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

Matthieu Moy (from a presentation by Hamza Rihani)

Verimag (Grenoble INP)
Grenoble, France

June 2017
Outline

1. Critical, Real-Time and Many-Core
2. Parallel code generation and analysis
3. Models Definition
4. Multicore Response Time Analysis of SDF Programs
5. Evaluation
6. Conclusion and Future Work
Outline

1 Critical, Real-Time and Many-Core

2 Parallel code generation and analysis

3 Models Definition

4 Multicore Response Time Analysis of SDF Programs

5 Evaluation

6 Conclusion and Future Work
Time-critical, compute intensive applications

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

- Fast
- Predictable

- 68000
- PowerPC
- Many Core
- i7
- GPU
Many-core

= 

Lots of simple cores
Many-core = Lots of simple cores

Kalray MPPA (Massively Parallel Processor Array):

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

⇒ good fit for real-time?
Kalray’s business model

Explore Kalray’s markets

Storage Solutions
> Learn More

Embedded Systems
> Learn More

Press releases
June 25 Kalray unveils its certified intelligent NVMe-oF solutions with server and storage leader AIC at ISC 2018
June 7 Kalray has raised €12.7M, the most significant IPO since Eurovent Growth was created in Paris
May 25 Kalray’s intelligent processor soon to go public
May 22 Kalray first to receive NVMe-oF certification for a fully integrated system
May 17 Kalray announces its IPO on Paris Eurovent Growth Stock Market
May 2 Alliance Ventures (Renault-Nissan-Mitsubishi) and Definist acquire stakes in Kalray

Upcoming Events
7 AUG FLASH MEMORY SUMMIT 2018
17 SEP AUTOSENS BRUSSELS 2018
25 SEP HFPC 18
12 NOV SC18 SuperComputing 2018

Latest tweets
2 Jul
@KalrayRetweeted 
EricBeausis, PDG de @KalrayInc et ex incubé de l’@IncubatPacaEst, vient partager avec les créateurs d’entreprise de la @CotedAzur son expérience d’entrepreneur et de levée de fonds. Belle introduction en bourse en juin: 44M€ pour poursuivre le dép. de leur superordinateur !

27 Jan
Just after being launched (November 2017), Definist (managed by @Alliance on behalf of the French Ministry of the Armed Forces) has acquired stakes in Kalray. #Kalray
https://t.co/BNvqQRqHa7
Hard Real-Time on Many-Core

High-level Data-Flow Application Model

Synchronous hypothesis:
computation/communication in 0-time
Hard Real-Time on Many-Core

High-level Data-Flow Application Model

Synchronous hypothesis:
computation/communication in 0-time

Take into account all levels in Worst-Case Execution Time (W CET) analysis and programming model
Synchronous hypothesis: computation/communication in 0-time

High-level Data-Flow Application Model

Take into account all levels in Worst-Case Execution Time (W CET) analysis and programming model
Hard Real-Time on Many-Core

High-level Data-Flow Application Model

Synchronous hypothesis:
computation/communication in 0-time

Network On Chip
Communication takes time

Shared Memory within Cluster
Interferences between tasks

Individual Cores
Cache, Pipeline, . . .
Hard Real-Time on Many-Core

High-level Data-Flow Application Model

Synchronous hypothesis:
computation/communication in 0-time

Network On Chip
Communication takes time

Shared Memory within Cluster
Interferences between tasks

Individual Cores
Cache, Pipeline, . . .

→ Take into account all levels
in Worst-Case Execution Time (WCET) analysis
and programming model
Context and Partners

CIFRE (Graillat)
CIFRE (Lo)

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

point-to-point

Projet CAPACITES (Ph.D Rihani)
Outline

1. Critical, Real-Time and Many-Core
2. Parallel code generation and analysis
3. Models Definition
4. Multicore Response Time Analysis of SDF Programs
5. Evaluation
6. Conclusion and Future Work
Execution of Synchronous Data Flow Programs

High level representation

Single-core code generation

Industrialized as SCADE (1993)
heavily used in avionics and nuclear
Execution of Synchronous Data Flow Programs

