Taking fragile communication means into account for stream programs

Stéphane Louise, X-K Do, Thierry Goubier, Paul Dubrule

CEA, LIST—January 24th, 2017

stephane.louise@cea.fr

Outline

1. Context
   - Stream programming and parallel distributed systems
   - Stream programming, principles and models
   - Graph model, properties and optimization

2. The case of fragile communication links
   - Scheduling approach, case of (C)SDF
   - A new model for stream programs: TPDF

3. Conclusion

Context

Parallel distributed embedded applications cumulate several issues:

- Parallelism
- Communication between sub-parts / sub-systems
- Possibly mixed criticality
- Heterogeneity of resources

Context

Parallel distributed embedded applications cumulate several issues:

- Parallelism
- Communication between sub-parts / sub-systems
- Possibly mixed criticality
- Heterogeneity of resources

Stream programming solves some of these issues:

- Simple programming model
- Parallelism and synchronization points are obvious both to the programmer and to the compiler
- Compilation tools can offer powerful optimization scenarios
- Most of broadly used stream programming models are deterministic and are well fit to DSP and video processing applications
- amenable to real-time constraints

Stream programming overview

The base is a directed graph (A, E, T)

- A a set of Actors
- E a set of directed edges in $A \times A$
- T a set of communication tokens

Stream programming overview

The base is a directed graph (A, E, T)

- A a set of Actors
- E a set of directed edges in $A \times A$
- T a set of communication tokens

A stands for processing parts, E stands for communication channels (FIFO style) and T for data elements that can be transmitted atomically.

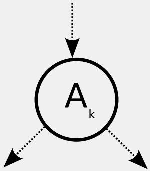
The base is a directed graph (A, E, T)

- A a set of Actors
- E a set of directed edges in $A \times A$
- T a set of communication tokens

A stands for processing parts, E stands for communication channels (FIFO style) and T for data elements that can be transmitted atomically.

Actors

Actor A_k



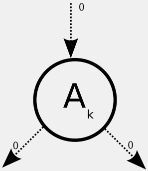
Stream programming overview

The base is a directed graph (A, E, T)

- A a set of Actors
- E a set of directed edges in $A \times A$
- T a set of communication tokens

A stands for processing parts, E stands for communication channels (FIFO style) and T for data elements that can be transmitted atomically.

Actors



Actor A_k

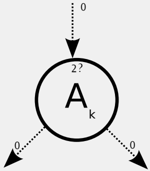
Stream programming overview

The base is a directed graph (A, E, T)

- A a set of Actors
- E a set of directed edges in $A \times A$
- T a set of communication tokens

A stands for processing parts, E stands for communication channels (FIFO style) and T for data elements that can be transmitted atomically.

Actors



Actor A_k

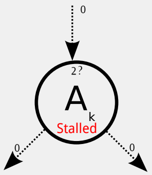
Stream programming overview

The base is a directed graph (A, E, T)

- A a set of Actors
- E a set of directed edges in $A \times A$
- T a set of communication tokens

A stands for processing parts, E stands for communication channels (FIFO style) and T for data elements that can be transmitted atomically.

Actors



Actor A_k

Stream programming overview

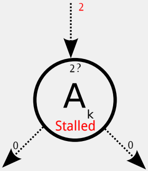
The base is a directed graph (A, E, T)

- A a set of Actors
- E a set of directed edges in $A \times A$
- T a set of communication tokens

A stands for processing parts, E stands for communication channels (FIFO style) and T for data elements that can be transmitted atomically.

Actors

Actor A_k



Stream programming overview

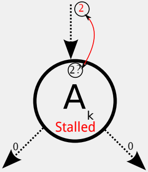
The base is a directed graph (A, E, T)

- A a set of Actors
- E a set of directed edges in $A \times A$
- T a set of communication tokens

A stands for processing parts, E stands for communication channels (FIFO style) and T for data elements that can be transmitted atomically.

