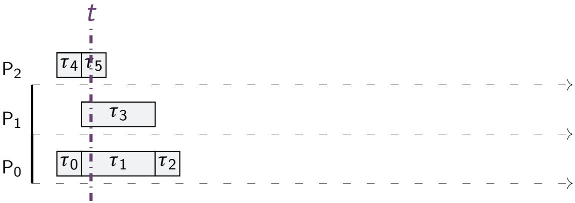
Return: (rel_i, R_i)

Theorem

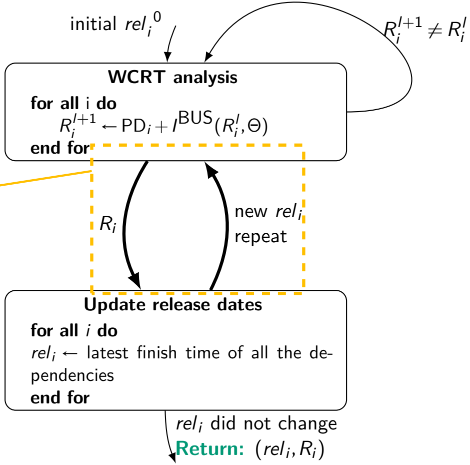
At each iteration, at least one task finds its final release date.

Full proof in the manuscript.

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
    $R_i^{l+1} \leftarrow \mathrm{PD}_i + I^{\mathrm{BUS}}(R_i^l, \Theta)$
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

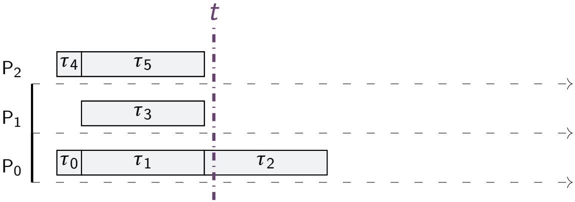
Return: (rel_i, R_i)

Theorem

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

Full proof in the manuscript.

Convergence Toward a Fixed-point

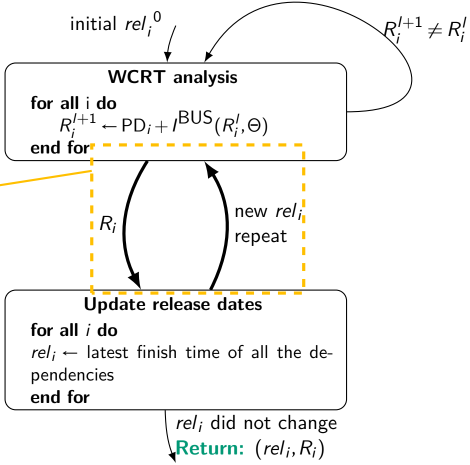


- Convergence of the 1^{st} fixed-point iteration:
  - Monotonic and bounded ✓
- Convergence of the 2^{nd} fixed-point iteration:
  - No monotonicity: R_i and rel_i may grow or shrink at each iteration. ?

Theorem

At each iteration, <u>at least one task</u> finds its final release date.

Full proof in the manuscript.



initial ${rel_i}^0$

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

WCRT analysis

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

R_i

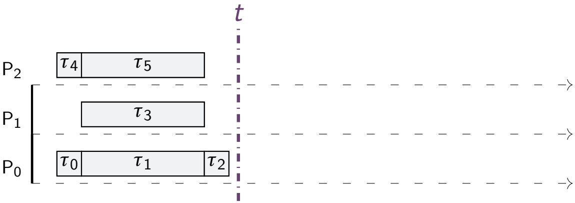
new rel_i
repeat

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

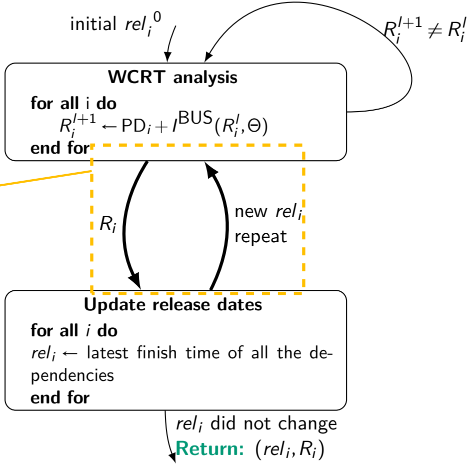


- Convergence of the 1^{st} fixed-point iteration:
  - Monotonic and bounded ✓
- Convergence of the 2^{nd} fixed-point iteration:
  - No monotonicity: R_i and rel_i may grow or shrink at each iteration. ?

Theorem

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

Full proof in the manuscript.



WCRT analysis

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

Outline: Many-Core Response Time Analysis

5 Implementation Choices of Synchronous Data Flow Programs

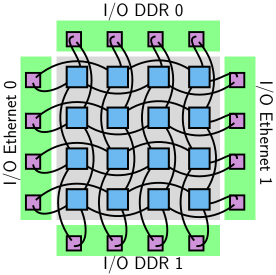
6 Multicore Response Time Analysis of SDF Programs

7 Target Many-Core: Kalray MPPA2

8 Evaluation

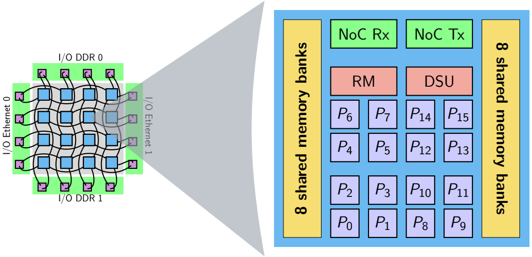
9 Summary and Future Work of Part II

Target Many-Core: Kalray MPPA2



- Kalray MPPA2 (codenamed Bostan)
- 16 compute clusters + 4 I/O clusters
- Dual NoC

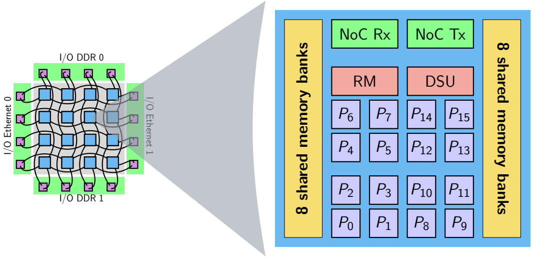
Target Many-Core: Kalray MPPA2

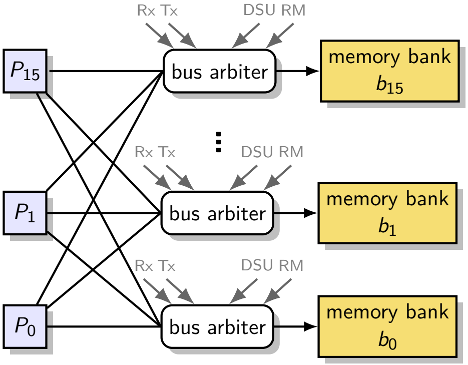


Per cluster:

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

Target Many-Core: Kalray MPPA2

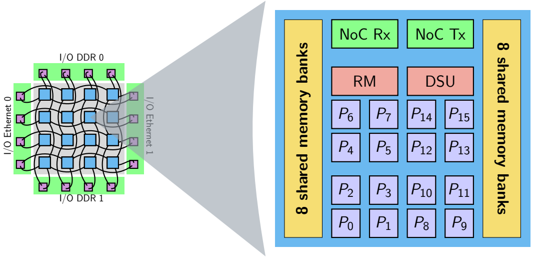


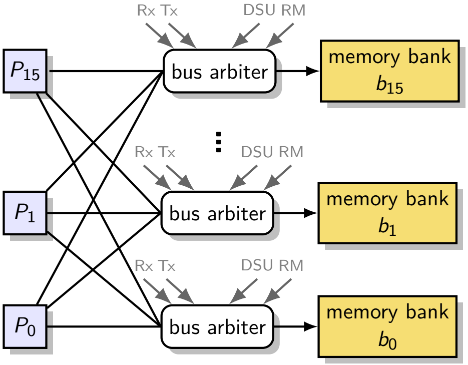


Per cluster:

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

Target Many-Core: Kalray MPPA2





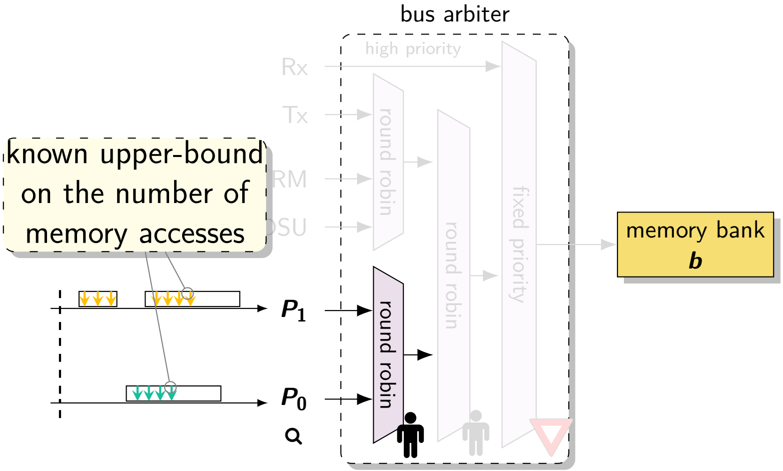
Per cluster:

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

- Possible spatial isolation

  assigning memory banks to cores
- Task execution model:
  - execute in a “local” bank
  - write to a “remote” bank
- Interference from communications

Model of the MPPA2 Bus



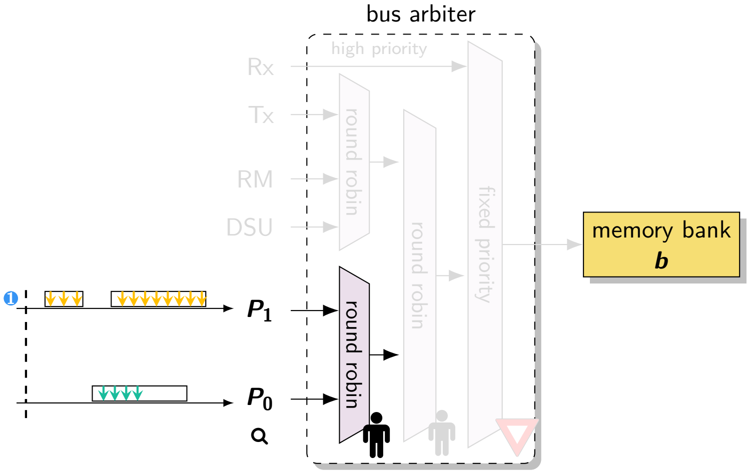
Shared Bus Model

$$I^{BUS}(R_i,\Theta) = \sum_{b\in\mathscr{B}} I_b^{BUS}(R_i,\Theta)$$

where $\mathscr{B}$: a set of memory banks

Bus interference $I_b^{\mathrm{BUS}}(R_i,\Theta)$:

Model of the MPPA2 Bus



Shared Bus Model

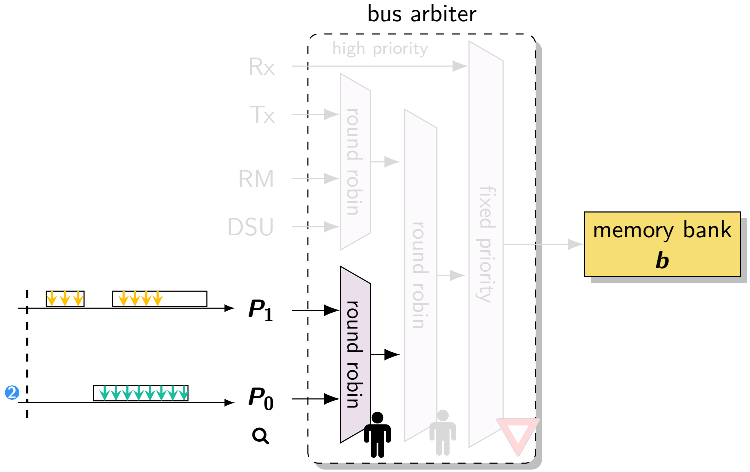
$$I^{BUS}(R_i,\Theta) = \sum_{b\in\mathscr{B}} I_b^{BUS}(R_i,\Theta)$$

where $\mathscr{B}$: a set of memory banks

Bus interference $I_b^{\mathrm{BUS}}(R_i,\Theta)$:

❶ ⚡$(P_0)=$↓↓↓…↓

Model of the MPPA2 Bus



Shared Bus Model

$$I^{BUS}(R_i,\Theta) = \sum_{b\in\mathscr{B}} I_b^{BUS}(R_i,\Theta)$$

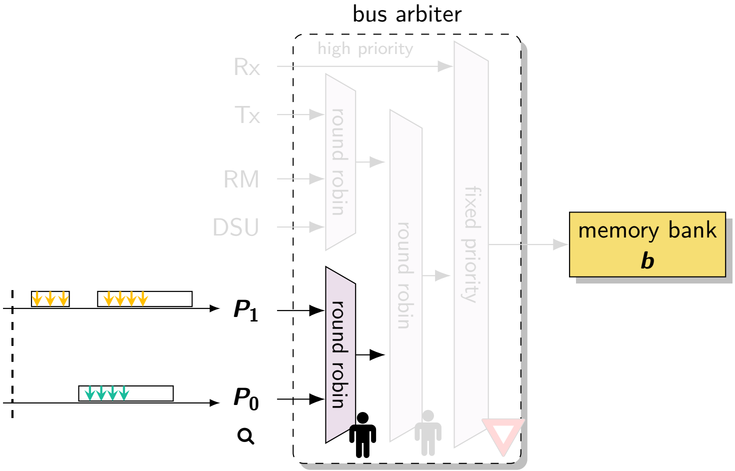
where $\mathscr{B}$: a set of memory banks

Bus interference $I_b^{\text{BUS}}(R_i,\Theta)$:

➊ ⚡$(P_0)=$↓↓↓…↓

➋ ⚡$(P_1)=$↓↓↓…↓

Model of the MPPA2 Bus



$\min(⚡(P_0),\ ⚡(P_1))$

Shared Bus Model

$$I^{BUS}(R_i,\Theta) = \sum_{b\in\mathcal{B}} I_b^{BUS}(R_i,\Theta)$$

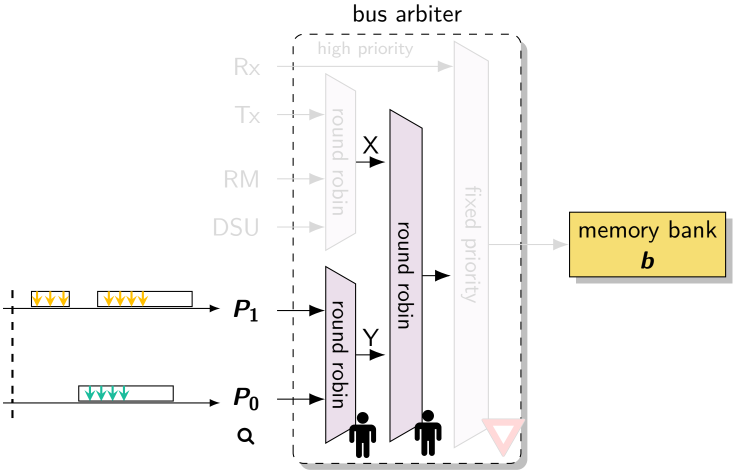
where $\mathcal{B}$: a set of memory banks

Bus interference $I_b^{\mathrm{BUS}}(R_i,\Theta)$:

❶ $⚡(P_0) = {\downarrow}{\downarrow}{\downarrow}\ldots{\downarrow}$

❷ $⚡(P_1) = {\downarrow}{\downarrow}{\downarrow}\ldots{\downarrow}$

Model of the MPPA2 Bus



Shared Bus Model

$$I^{BUS}(R_i, \Theta) = \sum_{b \in \mathscr{B}} I_b^{BUS}(R_i, \Theta)$$

where $\mathscr{B}$: a set of memory banks

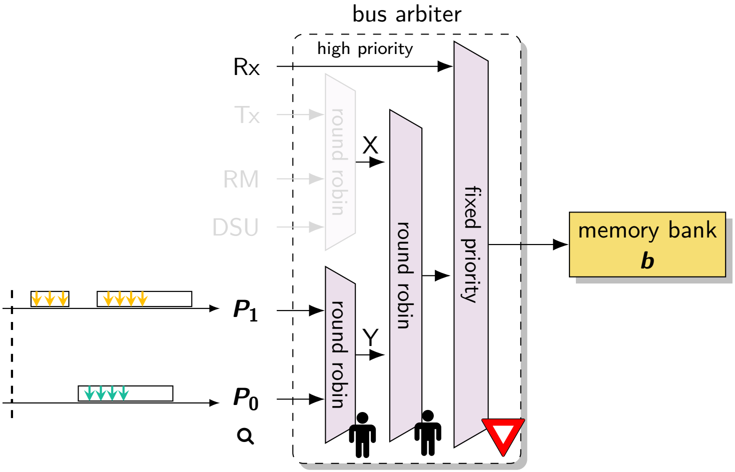
Bus interference $I_b^{\mathrm{BUS}}(R_i, \Theta)$:

❶ ⚡$(P_0) =$ ↓↓↓…↓

❷ ⚡$(P_1) =$ ↓↓↓…↓

min(⚡(P_0), ⚡(P_1)) + min(⚡(Y), ⚡(X))

Model of the MPPA2 Bus



Shared Bus Model

$$I^{BUS}(R_i,\Theta) = \sum_{b\in\mathcal{B}} I_b^{BUS}(R_i,\Theta)$$

where $\mathcal{B}$: a set of memory banks

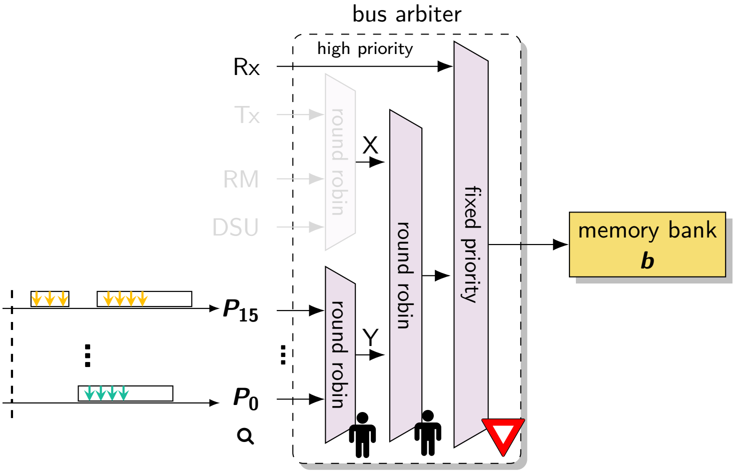
Bus interference $I_b^{\text{BUS}}(R_i,\Theta)$:

❶ ⚡$(P_0) = \downarrow\downarrow\downarrow\ldots\downarrow$

❷ ⚡$(P_1) = \downarrow\downarrow\downarrow\ldots\downarrow$

min(⚡(P_0), ⚡(P_1)) + min(⚡(Y), ⚡(X)) + ⚡(Rx)

Model of the MPPA2 Bus



Shared Bus Model

$$I^{BUS}(R_i,\Theta) = \sum_{b\in\mathscr{B}} I_b^{BUS}(R_i,\Theta)$$

where $\mathscr{B}$: a set of memory banks

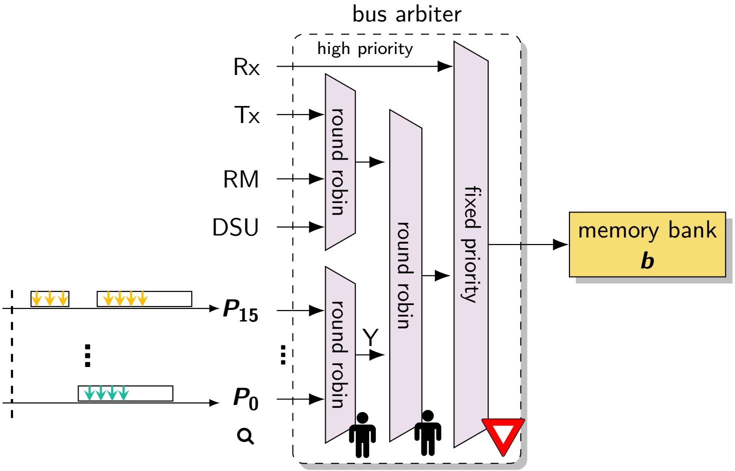
Bus interference $I_b^{\text{BUS}}(R_i,\Theta)$:

❶ ⚡(P_0)=↓↓↓...↓

❷ ⚡(P_1)=↓↓↓...↓

$\sum_{1\le i\le 15}$ min(⚡(P_0), ⚡(P_i)) + min(⚡(Y), ⚡(X)) + ⚡(Rx)

Model of the MPPA2 Bus



Shared Bus Model

$$I^{BUS}(R_i,\Theta) = \sum_{b\in\mathscr{B}} I_b^{BUS}(R_i,\Theta)$$

where $\mathscr{B}$: a set of memory banks

Bus interference $I_b^{\mathrm{BUS}}(R_i,\Theta)$:

1. ⚡(P_0)=↓↓↓…↓
2. ⚡(P_1)=↓↓↓…↓

$$\sum_{1\leq i\leq 15} \min(\text{⚡}(P_0),\ \text{⚡}(P_i)) + \min(\text{⚡}(Y),\ \text{⚡}(\mathrm{X})\ ) + \text{⚡}(\mathrm{Rx})$$

$X = \sum\{Tx, RM, DSU\}$

Outline: Many-Core Response Time Analysis

5 Implementation Choices of Synchronous Data Flow Programs

6 Multicore Response Time Analysis of SDF Programs

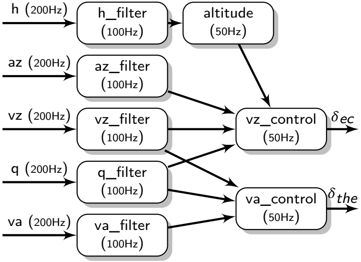
7 Target Many-Core: Kalray MPPA2

8 Evaluation

9 Summary and Future Work of Part II

Case Study: ROSACE, a Flight Management System Controller [2]

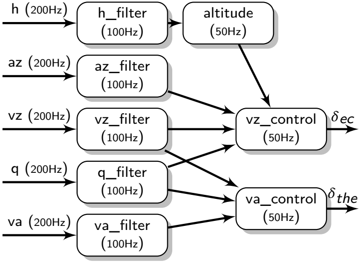
High level representation



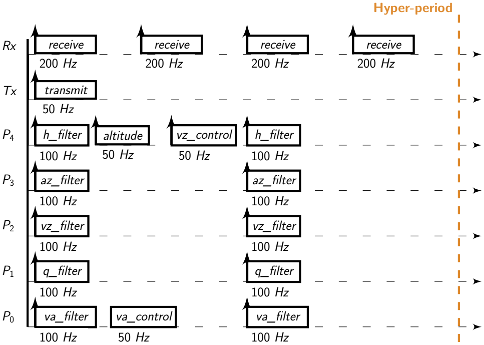
[2] Pagetti et al., 2014

Case Study: ROSACE, a Flight Management System Controller [2]

High level representation



Unrolled execution



[2] Pagetti et al., 2014

Evaluation: ROSACE Case Study

Function	WCET (cycles)	Memory accesses
altitude	275	22
az_filter	274	22
h_filter	326	24
q_filter	338	24
va_control	303	24
va_filter	301	23
vz_control	320	25
vz_filter	334	25

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

Evaluation: ROSACE Case Study

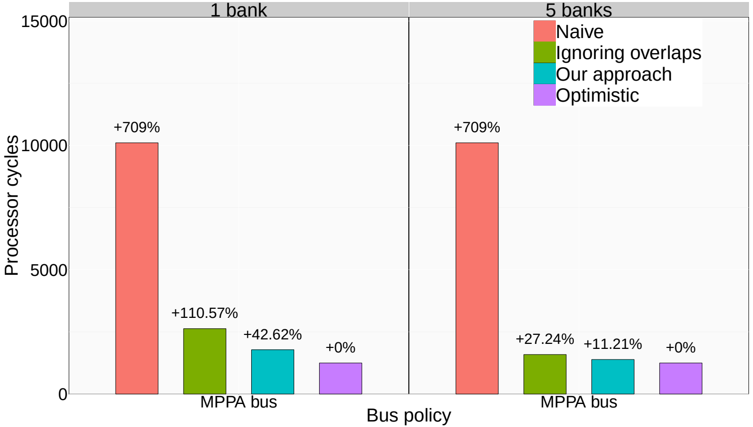
Function	WCET (cycles)	Memory accesses
altitude	275	22
az_filter	274	22
h_filter	326	24
q_filter	338	24
va_control	303	24
va_filter	301	23
vz_control	320	25
vz_filter	334	25

