Optimizations for intensive signal processing applications on Systems-on-Chip

Calin Glitia

September 6, 2010

Intensive signal processing

Detection systems

Multimedia

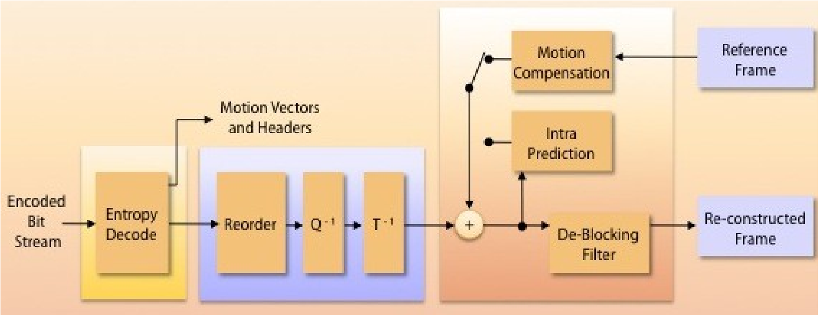
Intensive signal processing

Detection systems

Multimedia

- Repetitive computations
- Considerable amount of data ⇒ Multi-dimensional arrays

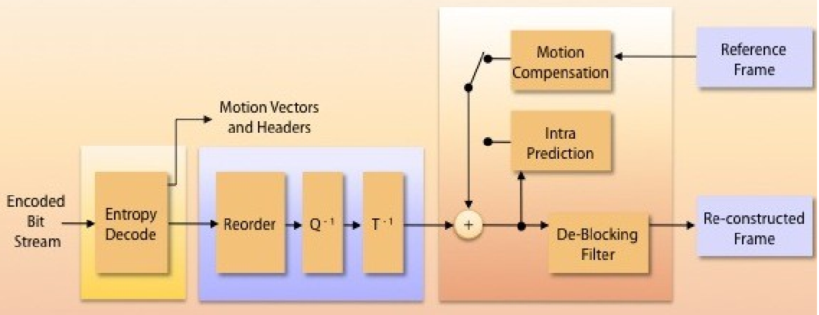
Modular decomposition



Data-flow oriented modeling

- Logical parallelism
  1. Task parallelism and pipeline
  2. Data parallelism

Modular decomposition



Data-flow oriented modeling

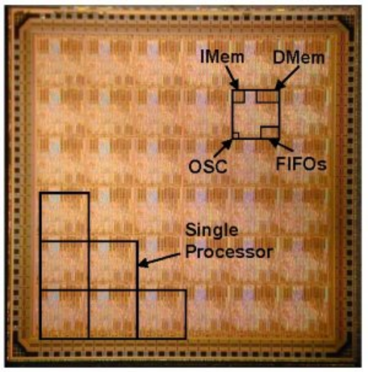
- Logical parallelism
  1. Task parallelism and pipeline
  2. Data parallelism
- Complexity:
  - Elementary functions assemblage
  - Complex accesses at the data structures

Execution platforms

Systems-on-Chip:

- Increase in the integration capacity
- Multiprocessors

Multiprocessor SoC



Execution platforms

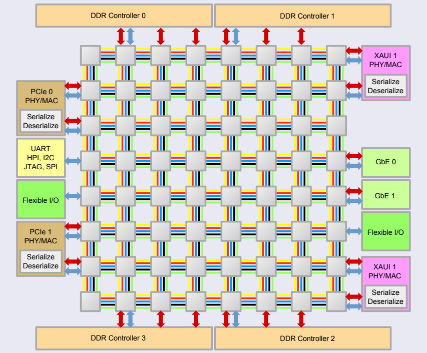
Systems-on-Chip:

- Increase in the integration capacity
- Multiprocessors

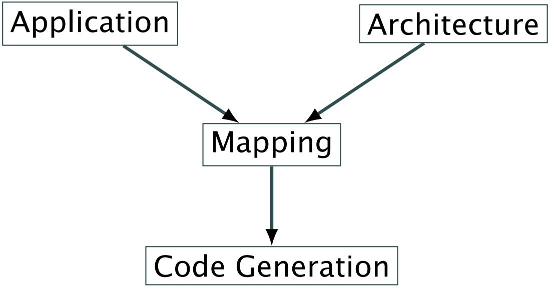
Architecture models :