Multi/Many-core code generation

static non-preemptive scheduling

High level representation
Execution of Synchronous Data Flow Programs

High level representation

- Respect the dependency constraints

Multi/Many-core code generation

static non-preemptive scheduling

PE0

PE1

PE2

\( \tau_0 \)

\( \tau_1 \)

\( \tau_2 \)

\( \tau_3 \)

\( \tau_4 \)

\( \tau_5 \)

\( i_0 \)

\( i_1 \)
Execution of Synchronous Data Flow Programs

High level representation

Multi/Many-core code generation

static non-preemptive scheduling

✓ Respect the dependency constraints
✓ Set the release dates to get precise upper bounds on the interference
// Generated by SCADE KCG

void NA(ctx_a *ctx) {
    // ... computation ...
}

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

// Generated by us

void worker_PEO(void) {
    ctx_a ctxa; ctx_b ctxb;
    while (1) {
        NA_wrapper(&ctxa);
        wait(release_t2);
        NB_wrapper(&ctxb);
        wait(end_of_period);
    }
}

#define RECV_NA(data) ...
1. Precise accounting for interference on shared resources in a many-core processor
Contributions (part of Ph.D Hamza Rihani, with Claire Maiza)

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

2. Model of a multi-level arbiter to the shared memory
Contributions (part of Ph.D Hamza Rihani, with Claire Maiza)

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

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

3. Response time and release dates analysis respecting dependencies.
Outline

1. Critical, Real-Time and Many-Core
2. Parallel code generation and analysis
3. Models Definition
4. Multicore Response Time Analysis of SDF Programs
5. Evaluation
6. Conclusion and Future Work
Architecture Model

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

- Static non-preemptive scheduling
- Spatial Isolation
  - different tasks go to different memory banks
- Interference from communications

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

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

- Directed Acyclic Task Graph
- Mono-rate
- Fixed mapping and execution order
- Each task $\tau_i$:

- Input: Processor Demand, Memory Demand
- Output: Release date ($rel_i$), response time ($R_i$)
Application Model

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

![Diagram of directed acyclic task graph]

- Processor Demand
- Memory access time

Wait time $\omega$
Application Model

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

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

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

Find $R_i$ (including the interference)
Find $rel_i$ respecting precedence constraints
Outline

1  Critical, Real-Time and Many-Core
2  Parallel code generation and analysis
3  Models Definition
4  Multicore Response Time Analysis of SDF Programs
5  Evaluation
6  Conclusion and Future Work
Response Time Analysis

\[ R = PD + I_{BUS}(R) \]

- Response Time

\[ I_{BUS}(R) = \sum_{b \in B} I_{BUS_b}(R) \]

where \( B \): a set of memory banks

Requires a model of the bus arbiter.
Response Time Analysis

\[ R = PD + I^{BUS}(R) \]

- Response Time
  - Processor Demand

- Requires a model of the bus arbiter

\( I^{BUS}(R) \): Interference from bus interference

\( I_{PROC}(R) \): Interference from preempting tasks

\( I_{DRAM}(R) \): Interference from DRAM refreshes
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 preemtping 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 preemtping tasks
      (no preemption: \( I_{PROC} = 0 \))
    - Interference from DRAM refreshes
      (out of scope. \( I_{DRAM} = 0 \))

- Fix-point formula \( \Rightarrow \) iterative 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. \( I_{DRAM} = 0 \))

- Fix-point formula \( \Rightarrow \) iterative algorithm.
- Multiple shared resources (memory banks)
Response Time Analysis

\[ R = PD + I^{BUS}(R) + I^{PROC}(R) + I^{DRAM}(R) \]

- Response Time
  - Processor Demand
    - Bus Interference
      * (given a model of the bus arbiter)
    - Interference from preempting tasks
      * (no preemption: \( I^{PROC} = 0 \))
    - Interference from DRAM refreshes
      * (out of scope. \( I^{DRAM} = 0 \))

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

\[ I^{BUS}(R) = \sum_{b \in B} I^{BUS}_b(R) \]

where \( B \): a set of memory banks
Response Time Analysis

\[ R = PD + I_{BUS}(R) + I_{PROC}(R) + I_{DRAM}(R) \]