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

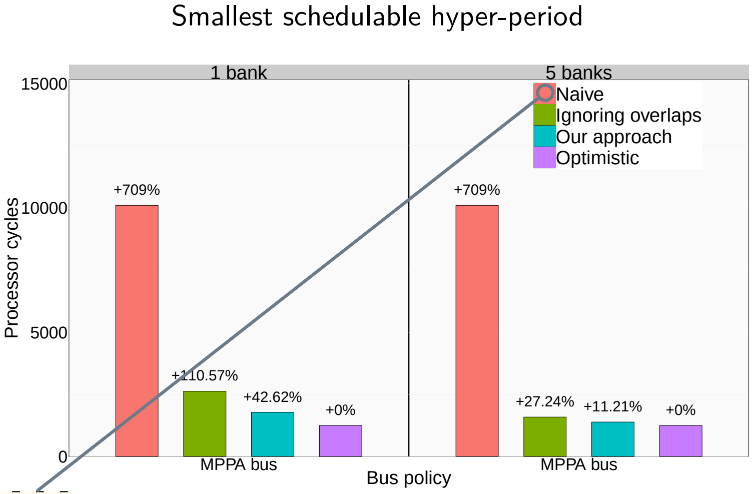
⚗ Experiments: Find the smallest schedulable hyper-period

Evaluation: Experiments

Smallest schedulable hyper-period

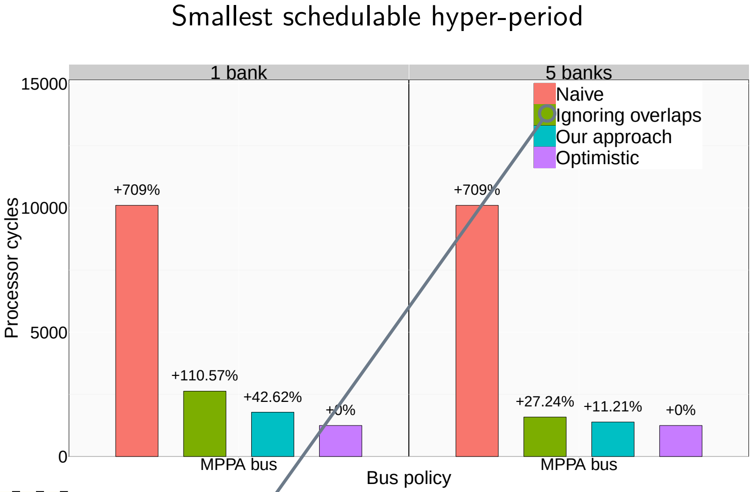


Evaluation: Experiments



Naive:
- All accesses interfere
- Rx bounded by 1 access per bank

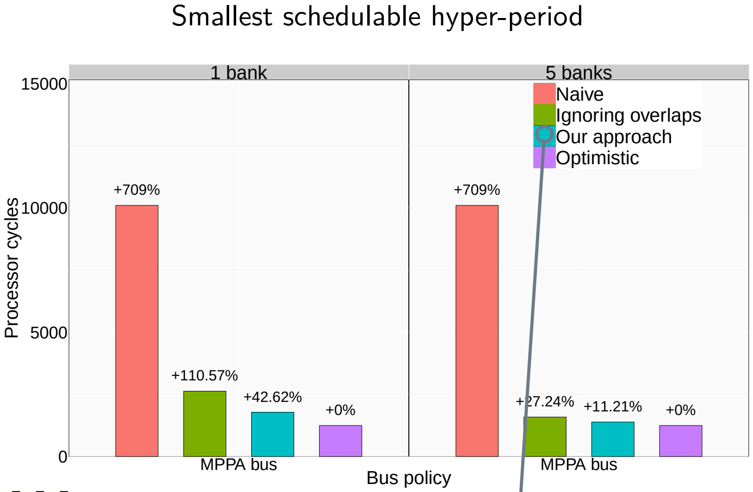
Evaluation: Experiments



Naive:
- All accesses interfere
- Rx bounded by 1 access per bank

Ignoring overlap: We don't use the release dates

Evaluation: Experiments

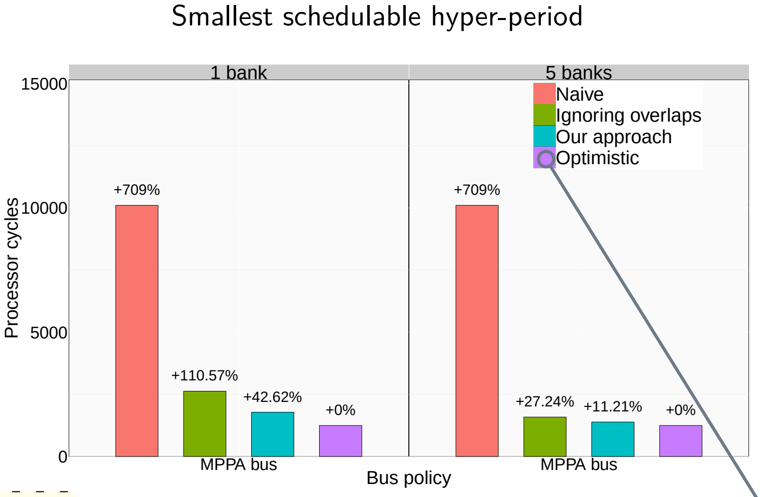


Naive:
- All accesses interfere
- Rx bounded by 1 access per bank

Ignoring overlap:
We don't use the release dates

Our approach:
We use the release dates

Evaluation: Experiments



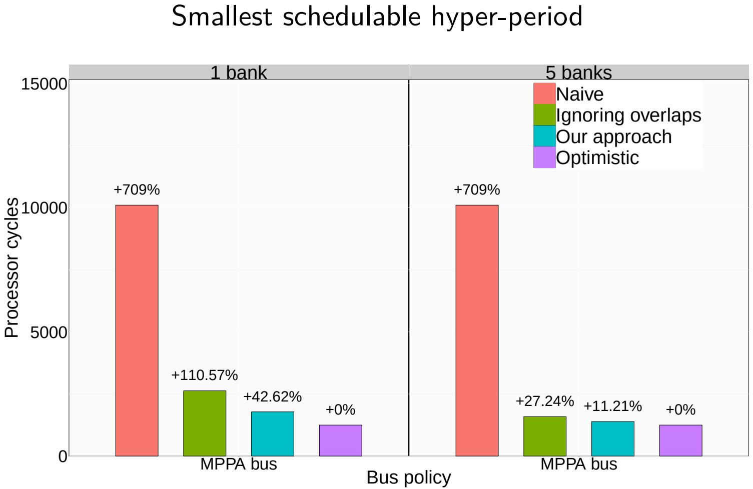
Naive:
- All accesses interfere
- Rx bounded by
 1 access per bank

Ignoring overlap:
We don't use
the release dates

Our approach:
We use the release dates

Optimistic:
No bus interference

Evaluation: Experiments



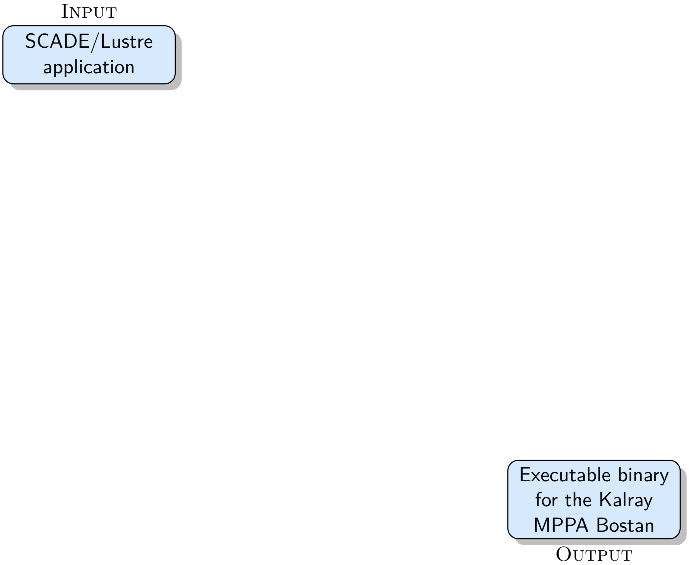
Naive:
- All accesses interfere
- Rx bounded by 1 access per bank