- Repetitive topologies
- Physical parallelism

Multiprocessor SoC



Co-design



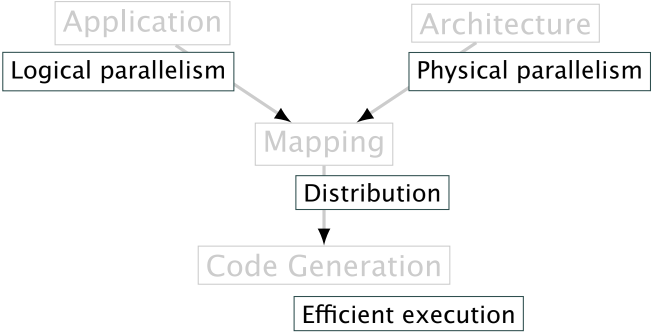
Application

Architecture

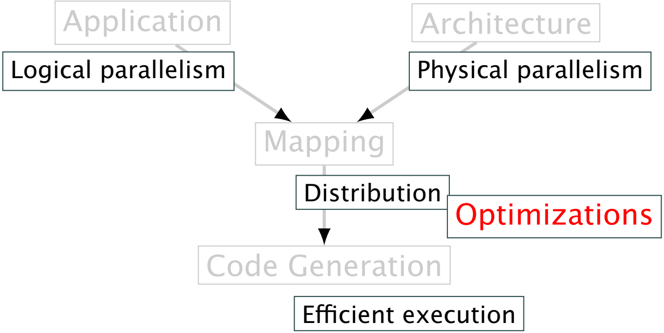
Mapping

Code Generation

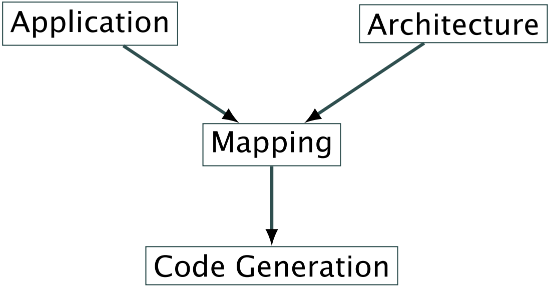
Co-design


Application
Logical parallelism
Architecture
Physical parallelism
Mapping
Distribution
Code Generation
Efficient execution

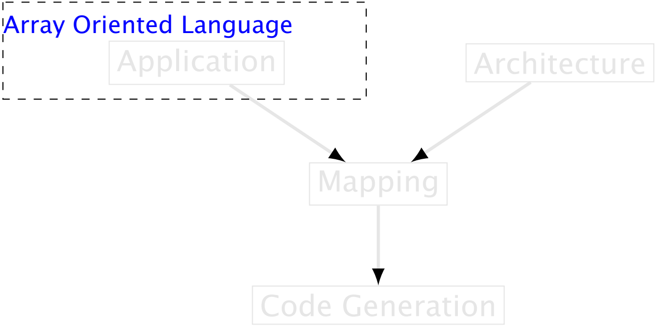
Co-design


Application
Architecture
Logical parallelism
Physical parallelism
Mapping
Distribution
Optimizations
Code Generation
Efficient execution