Actors

Actor A_k



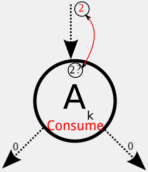
Stream programming overview

The base is a directed graph (A, E, T)

- A a set of Actors
- E a set of directed edges in $A \times A$
- T a set of communication tokens

A stands for processing parts, E stands for communication channels (FIFO style) and T for data elements that can be transmitted atomically.

Actors



Actor A_k

Stream programming overview

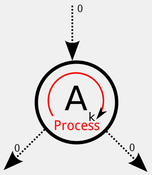
The base is a directed graph (A, E, T)

- A a set of Actors
- E a set of directed edges in $A \times A$
- T a set of communication tokens

A stands for processing parts, E stands for communication channels (FIFO style) and T for data elements that can be transmitted atomically.

Actors

Actor A_k



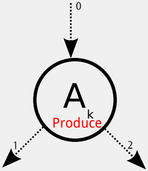
Stream programming overview

The base is a directed graph (A, E, T)

- A a set of Actors
- E a set of directed edges in $A \times A$
- T a set of communication tokens

A stands for processing parts, E stands for communication channels (FIFO style) and T for data elements that can be transmitted atomically.

Actors



Actor A_k

Stream programming overview

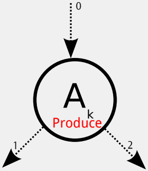
The base is a directed graph (A, E, T)

- A a set of Actors
- E a set of directed edges in $A \times A$
- T a set of communication tokens

A stands for processing parts, E stands for communication channels (FIFO style) and T for data elements that can be transmitted atomically.

Actors

Actor A_k



Locally deterministic

Models of computations (MoCs)

Several Models of Computation are available:

- KPN: firing rules can change freely according to the program encapsulated within the actor
- HDF: firing rules are static allowing for production and consumption of a single data token for each link
- CSDF: static firing rules are triggered according to a fixed size cycle
- SADF: commutations of firing rules according to a fixed set of scenarios
- BDF: control link allow for switching between branches
- etc.

Analizability of the MoC depends on the model. The nicest ones provide guaranties against deadlocks, livelocks and inconsistencies.

The assets of a graph representation

The graph structure allows for:

- A hierarchical definition of an application
- A set of graph transformations: coarsening, some rules of permutations or rewriting (i.e. optimization)
- A graphical representation showing in an obvious way where are the performance pitfalls
- Partitioning: good for heterogeneous computing

The assets of a graph representation

The graph structure allows for:

- A hierarchical definition of an application
- A set of graph transformations: coarsening, some rules of permutations or rewriting (i.e. optimization)
- A graphical representation showing in an obvious way where are the performance pitfalls
- Partitioning: good for heterogeneous computing

SDF and CSDF models are often used when safety is important:

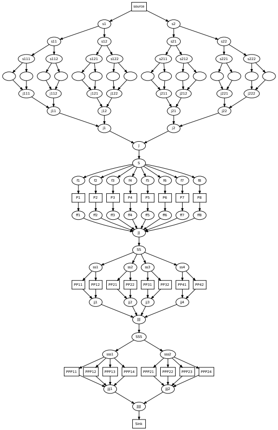
- Deadlock and livelock free
- Amenable to real-time constraints (RT scheduling)
- Some extensions aim to extend the expressiveness of the models

Graph optimizations

Some actors are defined to simply distribute, copy or gather data values without doing further processing