Ignoring overlap: We don't use the release dates

Our approach: We use the release dates

Optimistic: No bus interference

Evaluation: CAPACITES Project



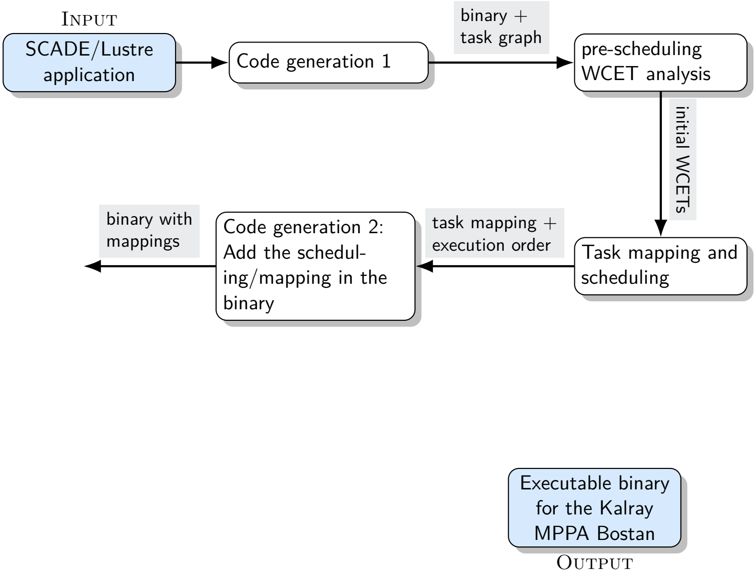
INPUT

SCADE/Lustre application

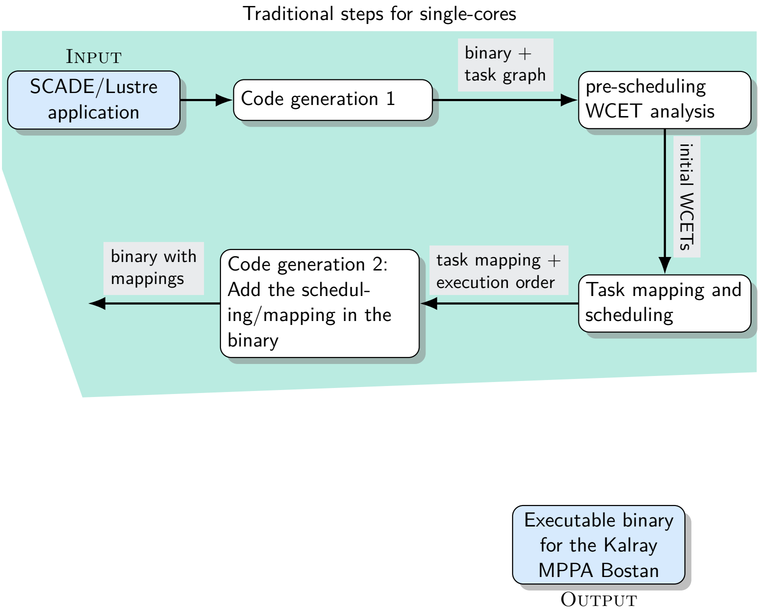
Executable binary for the Kalray MPPA Bostan

OUTPUT

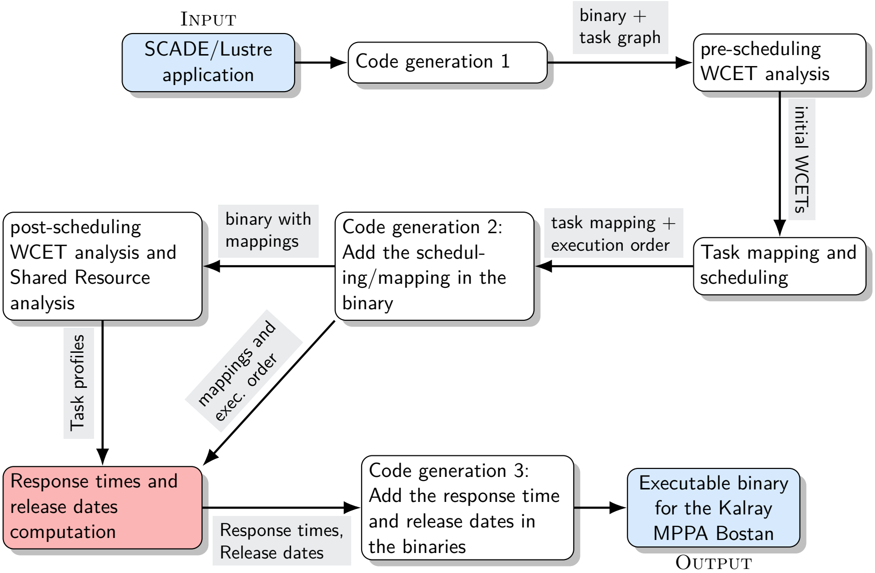
Evaluation: CAPACITES Project


INPUT
SCADE/Lustre application
Code generation 1
binary + task graph
pre-scheduling WCET analysis
initial WCETs
Task mapping and scheduling
task mapping + execution order
Code generation 2: Add the scheduling/mapping in the binary
binary with mappings
Executable binary for the Kalray MPPA Bostan
OUTPUT

Evaluation: CAPACITES Project


Traditional steps for single-cores
INPUT
SCADE/Lustre application
Code generation 1
binary + task graph
pre-scheduling WCET analysis
initial WCETs
Task mapping and scheduling
task mapping + execution order
Code generation 2: Add the scheduling/mapping in the binary
binary with mappings
Executable binary for the Kalray MPPA Bostan
OUTPUT

Evaluation: CAPACITES Project



INPUT

SCADE/Lustre application

Code generation 1

binary + task graph

pre-scheduling WCET analysis

initial WCETs

Task mapping and scheduling

task mapping + execution order

Code generation 2: Add the scheduling/mapping in the binary

binary with mappings

post-scheduling WCET analysis and Shared Resource analysis

Task profiles

mappings and exec. order

Response times and release dates computation

Response times, Release dates

Code generation 3: Add the response time and release dates in the binaries

Executable binary for the Kalray MPPA Bostan

OUTPUT

Outline: Many-Core Response Time Analysis

5 Implementation Choices of Synchronous Data Flow Programs

6 Multicore Response Time Analysis of SDF Programs

7 Target Many-Core: Kalray MPPA2

8 Evaluation

9 Summary and Future Work of Part II

Summary of Part II

- A response time analysis of synchronous data flow programs on the Kalray MPPA2

Summary of Part II

- A response time analysis of synchronous data flow programs on the Kalray MPPA2
- Given:
  - Task profiles: WCET in isolation and number of accesses
  - Mapping of Tasks
  - Execution Order

Summary of Part II

- A response time analysis of synchronous data flow programs on the Kalray MPPA2

- Given:
  - Task profiles: WCET in isolation and number of accesses
  - Mapping of Tasks
  - Execution Order

- We compute:
  - Tight response times taking into account the interference
  - Release dates respecting the dependency constraints

Summary of Part II

- A response time analysis of synchronous data flow programs on the Kalray MPPA2
- Given:
  - Task profiles: WCET in isolation and number of accesses
  - 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

Summary of Part II

- A response time analysis of synchronous data flow programs on the Kalray MPPA2

- Given:
  - Task profiles: WCET in isolation and number of accesses
  - 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

Summary of Part II

- A response time analysis of synchronous data flow programs on the Kalray MPPA2

- Given:
  - Task profiles: WCET in isolation and number of accesses
  - Mapping of Tasks
  - Execution Order

- We compute:
  - Tight response times taking into account the interference
  - Release dates respecting the dependency constraints

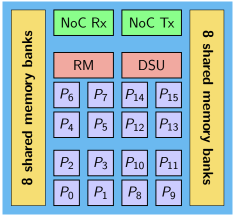
model of the multi-level arbiter

double fixed-point algorithm

- **Find in the manuscript:**
  - Execution phases: execution phase + communication phase
  - Support of: accesses pipelining, blocking and non-blocking accesses, bursts of accesses
  - More experiments with randomly generated task graphs

Future Work of Part II

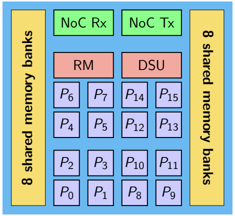
- Model of the Resource Manager
- Analysis with a Real-Time Operating System



Future Work of Part II

tighter estimation of context switches and other interrupts

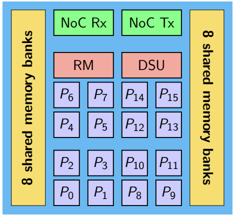
- Model of the Resource Manager
- Analysis with a Real-Time Operating System



Future Work of Part II

tighter estimation of context switches and other interrupts

- Model of the Resource Manager
- Analysis with a Real-Time Operating System
- Model of the NoC accesses.