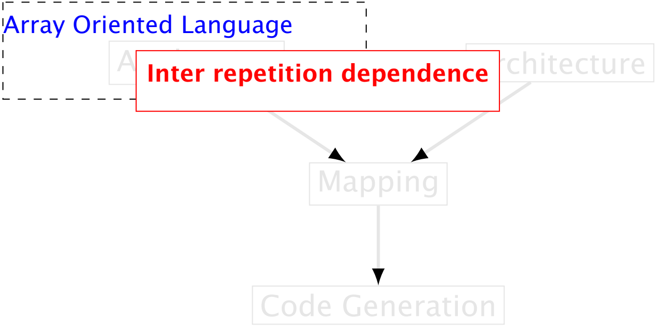
Outline



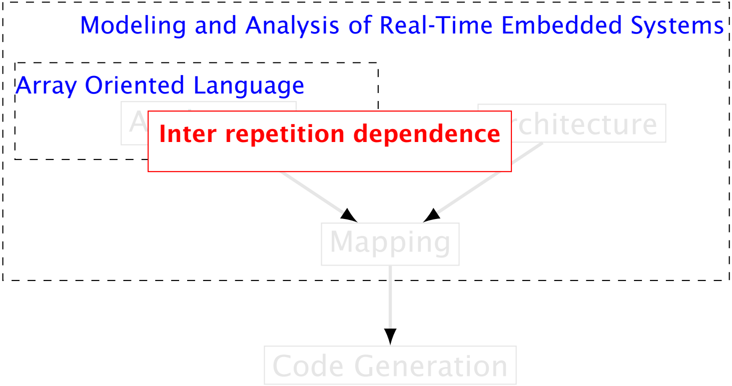
Outline


Array Oriented Language
Application
Architecture
Mapping
Code Generation

Outline


Array Oriented Language
A
Inter repetition dependence
rchitecture
Mapping
Code Generation

Outline



Modeling and Analysis of Real-Time Embedded Systems

Array Oriented Language

A

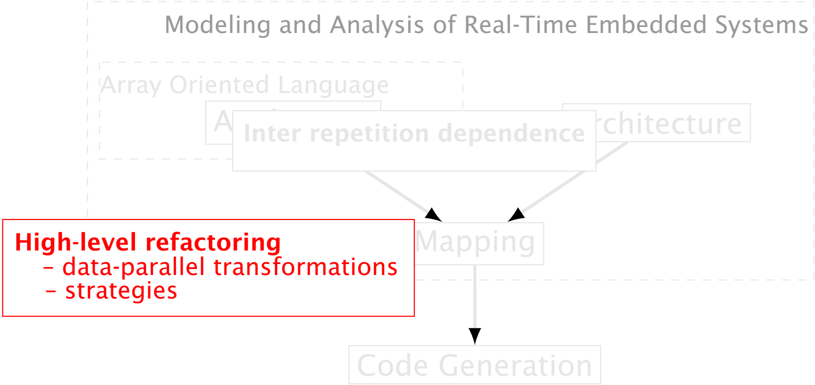
rchitecture

Inter repetition dependence

Mapping

Code Generation

Outline



Modeling and Analysis of Real-Time Embedded Systems

Array Oriented Language

A

Inter repetition dependence

rchitecture

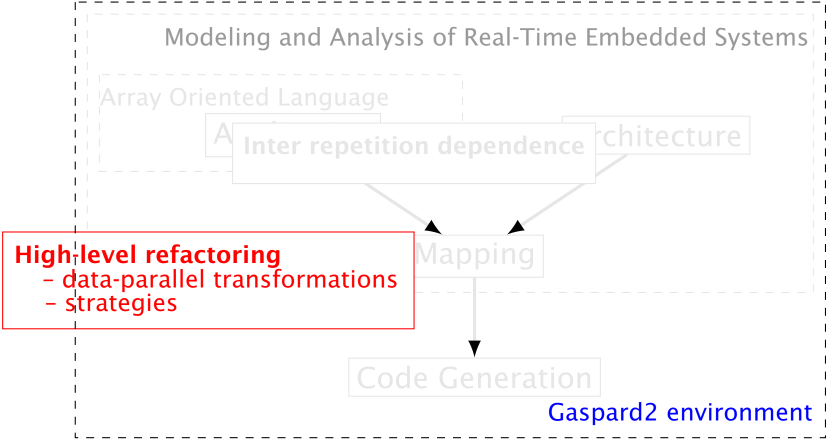
Mapping

Code Generation

High-level refactoring

- data-parallel transformations
- strategies

Outline



Modeling and Analysis of Real-Time Embedded Systems
Array Oriented Language
A
Inter repetition dependence
rchitecture
Mapping
Code Generation

