Synchronous Modeling of Data-Intensive Applications


{Yu, Gamatie, Boulet, Dekeyser}@lifl.fr
Eric.Rutten@inrialpes.fr
DART project, INRIA Futurs / WEST group, LIFL
This talk:

- Detailed presentation of ARRAY-OL/GASPARD
- First results of study
Plan

Introduction
  GASPARD methodology
  General scheme

Data-intensive processing
  ARRAY-OL language
  Existing works

Simple synchronous modeling of GASPARD models
  Parallel model
  Serialized model

Validation issues

Conclusions
**Introduction**

**Context**: data-intensive applications (DIA) in embedded systems
- regular multidimensional data processing
- parallel processing in System-on-Chip (SoC)

**Motivations**: adequate techniques for
- efficient data manipulation
- analysis of implementation properties

**Approach**: combination of
- a formalism dedicated to DIA (ARRAY-OL)
- data-flow synchronous equation models
GASPARD methodology

- Array-OL pretty editors
- Synchronous Technologies

Interop Bridge

Refactoring

Appli PIM

Architectural PIM

Association PIM

High Perf PSM

Corba PSM

SystemC PSM

Fortran code

Corba code

SystemC code

produce

TLM PIMs

RTL PIM

VHDL PSM

Verilog PSM

SystemC code

Verilog files

VHDL files

Performance Analysis
General scheme

codesign transformations compilation simulation TLM, RTL

+ control

Gaspard2 (Array-OL metamodel) transf1 synchronous equations (metamodel)

+ control

transf2

diagnosis, debug

analysis verification
clock calculus simulation

code generators

Lustre
Lucid Synchrone
Signal

modeling language

intermediate format

formal specification language
Introduction

Gaspard methodology
General scheme

Data-intensive processing

Array-OL language
Existing works

Simple synchronous modeling of Gaspard models

Parallel model
Serialized model

Validation issues

Conclusions
**Array-OL**

**Array-OL** (Array-Oriented Language) : initially proposed by Thomson Marconi Sonar [DD98]

- Specification language for full parallelism
- Data manipulation through arrays
- Deadlock free and deterministic by construction
Array-OL (Array-Oriented Language) : initially proposed by Thomson Marconi Sonar [DD98]

- Specification language for full parallelism
- Data manipulation through arrays
- Deadlock free and deterministic by construction
- Descriptions independent from implementation platforms
- Two types of parallelism in application specifications: Task parallelism and Data parallelism
Task parallelism and data dependencies:

Task1
(1920, 1080, ∞) → (720, 1080, ∞)

Task2
(1600, 1200, ∞) → (720, 1080, ∞)

Task3
(720, 1080, ∞) → (720, 480, ∞)
Different task models:

- Elementary task: atomic computation block (instantaneous function)
- Hierarchical task: task represented by hierarchical acyclic graphs in which each node consists of a task, and edges are labeled by the arrays
- Repetition task: expression of data parallelism
Different task models:

- **Elementary task**: atomic computation block (instantaneous function)
- **Hierarchical task**: task represented by hierarchical acyclic graphs in which
  - each node consists of a task, and
  - edges are labeled by the arrays
Different task models:

- **Elementary task**: atomic computation block (instantaneous function)
- **Hierarchical task**: task represented by hierarchical acyclic graphs in which
  - each node consists of a task, and
  - edges are labeled by the arrays
- **Repetition task**: expression of data parallelism
Data parallelism

- **Repetition element**: the subtask to be repeated
- **Repetition space**: limitation of repetition number and link between inputs and outputs
- **Interface**: input and output arrays
- **Tiler**: defines how to obtain sub-arrays from an input array and how to store sub-arrays in an output array
Example of a repetition task

$R(3, 2)$

$E(1)$

$F = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

$o = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

$P = \begin{pmatrix} 3 & 0 \\ 0 & 4 \end{pmatrix}$

$F = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

$o = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

$P = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$

$F = \begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix}$

$o = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

$P = \begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix}$
Tiler specification:

- $o$: original point of the array or reference pattern
- $P$: paving matrix (how the array is tiled by patterns)
- $F$: fitting matrix (how patterns are filled by array elements)
**Paving**: how the array is tiled by patterns.

These patterns are calculated in any order.

**Paving example**: \( o = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad P = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix} \)

**Repetition space**: \([5,4]\), limitation of pattern repetitions.
Paving: how the array is tiled by patterns.

These patterns are calculated in any order.

Paving example: \( o = (0, 0), \ P = (2, 0, 0, 3) \)

Repetition space: [5,4], limitation of pattern repetitions.
**Array-OL (cont’d)**

**Paving**: how the array is tiled by patterns.

These patterns are calculated in **any order**.