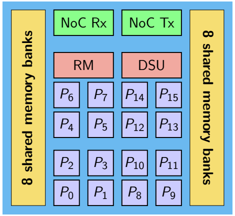
Future Work of Part II

tighter estimation of context switches and other interrupts

- Model of the Resource Manager
- Analysis with a Real-Time Operating System

use the output of any NoC analysis

- Model of the NoC accesses.
- Comparison between estimated and measured response times



Conclusion

Conclusion

- Multi/Many-cores in Real-Time Systems
  - Full isolation: for example TDMA
  - Bounded interference

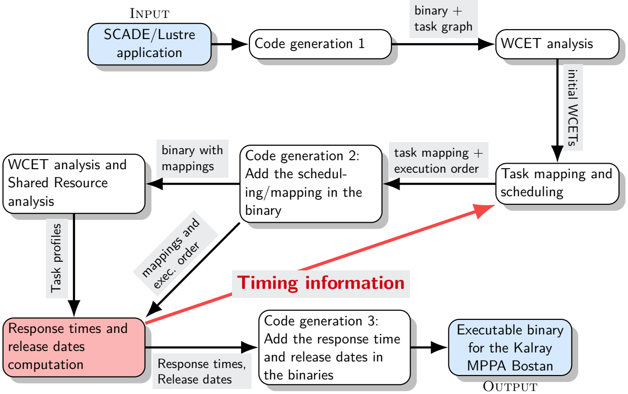
Conclusion

- Multi/Many-cores in Real-Time Systems
    - Full isolation: for example TDMA
    - Bounded interference
- Precise modeling of hardware components

Conclusion

- Multi/Many-cores in Real-Time Systems
  - Full isolation: for example TDMA
  - Bounded interference
- Precise modeling of hardware components
- Research directions for multi-core analysis:

Multi-core scheduling and timing analysis framework (in RTSOPS 2016)


INPUT
SCADE/Lustre application
Code generation 1
binary + task graph
WCET analysis
initial WCETs
Task mapping and scheduling
task mapping + execution order
Code generation 2: Add the scheduling/mapping in the binary
binary with mappings
WCET analysis and Shared Resource analysis
Task profiles
mappings and exec. order
Timing information
Response times and release dates computation
Response times, Release dates
Code generation 3: Add the response time and release dates in the binaries
Executable binary for the Kalray MPPA Bostan
OUTPUT

Many-Core Timing Analysis of Real-Time Systems

and its application to an industrial processor

Hamza Rihani

Université Grenoble Alpes / Verimag

Publications:

- Rihani, Hamza et al. (2015). "WCET analysis in shared resources real-time systems with TDMA buses". In: *Proceedings of the 23rd International Conference on Real-Time Networks and Systems*.
- Rihani, Hamza, Claire Maiza, and Matthieu Moy (2016a). "Efficient Execution of Dependent Tasks on Many-Core Processors". In: *RTSOPS 2016*. 7th International Real-Time Scheduling Open Problems Seminar. Toulouse, France.
- Rihani, Hamza et al. (2016b). "Response Time Analysis of Synchronous Data Flow Programs on a Many-Core Processor". In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems (RTNS)*.

This work is funded by grant CAPACITES (PIA-FSN2 n°P3425-146798) from the French *Ministère de l'économie, des finances et de l'industrie*.

References I

- Altmeyer, Sebastian et al. (2015). "A Generic and Compositional Framework for Multicore Response Time Analysis". In: *Proceedings of the 23rd International Conference on Real Time and Networks Systems (RTNS)*, pp. 129–138.

- Chattopadhyay, Sudipta, Abhik Roychoudhury, and Tulika Mitra (2010). "Modeling Shared Cache and Bus in Multicores for Timing Analysis". In: *Proceedings of the 13th International Workshop on Software and Compilers for Embedded Systems*. SCOPES '10. St. Goar, Germany: ACM, 6:1–6:10.

- Henry, Julien et al. (2014). "How to Compute Worst-case Execution Time by Optimization Modulo Theory and a Clever Encoding of Program Semantics". In: *Proceedings of the 2014 SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES)*, pp. 43–52.

- Kelter, Timon et al. (2014). "Static Analysis of Multi-core TDMA Resource Arbitration Delays". In: *Real-Time Syst.* 50.2, pp. 185–229.

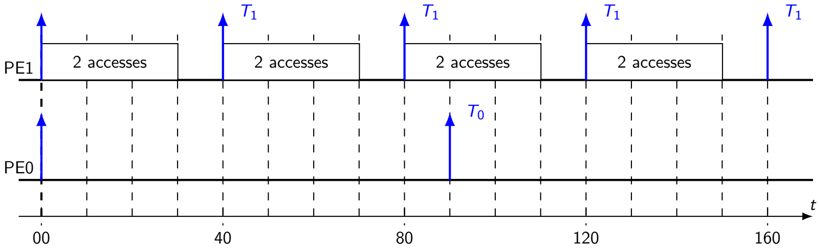
- Rosèn, Jacob et al. (2007). "Bus Access Optimization for Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip". In: *RTSS 2007*.

BACKUP

Multicore Response Time Analysis

Example: Fixed Priority bus arbiter, PE1 $>$ PE0

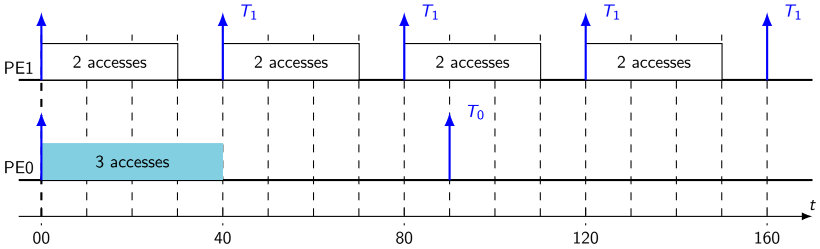
Bus access delay $= 10$



[1] Altmeyer et al., RTNS 2015

Multicore Response Time Analysis

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



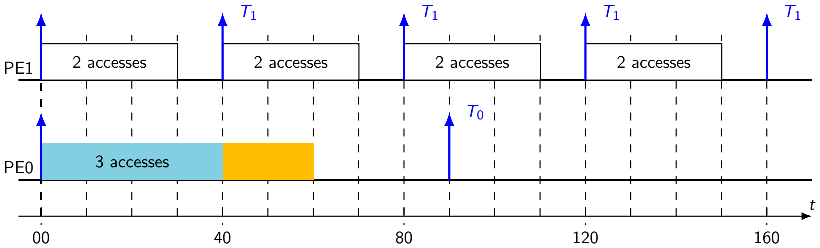
- Task of interest running on $PE0$:

  $R_0 = 10 + 3 \times 10$ (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)

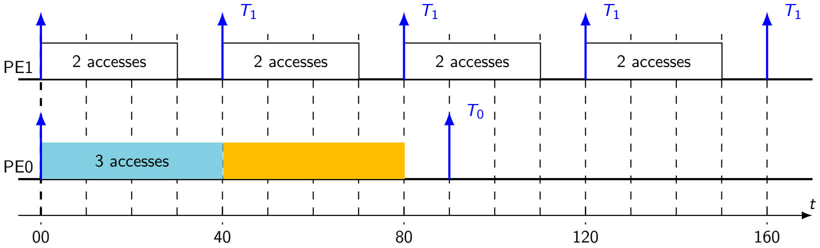
  $R_1 = 10 + 3 \times 10 + 2 \times 10 = 60$

[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)