Gaspard2 environment

High-level refactoring
- data-parallel transformations
- strategies

Some of the existing languages

Data-Flow

1. Synchronous Data Flow

Some of the existing languages

Data-Flow

1. Synchronous Data Flow
2. Extensions:
   - Cyclo-Static Data Flow
   - Multi-dimensional Synchronous Data Flow, Windowed Synchronous Data Flow
   - Boolean Data Flow, Dynamic Data Flow

Some of the existing languages

Data-Flow

1. Synchronous Data Flow
2. Extensions:
   - Cyclo-Static Data Flow
   - Multi-dimensional Synchronous Data Flow, Windowed Synchronous Data Flow
   - Boolean Data Flow, Dynamic Data Flow
3. **Array-OL**

Some of the existing languages

Data-Flow

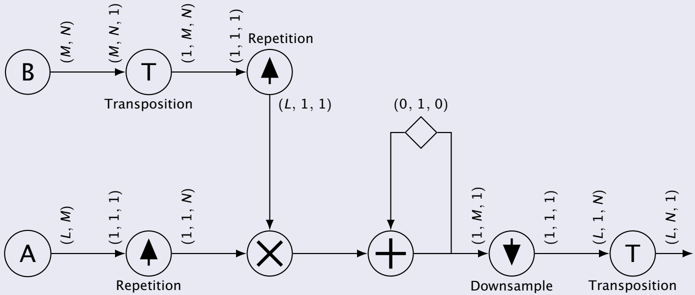
1. Synchronous Data Flow
2. Extensions:
   - Cyclo-Static Data Flow
   - Multi-dimensional Synchronous Data Flow, Windowed Synchronous Data Flow
   - Boolean Data Flow, Dynamic Data Flow
3. **Array-OL**

Functional languages

- Alpha: polyhedron, recurrence equations
- Sisal, Single Assignment C

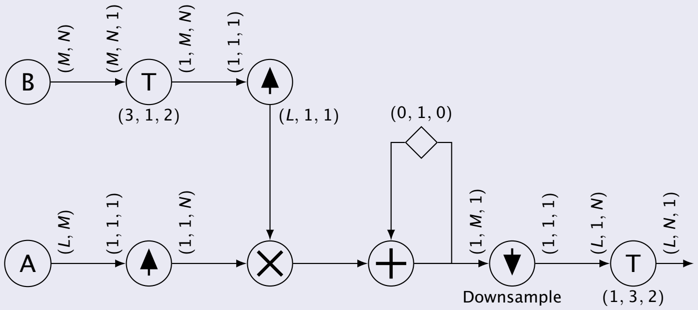
Multidimensional Synchronous DataFlow (MDSDF)

Matrix multiplication



Multidimensional Synchronous DataFlow (MDSDF)

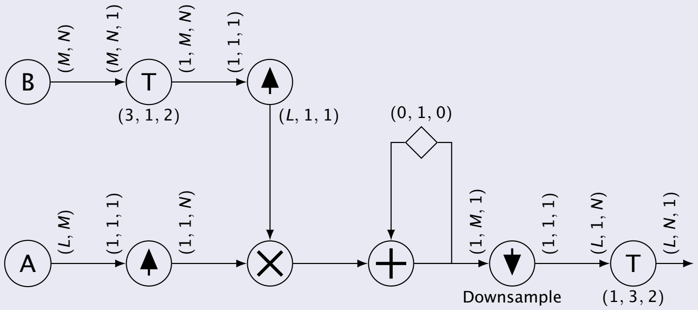
Matrix multiplication



- Data-flow graph: actors consuming/producing MultiDimensional-tokens
- Static analysis/schedule

Multidimensional Synchronous DataFlow (MDSDF)

Matrix multiplication

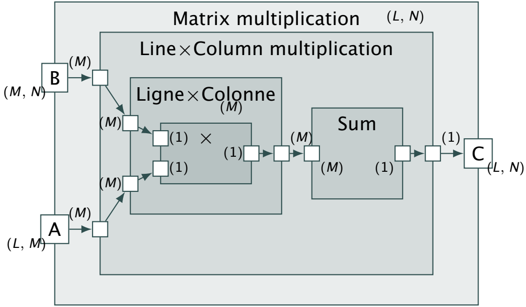


- Data-flow graph: actors consuming/producing MultiDimensional-tokens
- Static analysis/schedule
- Limitations: Multiple data consumptions
- Extensions