- Response Time
  - Processor Demand
    - Bus Interference
      (given a model of the bus arbiter)
    - Interference from preempting tasks
      (no preemption: \( I_{PROC} = 0 \))
    - Interference from DRAM refreshes
      (out of scope. \( I_{DRAM} = 0 \))
- Fix-point formula \( \Rightarrow \) iterative algorithm.
- Multiple shared resources (memory banks)

\[ I_{BUS}(R) = \sum_{b \in B} I_{BUS}^b(R) \]

where \( B \): a set of memory banks

Requires a model of the bus arbiter
Model of the MPPA Bus

\[ l_b^{\text{BUS}}: \text{delay from all accesses + concurrent ones} \]
Model of the MPPA Bus

- $I_{b}^{\text{BUS}}$: delay from all accesses + concurrent ones
- $S_{i}^{b}$: number of accesses of task $\tau_{i}$ to bank $b$
Model of the MPPA Bus

$P_0$  

$P_0$  

$L_{v1} = S_i^b$  

$L_{v2} = L_{v1} + \sum_{y=1}^{15} \min(\tau_i^{y,b}, L_{v1})$  

$I_{BUS}^b$: delay from all accesses + concurrent ones

$S_i^b$: number of accesses of task $\tau_i$ to bank $b$

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

\[ I_b \]: delay from all accesses + concurrent ones
\[ S_i^b \]: number of accesses of task \( \tau_i \) to bank \( b \)
\[ A_{i}^{y,b} \]: number of concurrent accesses from core \( y \) to bank \( b \)

\[ L_{v1} = S_i^b \]
\[ L_{v2} = L_{v1} + \sum_{y=1}^{15} \min( A_{i}^{y,b} , L_{v1} ) \]
\[ L_{v3} = L_{v2} + \min( A_{i}^{G2,b} , L_{v2} ) \]
Model of the MPPA Bus

\[ I_b \text{BUS} : \text{delay from all accesses} + \text{concurrent ones} \]

\[ S^b_i : \text{number of accesses of task } \tau_i \text{ to bank } b \]

\[ A^y,b_i : \text{number of concurrent accesses from core } y \text{ to bank } b \]

\[
L_{v1} = S^b_i \\
L_{v2} = L_{v1} + \sum_{y=1}^{15} \min( A^{y,b}_i, L_{v1} ) \\
L_{v3} = L_{v2} + \min( A^{G2,b}_i, L_{v2} ) \\
L_{v4} = L_{v4} + A^{G3,b}_i
\]
Model of the MPPA Bus

$I_{BUS}^b$: delay from all accesses + concurrent ones

$S_i^b$: number of accesses of task $\tau_i$ to bank $b$

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

$L_{V1} = S_i^b$

$L_{V2} = L_{V1} + \sum_{y=1}^{15} \min( A_{i}^{y,b} , L_{V1} )$

$L_{V3} = L_{V2} + \min( A_{i}^{G2,b} , L_{V2} )$

$L_{V4} = L_{V4} + A_{i}^{G3,b}$

$I_{BUS}^b = L_{V4} \times \text{Bus Delay}$
Model of the MPPA Bus

\[ I_{b}^{BUS} : \text{delay from all accesses + concurrent ones} \]

\[ S_{i}^{b} : \text{number of accesses of task } \tau_{i} \text{ to bank } b \]

\[ A_{i}^{y,b} : \text{number of concurrent accesses from core } y \text{ to bank } b \]

\[ L_{v1} = S_{i}^{b} \]

\[ L_{v2} = L_{v1} + \sum_{y=1}^{15} \min(A_{i}^{y,b}, L_{v1}) \]

\[ L_{v3} = L_{v2} + \min(A_{i}^{G2,b}, L_{v2}) \]

\[ L_{v4} = L_{v3} + A_{i}^{G3,b} \]

\[ I_{b}^{BUS} = L_{v4} \times \text{Bus Delay} \]
Model of the MPPA Bus

\[ I_{BUS}^b = \text{Delay from all accesses + concurrent ones} \]

\[ S_i^b: \text{number of accesses of task } \tau_i \text{ to bank } b \]

\[ A_{i}^{y,b}: \text{number of concurrent accesses from core } y \text{ to bank } b \]

\[ L_{V1} = S_i^b \]

\[ L_{V2} = L_{V1} + \sum_{y=1}^{15} \min(A_{i}^{y,b}, L_{V1}) \]

\[ L_{V3} = L_{V2} + \min(A_{i}^{G2,b}, L_{V2}) \]

\[ L_{V4} = L_{V4} + A_{i}^{G3,b} \]

\[ I_{BUS}^b = L_{V4} \times \text{Bus Delay} \]

\[ A_{i}^{y,b} \text{ depends on } \text{rel}_i \text{ and } R_i \]
Response Time Analysis with Dependencies