**Paving example**: \( \mathbf{o} = (0,0), \mathbf{P} = (2,0,0,3) \)

**Repetition space**: \([5,4]\), limitation of pattern repetitions.
**Array-OL (cont’d)**

**Paving**: how the array is tiled by patterns. These patterns are calculated in **any order**.

**Paving example**: \( o = (0, 0), P = (2, 0, 0, 3) \)

**Repetition space**: \([5, 4]\), limitation of pattern repetitions.
**Fitting**: how each pattern is filled by array elements.

**Fitting example 1**: \( o = (0), \quad F = \left( \frac{1}{3} \right) \)
**Array-OL (cont’d)**

**Fitting**: how each pattern is filled by array elements.

**Fitting example 1**: $o = (0)$, $F = \left( \frac{1}{3} \right)$
**Fitting**: how each pattern is filled by array elements.

**Fitting example 1**: \( o = (0), \ F = (\frac{1}{3}) \)
Fitting: how each pattern is filled by array elements.

Fitting example 1: \( o = (0), \ F = \left( \frac{1}{3} \right) \)
**Array-OL (cont’d)**

**Fitting**: how each pattern is filled by array elements.

**Fitting example 1**: \( o = (0), \quad F = \left( \frac{1}{3} \right) \)
**Fitting**: how each pattern is filled by array elements.

**Fitting example 1**: \( o = (0), \ F = (\frac{1}{3}) \)
**Array-OL (cont’d)**

**Fitting**: how each pattern is filled by array elements.

**Fitting example 1**: \( o = (0), \quad F = (\frac{1}{3}) \)

![Diagram of Array 1 with pattern 0 1 2 3 4 5 6 7 8 9 10]

**Fitting example 2**: \( o = (0), \quad F = (\frac{2}{6}) \)

![Diagram of Array 2 with pattern 0 1 2 3 4 5 6 7 8 9 10]
**Array-OL (cont’d)**

**Fitting**: how each pattern is filled by array elements.

**Fitting example 1**: \( o = (0), \ F = (\frac{1}{3}) \)

![Array 1 Pattern](image)

**Fitting example 2**: \( o = (0), \ F = (\frac{2}{6}) \)

![Array 2 Pattern](image)
**Fitting** : how each pattern is filled by array elements.

**Fitting example 1** : \( o = (0), \; F = (\frac{1}{3}) \)

![Array 1 Pattern](image)

**Fitting example 2** : \( o = (0), \; F = (\frac{2}{6}) \)

![Array 2 Pattern](image)
**Array-OL (cont’d)**

**Fitting**: how each pattern is filled by array elements.

**Fitting example 1**: \( o = (0), \ F = (\frac{1}{3}) \)

---

**Array 1**

---

**Fitting example 2**: \( o = (0), \ F = (\frac{2}{6}) \)

---

**Array 2**
**Fitting**: how each pattern is filled by array elements.

**Fitting example 1**: \( \mathbf{o} = (0), \ F = (\frac{1}{3}) \)

**Fitting example 2**: \( \mathbf{o} = (0), \ F = (\frac{2}{6}) \)
**Fitting**: how each pattern is filled by array elements.

**Fitting example 1**: \( o = (0), \quad F = (\frac{1}{3}) \)

![Array 1](image)

**Fitting example 2**: \( o = (0), \quad F = (\frac{2}{6}) \)

![Array 2](image)
An example of repetition task: array product

\[ O = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \]
\[ P = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \]
\[ F = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \]

\[ O = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \]
\[ P = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \]
\[ F = \begin{pmatrix} \_ \end{pmatrix} \]

Number of instances: \(2 \times 3 = 6\); Repetition point: \([0,0]\)
An example of repetition task: array product

\[ O = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \]
\[ P = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \]
\[ F = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \]

Number of instances: \( 2 \times 3 = 6 \); Repetition point: \([0,0]\)
An example of repetition task: array product

$$O = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$P = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

$$F = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Number of instances: 2*3 = 6; Repetition point: [0,0]
Array-OL (cont’d)

An example of repetition task: array product

\[ O = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \]
\[ P = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \]
\[ F = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \]

Number of instances: \( 2 \times 3 = 6 \); Repetition point: \([0,0]\)
An example of repetition task: array product

Number of instances: \(2 \times 3 = 6\); Repetition point: \([0,0]\)
An example of repetition task: array product

\[ O = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \]
\[ P = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \]
\[ F = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \]