Array Oriented Language – principles

Similarities with other data-flow languages

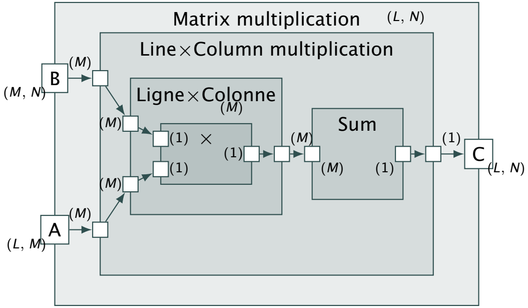
- Hierarchical decomposition – task parallelism
- Data-flow oriented formalism – multi-dimensional data structures



Array Oriented Language – principles

Similarities with other data-flow languages

- Hierarchical decomposition – task parallelism
- Data-flow oriented formalism – multi-dimensional data structures

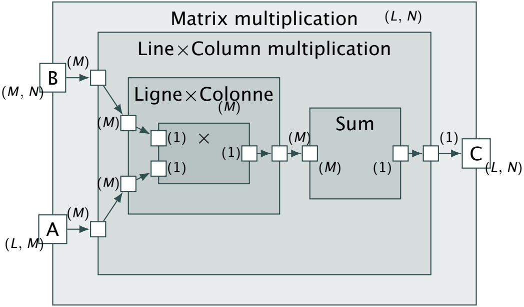


- **Explicit data parallelism : data-parallel repetitions (“loops”)**
- **Uniform paving by sub-arrays**

Array Oriented Language – principles

Similarities with other data-flow languages

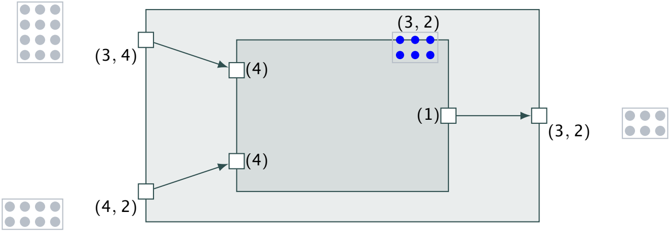
- Hierarchical decomposition – task parallelism
- Data-flow oriented formalism – multi-dimensional data structures



- **Explicit data parallelism : data-parallel repetitions ("loops")**
- **Uniform paving by sub-arrays**
- **Space and time mixed as dimensions of the data structures**

Data-parallel repetitions

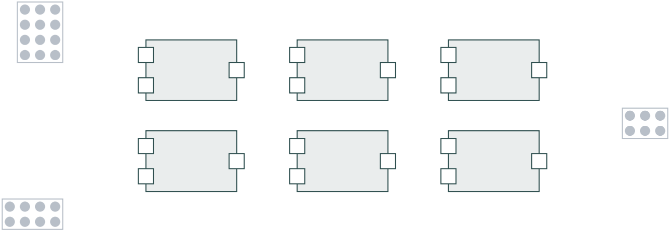
Matrix multiplication: $N = 3, M = 4, L = 2$