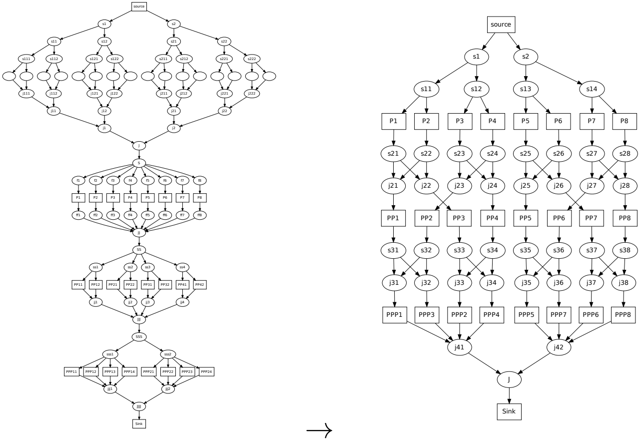
Graph optimizations

Some actors are defined to simply distribute, copy or gather data values without doing further processing



Graph optimizations

Some actors are defined to simply distribute, copy or gather data values without doing further processing



Stream programming and heterogeneous environments

Usual assumptions of stream programming:

- Actors have WCETs
- Communication links are robust and preserve order

Stream programming and heterogeneous environments

Usual assumptions of stream programming:

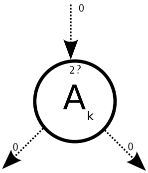
- Actors have WCETs
- Communication links are robust and preserve order
- Underlying hypothesis: the application runs on a single (MC) processor and I/O rates are well known and ensured by the HW

Stream programming and heterogeneous environments

Usual assumptions of stream programming:

- Actors have WCETs
- Communication links are robust and preserve order
- Underlying hypothesis: the application runs on a single (MC) processor and I/O rates are well known and ensured by the HW

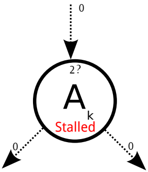
  The future of embedded system is thought to be more and more connected:
  - IoT and Home automation
  - Autonomous vehicles (e.g. traffic aware navigation)
  - Smart grids
  - Sensor networks, etc.



Stream programming and heterogeneous environments

Usual assumptions of stream programming:

- Actors have WCETs
- Communication links are robust and preserve order
- Underlying hypothesis: the application runs on a single (MC) processor and I/O rates are well known and ensured by the HW
- The future of embedded system is thought to be more and more connected:
  - IoT and Home automation
  - Autonomous vehicles (e.g. traffic aware navigation)
  - Smart grids
  - Sensor networks, etc.

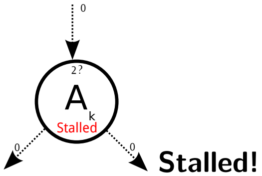


Stream programming and heterogeneous environments

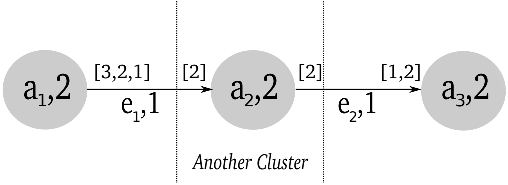
Usual assumptions of stream programming:

- Actors have WCETs
- Communication links are robust and preserve order
- Underlying hypothesis: the application runs on a single (MC) processor and I/O rates are well known and ensured by the HW

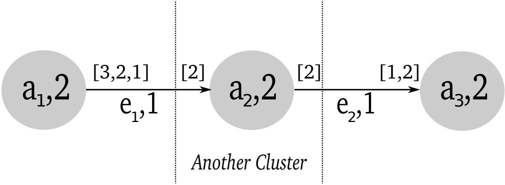
  The future of embedded system is thought to be more and more connected:
  - IoT and Home automation
  - Autonomous vehicles (e.g. traffic aware navigation)
  - Smart grids
  - Sensor networks, etc.