Number of instances: \(2 \times 3 = 6\); Repetition point: \([0,0]\)
An example of repetition task: array product

\[ O = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \]
\[ P = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \]
\[ F = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \]

Number of instances: \(2 \times 3 = 6\); Repetition point: \([0, 1]\)
An example of repetition task: array product

\[ O = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \]
\[ P = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \]
\[ F = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \]

Number of instances: \( 2 \times 3 = 6 \); Repetition point: [1, 0]
Existing works

**ALPHA** language [Mauras, 1989] *(vs. ARRAY-OL)*

- multidimensional data structures for data-intensive applications
- union of convex polyhedra *(vs. arrays)*
- data access through indices calculated by affine functions *(vs. hierarchical and modular pattern)*
- absence of modulo *(vs. presence of modulo)*
Introduction

GASPARD methodology
General scheme

Data-intensive processing

ARRAY-OL language
Existing works

Simple synchronous modeling of GASPARD models

Parallel model
Serialized model

Validation issues

Conclusions
Synchronous modeling of Gaspard models

Illustration of the modeling by the following example:

\[
\begin{align*}
R &= (3, 2) \\
E &= (1) \\
F &= (1 0 0 1) \\
o &= (0 0) \\
P &= (3 0 0 4) \\
F &= (1 0) \\
o &= (0 0) \\
P &= (0 0 0 1) \\
F &= (2 0 0 1) \\
o &= (0 0) \\
P &= (3 0 0 3)
\end{align*}
\]
Parallel model

Modeling of repetition task

\[ \forall j \in r, \ A_3[< \text{ind}_3^j >] := E(A_1[< \text{ind}_1^j >], A_2[< \text{ind}_2^j >]) \]

- \( j \) : a point in the repetition space \( r \)
- \( < \text{ind}_i^j > \) : the set of index associated with pattern \( j \)
- \( A_i[< \text{ind}_i^j >] \) : the pattern \( j \) associated with array \( A_i \)
Decomposition of a repetition

Input tilers: \( p^j_1 := A_1[< \text{ind}^j_1 >] \) \( p^j_2 := A_2[< \text{ind}^j_2 >] \)

Task: \( p^j_3 := E(p^j_1, p^j_2) \)

Output tiler: \( A_3[< \text{ind}^j_3 >] := p^j_3 \)

Introduction of local variables: \( p^j_1, p^j_2, p^j_3 \)
Decomposition of a repetition

A complete system of equations:

\[
( \mid p_1^1 := A_1[<ind_1^1>] \mid p_2^1 := A_2[<ind_2^1>] \\
\mid p_3^1 := E(p_1^1, p_2^1) \mid A_3[<ind_3^1>] := p_3^1 \\
\mid \ldots \\
\mid p_1^k := A_1[<ind_1^k>] \mid p_2^k := A_2[<ind_2^k>] \\
\mid p_3^k := E(p_1^k, p_2^k) \mid A_3[<ind_3^k>] := p_3^k \\
\mid )
\]

where \( p_1^1, p_2^1, p_3^1, \ldots, p_1^k, p_2^k, p_3^k; \) end;
Parallel model (cont’d)

Restructuring and finalization of the model

$p^1_1 = A_1[<ind^1_1>]$
$p^2_1 = A_1[<ind^2_1>]$
\[\ldots\]
$p^k_1 = A_1[<ind^k_1>]$

A similar structure is shown for $p_2$ and $p_3$.

\[\ldots\]

\[\ldots\]

\[\ldots\]

$A_3[<ind^1_3>] = p^1_3$
$A_3[<ind^2_3>] = p^2_3$
$A_3[<ind^k_3>] = p^k_3$

Commutativity and associativity of composition operator
Case study: video downscaling

\[
F = \begin{pmatrix}
0 & 1 \\
1 & 0 \\
0 & 0
\end{pmatrix}
\]

\[
o = \begin{pmatrix}
0 \\
0 \\
0
\end{pmatrix}
\]

\[
P = \begin{pmatrix}
8 & 0 & 0 \\
0 & 8 & 0 \\
0 & 0 & 1
\end{pmatrix}
\]
module Downscaler_module =
    process DOWNSCALER =
        (?type_array_i A_i;
            !type_array_o A_o;)
        (|(P_i1,...,P_iN:=
            HV_TILER_i(A_i)
        |(P_o1,...,P_oN):=
            R_HV_FILTER(P_i1,...
        |A_o:=HV_TILER_o(P_o1,... |))
where
        type_pattern_i P_i1,...
        type_pattern_o P_o1,...
    process HV_TILER_i =
        (?type_array_i A_i;
            !type_pattern_i P_i1,...,
                (|P_i1:=HV_PATTERN_i1(A_i)
            |...|)
    where
        process HV_PATTERN_i1 =
            (?type_array_i A_i;
                !type_pattern_i P_i;
                    (| p:= H_FILTER (P_i)
                | P_o := V_FILTER(p)
                    |)
    where
        process H_FILTER = ...
        process V_FILTER = ...
end%HV_FILTER%;