Matrix multiplication

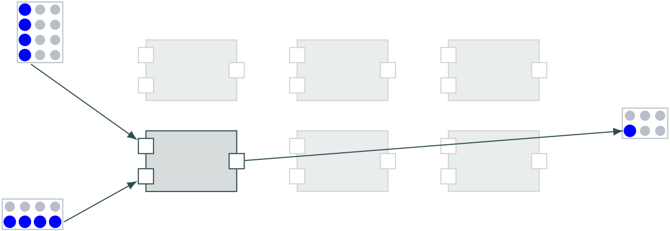
Data-parallel repetitions

Matrix multiplication: $N = 3, M = 4, L = 2$



Line×Column multiplications

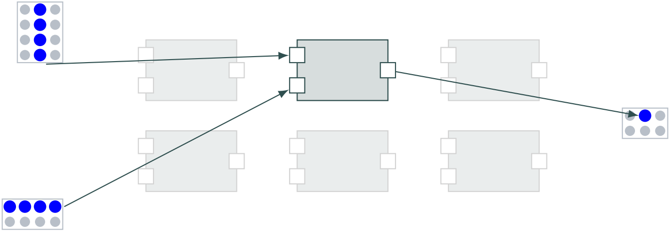
Matrix multiplication: $N = 3, M = 4, L = 2$



uniformly spaced input/output patterns

Data-parallel repetitions

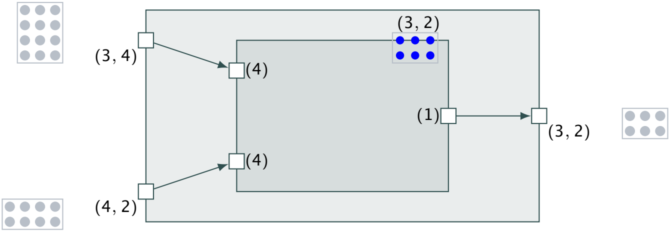
Matrix multiplication: $N = 3, M = 4, L = 2$



DATA-PARALLEL instances

Data-parallel repetitions

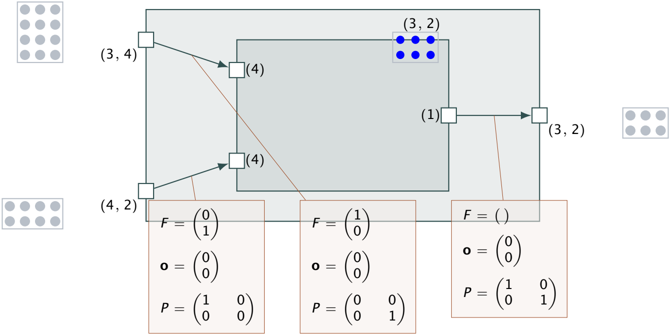
Matrix multiplication: $N = 3, M = 4, L = 2$



Compact representation: repetition space

Data-parallel repetitions

Matrix multiplication: $N = 3, M = 4, L = 2$



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

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

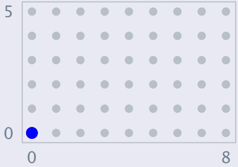
$F = (\,)$ $\mathbf{o} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ $P = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

Tilers – links between input and output patterns

Uniform sub-array accesses (patterns)

Tiler

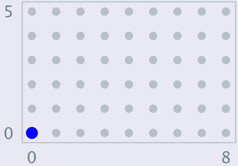
- **o**: origin of the reference pattern



Uniform sub-array accesses (patterns)

Tiler

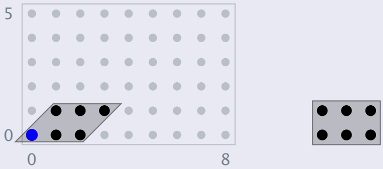
- **o**: origin of the reference pattern



Uniform sub-array accesses (patterns)

Tiler

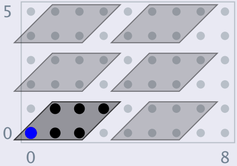
- **o**: origin of the reference pattern
- *F*: Fitting matrix – the shape of the tiles in the array

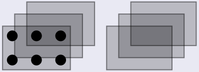


Uniform sub-array accesses (patterns)

Tiler

- **o**: origin of the reference pattern
- *F*: Fitting matrix – the shape of the tiles in the array
- *P*: Paving matrix – uniform spacing of the tiles





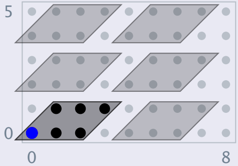
Formal specification:

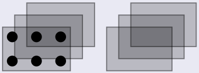
$$\mathbf{o} + (P\ F) \cdot \begin{pmatrix}\mathbf{r}\\ \mathbf{i}\end{pmatrix} \bmod \mathbf{s}_{\text{array}}$$