1. Start with initial release dates.

```
1 Initial rel

WCRT analysis
for all i do
R_{i+1}^{l} \leftarrow \text{PD}_{i} + \text{BUS}(R_{i}^{l}, rel_{i})
end for
```
Response Time Analysis with Dependencies

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

WCRT analysis

\[
\text{for all } i \text{ do} \\
R_i^{i+1} \leftarrow \text{PD}_i + \text{BUS}(R_i^i, rel_i) \\
\text{end for}
\]

Initial rel\(_i^0 \neq R_i^i \)
1 Start with initial release dates.
2 Compute response times

... ...

WCRT analysis

for all $i$
do

$R_{i+1}^{l+1} \leftarrow PD_i + I_{BUS}(R_i^l, rel_i)$

end for
Response Time Analysis with Dependencies

1. Start with initial release dates.
2. Compute response times
   ... ... ... a fixed-point is reached!

\[
\begin{align*}
\text{WCRT analysis} & \quad \text{for all } i \text{ do} \\
& \quad R_{i+1}^{t+1} \leftarrow \text{PD}_i + I \text{BUS}(R_i^t, rel_i) \\
& \text{end for}
\end{align*}
\]
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

\text{for all } i \text{ do}
\begin{align*}
R_{i+1} &\leftarrow \text{PD}_{i+1} + \text{BUS}(R_i, rel_i) \\
\end{align*}
\text{end for}

Update release dates

\text{for all } i \text{ do}
rel_i &\leftarrow \text{latest finish time of all the dependencies} \\
\end{align*}
\text{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

\[ R_{i}^{l+1} \leftarrow PD_{i} + BUS(R_{i}^{l}, rel_{i}) \]

end for

Initial $rel_{i}^{0}$

new $rel_{i}$ repeat

Update release dates

for all $i$ do

\[ rel_{i} \leftarrow \text{latest finish time of all the dependencies} \]

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

\[
\text{for all } i \text{ do } \quad \text{PD}_{i+1} \leftarrow \text{BUS}(R_i, \text{rel}_i) \\
\text{end for}
\]

Update release dates

\[
\text{for all } i \text{ do } \quad \text{rel}_i \leftarrow \text{latest finish time of all the dependencies} \\
\text{end for}
\]

Return: \((\text{rel}, R_i)\)
Proof of Convergence Toward a Fixed-point

- Convergence of the 1st fixed-point iteration:

```
PE2
  τ4  τ5
PE1
  τ3
PE0
  τ0  τ1  τ2
```

```
WCRT analysis
for all i do
  Ri+1 ← PD_i + BUS(R_i, rel_i)
end for
```

```
Update release dates
for all i do
  rel_i ← latest finish time of all the dependencies
end for
```

```
return: (rel_i, R_i)
```

```
initial rel_i^0
```

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

```
new rel_i repeat
```

```
rel_i did not change
```

Return: (rel_i, R_i)
Proof of Convergence Toward a Fixed-point

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

WCRT analysis

\[
\text{for all } i \text{ do } \quad R_{i}^{l+1} \leftarrow PD_{i} + BUS(R_{i}^{l}, rel_{i})
\]
end for

Update release dates

\[
\text{for all } i \text{ do } \quad rel_{i} \leftarrow \text{latest finish time of all the dependencies}
\]
end for

Return: \((rel_{i}, R_{i})\)
Proof of Convergence Toward a Fixed-point

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

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

```
WCRT analysis
for all i do
    R_{i+1} ← PD_i + BUS(R_i, rel_i)
end for
```