end%HV_TILER_i% ;
process R_HV_FILTER =
    (?type_pattern_i P_i1,...,
        !type_pattern_o P_o1,...,
        (|P_o1:=HV_FILTER(P_i1)
        |...|)
where
    process HV_FILTER =
        (? type_pattern_i P_i;
            ! type_pattern_o P_o;)
        (| p:= H_FILTER (P_i)
        | P_o := V_FILTER(p)
            |)
    where
        type_pattern_l p;
    process H_FILTER = ...
    process V_FILTER = ...
end%HV_FILTER%;
Serialized model

- Simple parallel model
  - Semantically equivalent
  - Naively enumeration
- Association of application with architecture: from repetition to iteration, introduction of flows
- Sequentialization at different granularity degrees [Labbani 2006]
From repetition to iteration: introduction of flows

- Array to flow: produces pattern flows from arrays
- Flow to array: produces arrays from pattern flows
Serialized model

Array to flow

Main components: clock oversampling, sequencer and Extraction
Serialized model

Flow to array

Main components: clock undersampling, sequencer and insertion
Introduction
- GASPARD methodology
- General scheme

Data-intensive processing
- ARRAY-OL language
- Existing works

Simple synchronous modeling of GASPARD models
- Parallel model
- Serialized model

Validation issues

Conclusions
We have a synchronous model with parallel and serialized version that can be combined (mixed model)

We want to use synchronous analysis tools to address design correctness issues
ex. N-synchronous Kahn network [Cohen et al. 2006], clock calculus, model-checking

Example: a simple application with affine clocks synchronizability analysis [Smarandache et al. 1999]
General scheme

codesign
transformations
compilation
simulation
TLM, RTL

diagnosis, debug

analysis
verification
clock calculus
simulation

Gaspard2
(Array-OL metamodel)

synchronous
equations
(metamodel)

control

+ transf1 + transf2 +

control

modeling
language

intermediate
format

formal
specification
language

Lustre
Lucid Synchrone
Signal
Synchronizability analysis

Camera functionality in a cell phone

CMOS sensor \( C_p \)

\( p_k^i \)

Downscaler

\( C_a \)

\( p_k^o \)

TFT display

\( C_i \)
Synchronizability analysis

Clock constraints:

1. $c_a$ is an affine undersampling of $c_p$: $c_p^{(1,\phi_1,d_1)} \rightarrow c_a$;
2. $c_i$ is an affine undersampling of $c_a$: $c_a^{(1,\phi_2,d_2)} \rightarrow c_i$;
Synchronizability analysis

Clock constraints:

1. $c_a$ is an affine undersampling of $c_p$: $c_p(1,\phi_1,d_1) \rightarrow c_a$;
2. $c_i$ is an affine undersampling of $c_a$: $c_a(1,\phi_2,d_2) \rightarrow c_i$;

Now, let us consider a given external constraint, which imposes a particular image production rate $c'_i$, from $c_p$ such that: $c_p(1,\phi_3,d_3) \rightarrow c'_i$. What about the synchronizability of $c'_i$ and $c_i$?
Synchronizability analysis

Clock constraints :

1. $c_a$ is an affine undersampling of $c_p$ : $c_p \xrightarrow{(1, \phi_1, d_1)} c_a$;

2. $c_i$ is an affine undersampling of $c_a$ : $c_a \xrightarrow{(1, \phi_2, d_2)} c_i$;

Now, let us consider a given external constraint, which imposes a particular image production rate $c_i'$, from $c_p$ such that : $c_p \xrightarrow{(1, \phi_3, d_3)} c_i'$. What about the synchronizability of $c_i'$ and $c_i$ ?

\[
\text{c}'_i \text{ and } c_i \text{ are synchronizable } \iff \begin{cases} 
\phi_1 + d_1 \phi_2 = \phi_3 \\
d_1 d_2 = d_3
\end{cases}
\]
Conclusions and perspectives

Current results:

- Synchronous modeling of Gaspard specifications
- Analysis of GASPARD applications with the help of synchronous techniques
- Implementation of modeling approach following MDE

In the future:

- Complete implementation and validation of current results
- Extension with control features: mode-automata
- Using mixed models (parallel/serialized) combined with task fusion technique for placements in time and space