A scheduling approach for distributed systems



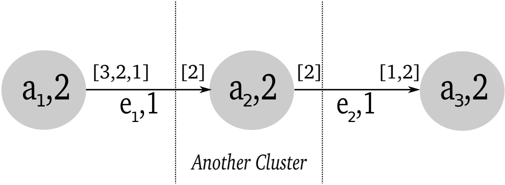
A scheduling approach for distributed systems



First idea: token matching with time stamping

a_1	a_2	a_3
1	1	1
2	2	2 3
3	3	4

A scheduling approach for distributed systems



First idea: token matching with time stamping

a_1	a_2	a_3
1	1	1
2	2	2
		3
3	3	4

For CSDF there exist 2 minimal repetition vectors:

$$\vec{r} = [1, 3, 2] \quad \text{and} \quad \vec{q} = [3, 3, 4]$$

These provide basis of all well-formed scheduling policies

Building RT schedule for CSDF

$$\Gamma \cdot \overrightarrow{r} = 0, \tag{1}$$

where Γ is the *topology matrix* of G.

Building RT schedule for CSDF

$$\Gamma \cdot \overrightarrow{r} = 0, \quad (1)$$

where Γ is the *topology matrix* of G.
For $\overrightarrow{q}$ multiply by the length of cycles.

Building RT schedule for CSDF

$$\Gamma \cdot \overrightarrow{r} = 0, \qquad (1)$$

where Γ is the *topology matrix* of G.
For $\overrightarrow{q}$ multiply by the length of cycles.

Period vector:

$$\lambda_i^{min} = \frac{Q}{q_i}\left\lceil \frac{\eta}{Q} \right\rceil \quad for \;\; a_i \in A, \qquad (2)$$

where $\eta = \max_{a_i \in A}(\omega_i q_i)$ and $Q = lcm(q_1, q_2, \ldots, q_n)$

Building RT schedule for CSDF

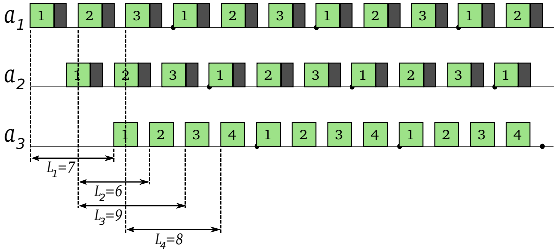
$$\Gamma \cdot \overrightarrow{r} = 0, \qquad (1)$$

where Γ is the *topology matrix* of G.
For $\overrightarrow{q}$ multiply by the length of cycles.

Period vector:

$$\lambda_i^{min} = \frac{Q}{q_i} \left\lceil \frac{\eta}{Q} \right\rceil \quad for \;\; a_i \in A, \qquad (2)$$

where $\eta = \max_{a_i \in A}(\omega_i q_i)$ and $Q = lcm(q_1, q_2, \ldots, q_n)$



Actor Dependence Function (ADF)

$$ADF_{A \leftarrow B}(n) = \underset{\phi \in \Phi}{maxList}(|\phi \wedge A(n)| \wedge \{B\})$$

where ϕ is an ordered sequence of actor firings of a dataflow graph and Φ denotes the set of all legal schedules.

Actor Dependence Function (ADF)

$$ADF_{A \leftarrow B}(n) = \underset{\phi \in \Phi}{maxList}(|\phi \wedge A(n)| \wedge \{B\})$$

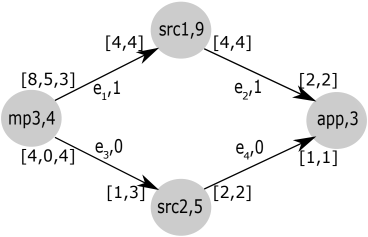
where ϕ is an ordered sequence of actor firings of a dataflow graph and Φ denotes the set of all legal schedules.