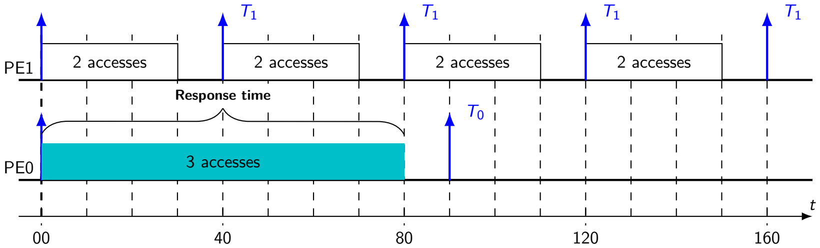
  $R_1 = 10 + 3 \times 10 + 2 \times 10 = 60$

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

[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)

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

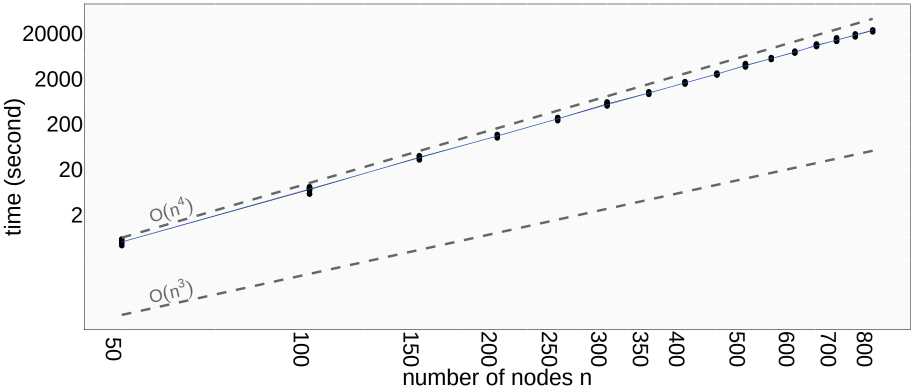
  $R_2 = 10 + 3 \times 10 + 2 \times 10 + 2 \times 10 = 80$

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

[1] Altmeyer et al., RTNS 2015

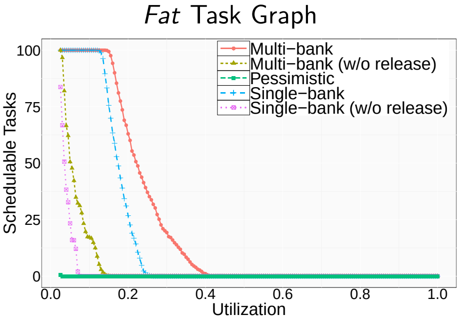
Evaluation: Runtime Perfomance

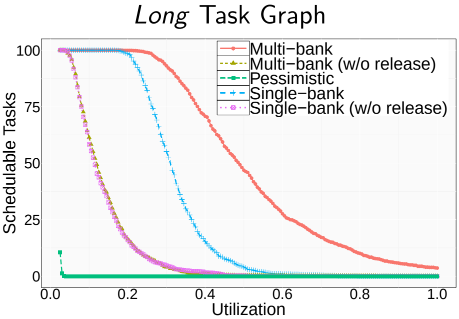
Analysis time of randomly generated task graphs in log-log scale



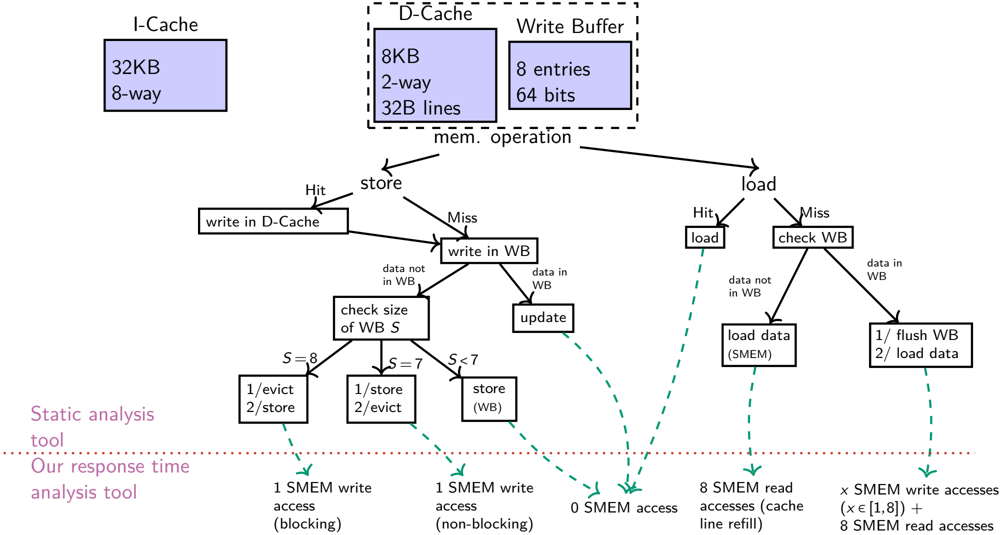
Theoretical complexity $\mathcal{O}(n^4)$. Experimental complexity $\mathcal{O}(n^{3.87})$.

Randomly Generated Task Graphs





Cached Memory Operations



I-Cache

32KB
8-way

D-Cache

8KB
2-way
32B lines

Write Buffer

8 entries
64 bits

mem. operation

store

Hit

write in D-Cache

Miss

write in WB

data not in WB

check size of WB S

$S = 8$

1/evict
2/store

$S = 7$

1/store
2/evict

$S < 7$

store (WB)

data in WB

update

load

Hit

load

Miss

check WB

data not in WB

load data (SMEM)

data in WB

1/ flush WB
2/ load data

Static analysis tool

Our response time analysis tool

1 SMEM write access (blocking)

1 SMEM write access (non-blocking)

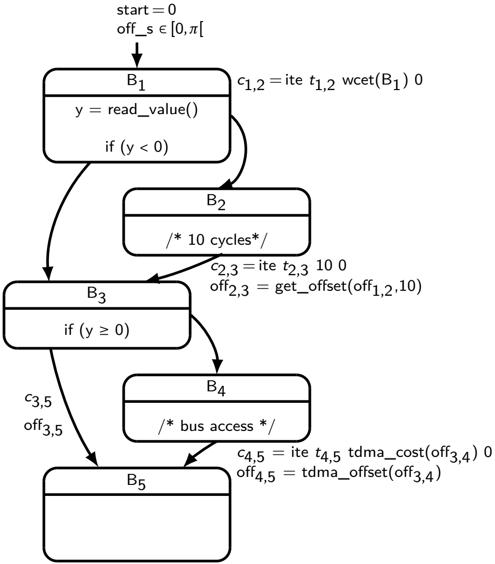
0 SMEM access

8 SMEM read accesses (cache line refill)

x SMEM write accesses ($x \in [1,8]$) +
8 SMEM read accesses

In isolation:
1 access = 10 cycles / 1 cache line refill = 17 cycles

Offset-based SMT Encoding



start $= 0$
off_s $\in [0, \pi[$

B_1
y = read_value()
if (y < 0)

$c_{1,2} =$ ite $t_{1,2}$ wcet(B_1) 0

B_2
/* 10 cycles*/

$c_{2,3} =$ ite $t_{2,3}$ 10 0
$\mathrm{off}_{2,3}$ = get_offset($\mathrm{off}_{1,2}$,10)

B_3
if (y ≥ 0)

$c_{3,5}$
$\mathrm{off}_{3,5}$

B_4
/* bus access */

$c_{4,5}$ = ite $t_{4,5}$ tdma_cost($\mathrm{off}_{3,4}$) 0
$\mathrm{off}_{4,5}$ = tdma_offset($\mathrm{off}_{3,4}$)

B_5

execution time $= \sum c_{i,j}$

- $\mathrm{off}_{i,j} = e_{i,j} \mod \pi$

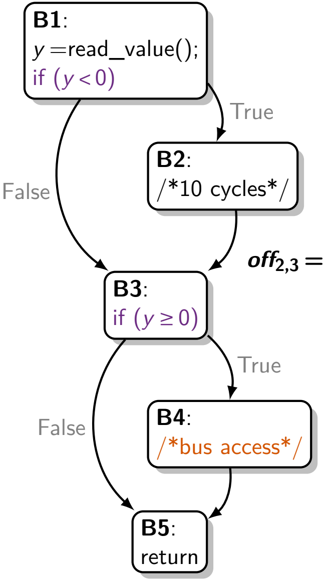
  Encode the costs of the basic blocks

  $e_{i,j}$ (absolute time) $\dashrightarrow$ $c_{i,j}$ (cost)