- Update release dates

```
for all i do
    R_i ← new rel_i
end for
```

Initial rel_i^0

\[ R_{i+1}^{l+1} = PD_i + BUS(R_i^l, rel_i) \]

\[ R_i \]

\[ rel_i \] did not change

Return: (rel_i, R_i)

\[ R_i^{l+1} \neq R_i^l \]

\[ \tau_0 \]

\[ \tau_1 \]

\[ \tau_2 \]

\[ \tau_3 \]

\[ \tau_4 \]

\[ \tau_5 \]
Proof of Convergence Toward a Fixed-point

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

WCRT analysis

\begin{align*}
\text{for all } i \text{ do} \\
R_{i+1}^{l+1} &\leftarrow PD_i + \text{BUS}(R_i^l, rel_i) \\
\text{end for}
\end{align*}

Update release dates

\begin{align*}
\text{for all } i \text{ do} \\
rel_i &\leftarrow \text{latest finish time of all the dependencies} \\
\text{end for}
\end{align*}

Return: $(rel_i, R_i)$
Proof of Convergence Toward a Fixed-point

- Convergence of the 1st fixed-point iteration:
  - Monotonic and bounded

- Convergence of the 2nd fixed-point iteration:
  - no monotonicity: $R_i$ and $rel_i$ may grow or shrink at each iteration.

**Theorem**

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

Full proof in our technical report:

http://www-verimag.imag.fr/TR/TR-2016-1.pdf
Proof of Convergence Toward a Fixed-point

- Convergence of the 1\textsuperscript{st} fixed-point iteration:
  - Monotonic and bounded \(\checkmark\)
- Convergence of the 2\textsuperscript{nd} fixed-point iteration:
  - no monotonicity: \(R_i\) and \(rel_i\) may grow or shrink at each iteration.

**Theorem**

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

Full proof in our technical report:

http://www-verimag.imag.fr/TR/TR-2016-1.pdf
Proof of Convergence Toward a Fixed-point

- Convergence of the 1\textsuperscript{st} fixed-point iteration:
  - Monotonic and bounded

- Convergence of the 2\textsuperscript{nd} fixed-point iteration:
  - No monotonicity: $R_i$ and $rel_i$ may grow or shrink at each iteration.

Theorem

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

Full proof in our technical report:

http://www-verimag.imag.fr/TR/TR-2016-1.pdf
Proof of Convergence Toward a Fixed-point Iteration

- Convergence of the 1\textsuperscript{st} fixed-point iteration:
  - Monotonic and bounded

- Convergence of the 2\textsuperscript{nd} fixed-point iteration:
  - No monotonicity: \( R_i \) and \( rel_i \) may grow or shrink at each iteration.

\textbf{Theorem}

\textit{At each iteration, at least one task finds its final release date.}

\textbf{Full proof in our technical report:}

[http://www-verimag.imag.fr/TR/TR-2016-1.pdf](http://www-verimag.imag.fr/TR/TR-2016-1.pdf)
Proof of Convergence Toward a Fixed-point

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

Theorem

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

Full proof in our technical report:
http://www-verimag.imag.fr/TR/TR-2016-1.pdf
Proof of Convergence Toward a Fixed-point

- Convergence of the 1st fixed-point iteration:
  - Monotonic and bounded
- Convergence of the 2nd 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](http://www-verimag.imag.fr/TR/TR-2016-1.pdf)
Outline

1. Critical, Real-Time and Many-Core
2. Parallel code generation and analysis
3. Models Definition
4. Multicore Response Time Analysis of SDF Programs
5. Evaluation
6. Conclusion and Future Work
Evaluation: ROSACE Case Study

- Flight management system controller

---

1 Pagetti et al., RTAS 2014
Evaluation: ROSACE Case Study

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

---

1 Pagetti et al., RTAS 2014
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
• 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

<table>
<thead>
<tr>
<th>Task</th>
<th>Processor Demand (cycles)</th>
<th>Memory Demand (accesses)</th>
</tr>
</thead>
<tbody>
<tr>
<td>altitude</td>
<td>275</td>
<td>22</td>
</tr>
<tr>
<td>az_filter</td>
<td>274</td>
<td>22</td>
</tr>
<tr>
<td>h_filter</td>
<td>326</td>
<td>24</td>
</tr>
<tr>
<td>va_control</td>
<td>303</td>
<td>24</td>
</tr>
<tr>
<td>va_filter</td>
<td>301</td>
<td>23</td>
</tr>
<tr>
<td>vz_control</td>
<td>320</td>
<td>25</td>
</tr>
<tr>
<td>vz_filter</td>
<td>334</td>
<td>25</td>
</tr>
</tbody>
</table>