Definition of a "Late schedule": fire Actors as late as possible

Late schedule formulation

//Late Schedule: X predecessors are fired as late as possible
lateSchedule(X,n) {
$\phi = \{\}$
for $i = 1$ to n **do**
    for all input channels c_i of X **do**
        while X need more tokens on c_i in order to fire **do**
            //extend schedule ($\oplus$ denotes concatenation)
            $\phi = \phi \oplus lateSchedule(source(c_i, 1)$
        end while
    end for
    //add X to schedule
    $\phi = \phi \oplus X$
    //update number of tokens on I/O channels of X
    simulateExecution(X)
end for
return ϕ }

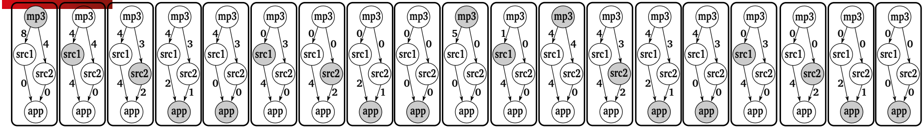
Example



And the data dependence:

mp3	src1	src2	app
1	1	1	1
			2
	2	2	3
			4
2	3		
3		3	5
			6
	4	4	7
			8

Scheduling and Actor Dependence Function (ADF)



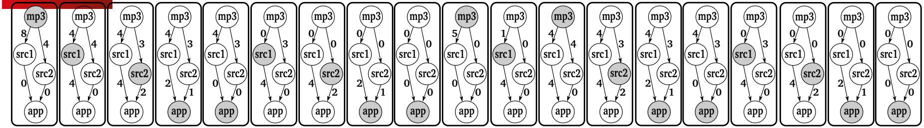
mp3	src1	src2	app	app	src1	src2	app	app	mp3	src1	mp3	src2	app	app	src1	src2	app	app
1	1	1	1	2	2	2	3	4	2	3	3	3	5	6	4	4	7	8

Count the ordinal number of executions for each actor in Late schedule

$ADF_{mp3 \leftarrow app}(1)=\{1,2,3,4\}$ $ADF_{mp3 \leftarrow app}(2)=\{\}$ $ADF_{mp3 \leftarrow app}(3)=\{5,6,7,8\}$

$ADF_{src2 \leftarrow app}(1)=\{1,2\}$ $ADF_{src2 \leftarrow app}(2)=\{3,4\}$ $ADF_{src2 \leftarrow app}(3)=\{5,6\}$ $ADF_{src2 \leftarrow app}(4)=\{7,8\}$

Scheduling and Actor Dependence Function (ADF)



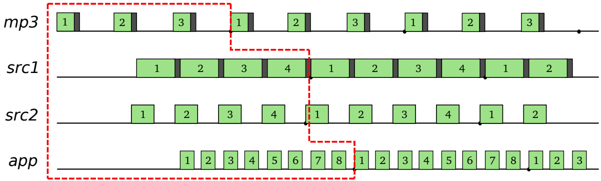
mp3	src1	src2	app	app	src1	src2	app	app	mp3	src1	mp3	src2	app	app	src1	src2	app	app

Count the ordinal number of executions for each actor in Late schedule

1	1	1	1	2	2	2	3	4	2	3	3	3	5	6	4	4	7	8

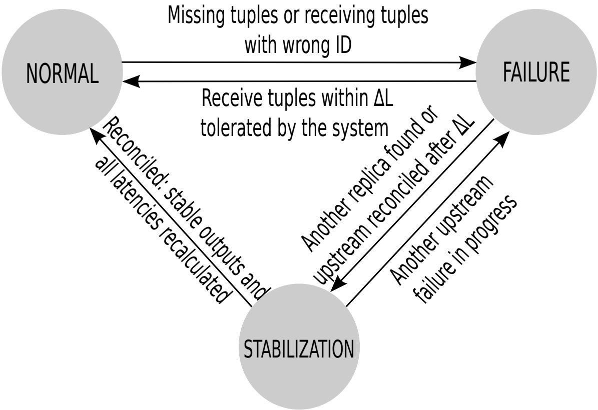
$ADF_{mp3 \leftarrow app}(1)=\{1,2,3,4\}$ $ADF_{mp3 \leftarrow app}(2)=\{\}$ $ADF_{mp3 \leftarrow app}(3)=\{5,6,7,8\}$