$c_{i,j}=$ ite $t_{i,j}$ $cost$ 0

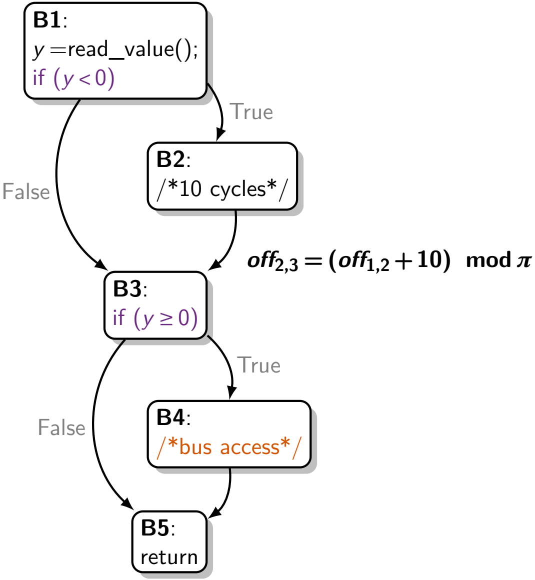
"ite C A B" ⇔ "if C then A else B"

Offset-based SMT Encoding



$$\boldsymbol{off_{2,3} = (off_{1,2} + 10) \mod \pi}$$

Offset-based SMT Encoding



B1:
$y =$read_value();
if $(y < 0)$

True

False

B2:
/*10 cycles*/

$\boldsymbol{off_{2,3} = (off_{1,2} + 10) \mod \pi}$

B3:
if $(y \geq 0)$

True

False

B4:
/*bus access*/

B5:
return

$$\overbrace{(\mathrm{off}_{i,j} + c)}^{<\pi} \quad \mathrm{mod}\ \pi$$

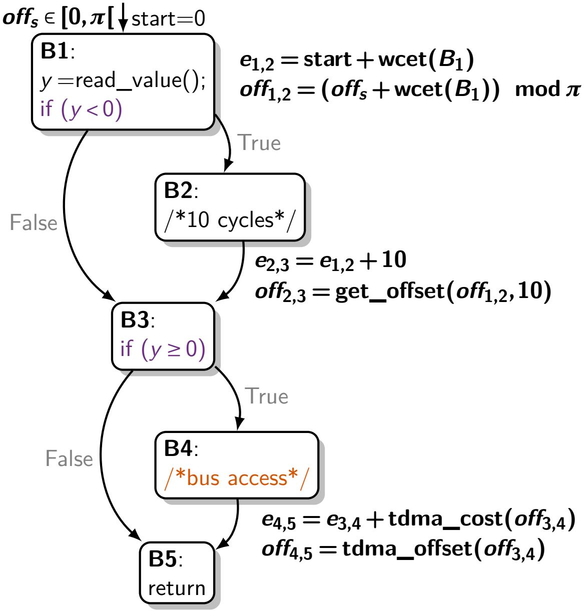
$$\Updownarrow$$

$$\overbrace{(\mathrm{off}_{i,j} + c \quad \mathrm{mod}\ \pi)}^{\overset{\text{def}}{=}\ \alpha\ <\ 2\pi}\ \underline{\mathrm{mod}}\ \pi$$

$$\downarrow$$

$$\text{if } \alpha < \pi \text{ then } \alpha \text{ else } \alpha - \pi$$

Offset-based SMT Encoding



$\boldsymbol{off_s \in [0, \pi[}$ start=0

B1:
y =read_value();
if ($y < 0$)

$$e_{1,2} = \text{start} + \text{wcet}(B_1)$$
$$off_{1,2} = (off_s + \text{wcet}(B_1)) \mod \pi$$

B2:
/*10 cycles*/

$$e_{2,3} = e_{1,2} + 10$$
$$off_{2,3} = \text{get\_offset}(off_{1,2}, 10)$$

B3:
if ($y \geq 0$)

B4:
/*bus access*/

$$e_{4,5} = e_{3,4} + \text{tdma\_cost}(off_{3,4})$$
$$off_{4,5} = \text{tdma\_offset}(off_{3,4})$$

B5:
return

execution time = if $t_{3,5}$ then $e_{3,5}$ else $e_{4,5}$

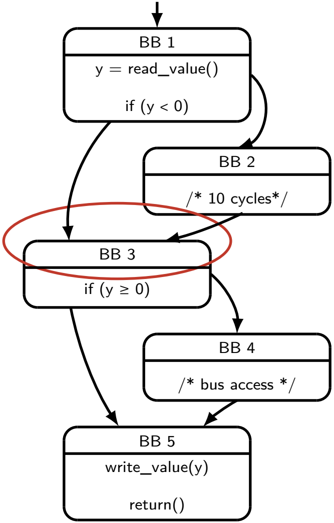
Using TDMA functions

- *if..then..else* encoding

  off $=$ *ite* t_{13} off_{13} off_{23}

  off_{35} $=$ off

  off_{34} $=$ off



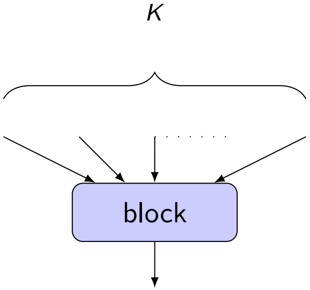
- *sum* encoding

  off $=$ off_{13} $+$ off_{23}

  off_{35} $=$ *ite* t_{35} off 0

  off_{34} $=$ *ite* t_{34} off 0

Using TDMA functions



if..then..else (ite) encoding:

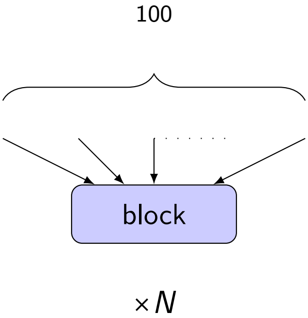
$$
\begin{aligned}
\mathit{off}_{i,j} = (&\textbf{if } t_{1,i} \textbf{ then } \mathit{off}_{1,i}\\
&\textbf{else if } t_{2,i} \textbf{ then } \mathit{off}_{2,i}\\
&\textbf{else } \ldots\\
&\textbf{else if } t_{K,i} \textbf{ then } \mathit{off}_{K,i} \textbf{ else } \mathit{0})
\end{aligned}
$$

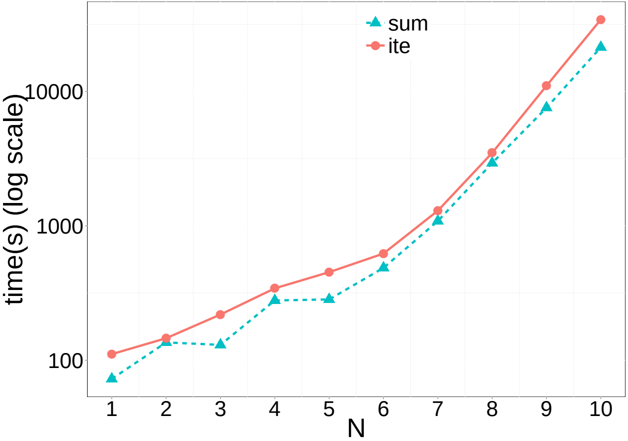
sum encoding:

$$
\mathit{off}_i = \sum_{k=1}^{k=K} \mathit{off}_{k,i}
$$

$$
\mathit{off}_{i,j} = \textbf{if } t_{i,j} \textbf{ then } \mathit{off}_i \textbf{ else } \mathit{0}
$$

Performance 3





How it works?

- Example with binary search:

```
Testing wcet >= 0... SAT (value found = 18).
                              New interval = [18, 73].
Testing wcet >= 46... UNSAT. New interval = [18, 45].
Testing wcet >= 32... UNSAT. New interval = [18, 31].
Testing wcet >= 25... UNSAT. New interval = [18, 24].
Testing wcet >= 21... UNSAT. New interval = [18, 20].
Testing wcet >= 19... UNSAT. New interval = [18, 18].
The maximum value of wcet is 18 .
Computation time is 0.010000s
```

Evaluation: Analysis Time

Analysis time

Name	$\pi = 40,\ \sigma = 20,\ acc = 10$	$\pi = 400,\ \sigma = 200,\ acc = 40$
bs	**0.45s**	0.80s
insertsort	1.37s	7.19s
jfdctint	44.10s	55.47s
fdct	41.36s	34.42s
compressdata	4.66s	3.44s
fly-by-wire	28.78s	**109.37s**