Table: Task profiles of the FMS controller

- Profile obtained from measurements
Evaluation: ROSACE Case Study

<table>
<thead>
<tr>
<th>Task</th>
<th>Processor Demand (cycles)</th>
<th>Memory Demand (accesses)</th>
</tr>
</thead>
<tbody>
<tr>
<td>altitude</td>
<td>275</td>
<td>22</td>
</tr>
<tr>
<td>az_filter</td>
<td>274</td>
<td>22</td>
</tr>
<tr>
<td>h_filter</td>
<td>326</td>
<td>24</td>
</tr>
<tr>
<td>va_control</td>
<td>303</td>
<td>24</td>
</tr>
<tr>
<td>va_filter</td>
<td>301</td>
<td>23</td>
</tr>
<tr>
<td>vz_control</td>
<td>320</td>
<td>25</td>
</tr>
<tr>
<td>vz_filter</td>
<td>334</td>
<td>25</td>
</tr>
</tbody>
</table>

Table: Task profiles of the FMS controller

- Profile obtained from measurements
- Memory Demand: data and instruction cache misses + communications
Evaluation: ROSACE Case Study

<table>
<thead>
<tr>
<th>Task</th>
<th>Processor Demand (cycles)</th>
<th>Memory Demand (accesses)</th>
</tr>
</thead>
<tbody>
<tr>
<td>altitude</td>
<td>275</td>
<td>22</td>
</tr>
<tr>
<td>az_filter</td>
<td>274</td>
<td>22</td>
</tr>
<tr>
<td>h_filter</td>
<td>326</td>
<td>24</td>
</tr>
<tr>
<td>va_control</td>
<td>303</td>
<td>24</td>
</tr>
<tr>
<td>va_filter</td>
<td>301</td>
<td>23</td>
</tr>
<tr>
<td>vz_control</td>
<td>320</td>
<td>25</td>
</tr>
<tr>
<td>vz_filter</td>
<td>334</td>
<td>25</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Task</th>
<th>Processor Demand (cycles)</th>
<th>Memory Demand (accesses)</th>
</tr>
</thead>
<tbody>
<tr>
<td>altitude</td>
<td>275</td>
<td>22</td>
</tr>
<tr>
<td>az_filter</td>
<td>274</td>
<td>22</td>
</tr>
<tr>
<td>h_filter</td>
<td>326</td>
<td>24</td>
</tr>
<tr>
<td>va_control</td>
<td>303</td>
<td>24</td>
</tr>
<tr>
<td>va_filter</td>
<td>301</td>
<td>23</td>
</tr>
<tr>
<td>vz_control</td>
<td>320</td>
<td>25</td>
</tr>
<tr>
<td>vz_filter</td>
<td>334</td>
<td>25</td>
</tr>
</tbody>
</table>

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

**Bus Policy**
- E5: Pessimistic
- E4: 1-Phase (w/o release)
- E3: 2-Phase (w/o release)
- E2: 1-Phase
- E1: 2-Phase

**Smallest schedulable hyper-period**
Evaluation: Experiments

E5: E5: Pessimistic
E4: E4: 1-Phase (w/o release)
E3: E3: 2-Phase (w/o release)
E2: E2: 1-Phase
E1: E1: 2-Phase

Smallest schedulable hyper-period
Evaluation: Experiments

Smallest schedulable hyper-period

- E5: All accesses interfere
- E4, E3: We don’t use the release dates
Evaluation: Experiments

Smallest schedulable hyper-period

- E5: All accesses interfere
- E4, E3: We don’t use the release dates
- E2, E1: Our approach. We use the release dates
Evaluation: Experiments

1 bank

5 banks

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

Phases are modeled as sub-tasks
Evaluation: Experiments

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

<table>
<thead>
<tr>
<th>Processor cycles</th>
<th>1 bank</th>
<th>5 banks</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPPA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bus Policy
- E5: Pessimistic
- E4: 1−Phase (w/o release)
- E3: 2−Phase (w/o release)
- E2: 1−Phase
- E1: 2−Phase