$ADF_{src2 \leftarrow app}(1)=\{1,2\}$ $ADF_{src2 \leftarrow app}(2)=\{3,4\}$ $ADF_{src2 \leftarrow app}(3)=\{5,6\}$ $ADF_{src2 \leftarrow app}(4)=\{7,8\}$



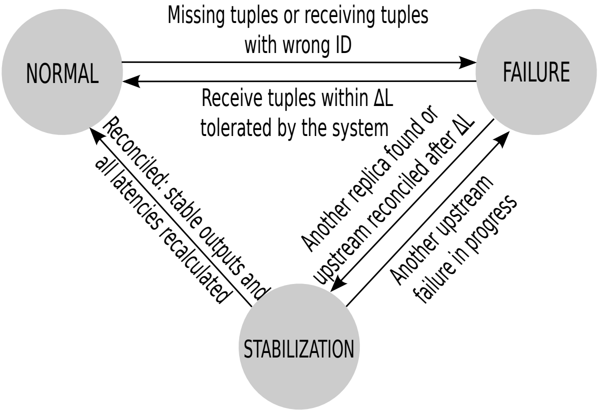
Fragile communication and fault tolerance

Model of fault tolerance within CSDF:



Fragile communication and fault tolerance

Model of fault tolerance within CSDF:

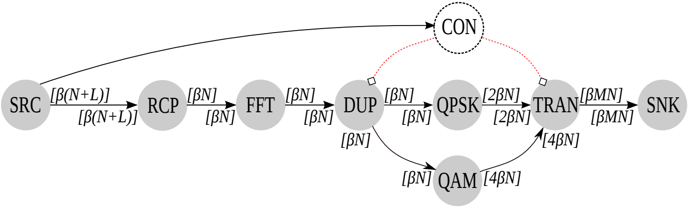


Alternatively: Modify the MoC to take uncertainties into account → TPDF

Transaction Parametrized Data-Flow

Was introduced to take into account 2 use cases:

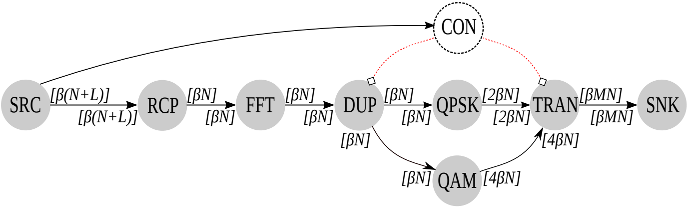
- Dynamic rates in embedded applications: e.g. SW radio



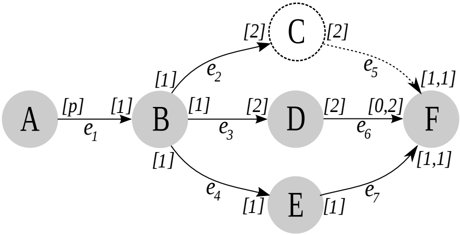
Transaction Parametrized Data-Flow

Was introduced to take into account 2 use cases:

- Dynamic rates in embedded applications: e.g. SW radio



- Multiple path with fragile timing:



Properties of TPDF

TPDF is:

- globally deterministic
- RT aware
- Amenable to fault-tolerance

On going work, but well-fitted to Mixed Criticality systems

Conclusion

Stream programming is a good model for embedded and distributed applications

- Deadlock, livelock, race-condition free
- Amenable to RT constraints
- Lots of optimization and transformations on graph constructs

But requires improvements:

- For fragile communications
- For scheduling

Conclusion

Stream programming is a good model for embedded and distributed applications

- Deadlock, livelock, race-condition free
- Amenable to RT constraints
- Lots of optimization and transformations on graph constructs

But requires improvements:

- For fragile communications
- For scheduling

Future work:

- Improve the model: TPDF as one possible way
- Parameters
- Checkpoints ("Transaction") with control actors to check timing and change the execution state of the application

Thanks!

Centre de Saclay
Nano-Innov PC 172 - 91191 Gif sur Yvette Cedex