Smallest schedulable hyper-period

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MPPA</td>
<td>4.15</td>
<td>4.12</td>
<td>1.68</td>
<td>1.29</td>
<td>~1.01</td>
<td>0.77</td>
</tr>
<tr>
<td>RR</td>
<td>3.3</td>
<td>3.29</td>
<td>1.24</td>
<td>1.13</td>
<td>~1.01</td>
<td>0.91</td>
</tr>
</tbody>
</table>
Taking into account the memory banks improves the analysis with a factor in $[1.77, 2.52]$.
Taking into account the memory banks improves the analysis with a factor in $[1.77, 2.52]$.

<table>
<thead>
<tr>
<th>Bus Policy</th>
<th>Processor cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>E5: Pessimistic</td>
<td>16000</td>
</tr>
<tr>
<td>E4: 1-Phase (w/o release)</td>
<td>12000</td>
</tr>
<tr>
<td>E3: 2-Phase (w/o release)</td>
<td>8000</td>
</tr>
<tr>
<td>E2: 1-Phase</td>
<td>4000</td>
</tr>
<tr>
<td>E1: 2-Phase</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MPPA</td>
<td>4.15</td>
<td>4.12</td>
<td>1.68</td>
<td>1.29</td>
<td>~1.01</td>
<td>0.77</td>
</tr>
<tr>
<td>RR</td>
<td>3.3</td>
<td>3.29</td>
<td>1.24</td>
<td>1.13</td>
<td>~1.01</td>
<td>0.91</td>
</tr>
</tbody>
</table>

Speedup factors

Smallest schedulable hyper-period
Evaluation: Experiments

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

<table>
<thead>
<tr>
<th>Bus Policy</th>
<th>Processor cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>E5: Pessimistic</td>
<td>16000</td>
</tr>
<tr>
<td>E4: 1-Phase (w/o release)</td>
<td>12000</td>
</tr>
<tr>
<td>E3: 2-Phase (w/o release)</td>
<td>8000</td>
</tr>
<tr>
<td>E2: 1-Phase</td>
<td>4000</td>
</tr>
<tr>
<td>E1: 2-Phase</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Smallest schedulable hyper-period</th>
</tr>
</thead>
<tbody>
<tr>
<td>E5/E1</td>
</tr>
<tr>
<td>E5/E2</td>
</tr>
<tr>
<td>E3/E1</td>
</tr>
<tr>
<td>E4/E2</td>
</tr>
<tr>
<td>E2/E1</td>
</tr>
<tr>
<td>E4/E3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Artifact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consistent</td>
</tr>
<tr>
<td>Evaluated</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Speedup factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPPA</td>
</tr>
<tr>
<td>RR</td>
</tr>
</tbody>
</table>
Taking into account the memory banks improves the analysis with a factor in \([1.77, 2.52]\)
Evaluation: Experiments

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

<table>
<thead>
<tr>
<th>Bus Policy</th>
<th>Processor cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPPA</td>
<td>1 bank</td>
</tr>
<tr>
<td>RR</td>
<td>5 banks</td>
</tr>
<tr>
<td>E5: Pessimistic</td>
<td>9460</td>
</tr>
<tr>
<td>E4: 1−Phase (w/o release)</td>
<td>3312</td>
</tr>
<tr>
<td>E3: 2−Phase (w/o release)</td>
<td>9310</td>
</tr>
<tr>
<td>E2: 1−Phase</td>
<td>3318</td>
</tr>
<tr>
<td>E1: 2−Phase</td>
<td>8528</td>
</tr>
</tbody>
</table>

Smallest schedulable hyper-period

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MPPA</td>
<td>4.15</td>
<td>4.12</td>
<td>1.68</td>
<td>1.29</td>
<td>~1.01</td>
<td>0.77</td>
</tr>
<tr>
<td>RR</td>
<td>3.3</td>
<td>3.29</td>
<td>1.24</td>
<td>1.13</td>
<td>~1.01</td>
<td>0.91</td>
</tr>
</tbody>
</table>
Outline

1 Critical, Real-Time and Many-Core
2 Parallel code generation and analysis
3 Models Definition
4 Multicore Response Time Analysis of SDF Programs
5 Evaluation
6 Conclusion and Future Work
Conclusion

- Code generation and real-time analysis for many-core (Kalray MPPA 256) = major challenge for industry and research
- Hard Real-Time ⇒ simplicity, predictability ⇒ static, time-driven schedule
- Critical ⇒ traceability ⇒ no aggressive optimization
- Our work:
  - Understand and model the precise architecture of MPPA
  - Extension of Multi-Core Response Time Analysis
  - Non-trivial proof of termination
Future Work

- Model of the Resource Manager.

Questions?
Future Work

- Model of the Resource Manager.
  - Tighter estimation of context switches and other interrupts

<table>
<thead>
<tr>
<th>8 shared memory banks</th>
<th>8 shared memory banks</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoC Rx</td>
<td>NoC Tx</td>
</tr>
<tr>
<td>RM</td>
<td>DSU</td>
</tr>
<tr>
<td>P_8</td>
<td>P_9</td>
</tr>
<tr>
<td>P_7</td>
<td>P_10</td>
</tr>
<tr>
<td>P_6</td>
<td>P_11</td>
</tr>
<tr>
<td>P_5</td>
<td>P_12</td>
</tr>
<tr>
<td>P_4</td>
<td>P_13</td>
</tr>
<tr>
<td>P_3</td>
<td></td>
</tr>
<tr>
<td>P_2</td>
<td></td>
</tr>
<tr>
<td>P_1</td>
<td></td>
</tr>
<tr>
<td>P_0</td>
<td></td>
</tr>
</tbody>
</table>
Future Work

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

![Diagram showing resource management and network on chip (NoC) components with nodes labeled P0 to P15, highlighted areas for NoC Rx, NoC Tx, RM, and DSU, and 8 shared memory banks labeled P0 to P7, P8 to P15.]

Tighter estimation of context switches and other interrupts.
Future Work

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

 Tighter estimation of context switches and other interrupts

Use the output of any NoC analysis

Questions?
Future Work

- Model of the Resource Manager.
- Model of the NoC accesses.
- Memory access pipelining.
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.
- Model of the NoC accesses.
- Memory access pipelining.
- Model Blocking and non-blocking accesses.
Future Work

- Model of the Resource Manager.
- Model of the NoC accesses.
- Memory access pipelining.
- Model Blocking and non-blocking accesses.

- Tighter estimation of context switches and other interrupts.
- Use the output of any NoC analysis.
- Current assumption: bus delay is 10 cycles.
- Reads are blocking, writes are non-blocking.
Future Work

- Model of the Resource Manager.
- Model of the NoC accesses.
- Memory access pipelining.
- Model Blocking and non-blocking accesses.

Questions?
BACKUP
Multicore Response Time Analysis

Example: Fixed Priority bus arbiter, PE1 > PE0

Bus access delay = 10

\[ T_0 \times 2 \text{ accesses} \]

\[ T_1 \times 2 \text{ accesses} \]

\[ \text{Response time} \]

\[ ^1 \text{Altmeyer et al., RTNS 2015} \]
### Multicore Response Time Analysis

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

*Bus access delay = 10*

![Time Chart]

- Task of interest running on *PE0*:
  
  \[ R_0 = 10 + 3 \times 10 \text{ (response time in isolation)} \]

---

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)
\[ R_1 = 10 + 3 \times 10 + 2 \times 10 = 60 \]

\(^1\text{Altmeyer et al., RTNS 2015}\)
Multicore Response Time Analysis

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

○ Task of interest running on PE0:
  
  \[ R_0 = 10 + 3 \times 10 \text{ (response time in isolation)} \]

  \[ R_1 = 10 + 3 \times 10 + 2 \times 10 = 60 \]

  \[ R_2 = 10 + 3 \times 10 + 2 \times 10 + 2 \times 10 = 80 \]

---

\(^1\) Altmeyer et al., RTNS 2015
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)
- Static Mapping/Scheduling
- Local WCRT Analysis
- Probabilistic Models
- Tasks WCRT + WC Access
- Tasks + Dependencies
- Static Mapping/Scheduling
- Mapping
- Execution Order
- Release Dates
- WCRT with Interferences
- Binary Generation
- Executable Binary