Uniform sub-array accesses (patterns)

Tiler

- **o**: origin of the reference pattern
- *F*: Fitting matrix – the shape of the tiles in the array
- *P*: Paving matrix – uniform spacing of the tiles



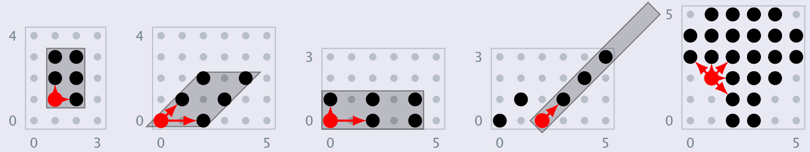


Formal specification:

$$\mathbf{o} + (P\ F) \cdot \begin{pmatrix} \mathbf{r} \\ \mathbf{i} \end{pmatrix} \bmod \mathbf{s}_{\text{array}}$$

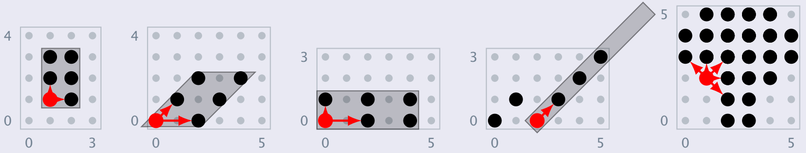
Common motif-based accesses

Pattern examples

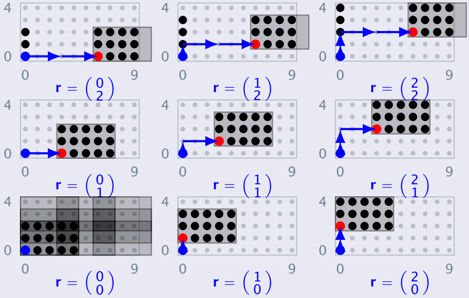


Common motif-based accesses

Pattern examples



Paving example



4
0
0
9
$\mathbf{r} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$
$\mathbf{r} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$
$\mathbf{r} = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$
$\mathbf{r} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$
$\mathbf{r} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$
$\mathbf{r} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$
$\mathbf{r} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$
$\mathbf{r} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$
$\mathbf{r} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$

$$F = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \mathbf{s}_{\text{pattern}} = \begin{pmatrix} 5 \\ 3 \end{pmatrix}$$

$$\mathbf{o} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \mathbf{s}_{\text{array}} = \begin{pmatrix} 10 \\ 5 \end{pmatrix}$$

$$P = \begin{pmatrix} 0 & 3 \\ 1 & 0 \end{pmatrix} \qquad \mathbf{s}_{\text{repetition}} = \begin{pmatrix} 3 \\ 3 \end{pmatrix}$$

Summary Array-OL

Specification

- Data-flow oriented visual formalism
- Express the regularity of computations/data accesses
- Exploit the parallelism

Summary ARRAY-OL

Specification

- Data-flow oriented visual formalism
- Express the regularity of computations/data accesses
- Exploit the parallelism
- **Rules that allow static analysis**

Summary ARRAY-OL

Specification

- Data-flow oriented visual formalism
- Express the regularity of computations/data accesses
- Exploit the parallelism
- **Rules that allow static analysis**

Limitations

- Numerical values for the multidimensional spaces/accesses

Summary ARRAY-OL

Specification

- Data-flow oriented visual formalism
- Express the regularity of computations/data accesses
- Exploit the parallelism
- **Rules that allow static analysis**

Limitations

- Numerical values for the multidimensional spaces/accesses
- Cycles not allowed in the dependence graph

Summary ARRAY-OL

Specification

- Data-flow oriented visual formalism
- Express the regularity of computations/data accesses
- Exploit the parallelism
- **Rules that allow static analysis**

Limitations

- Numerical values for the multidimensional spaces/accesses
- Cycles not allowed in the dependence graph
- **Extension: inter-repetition dependences**

Need for uniform dependences

State construction

- *Transfer data between different instances of the same repetition.*

Need for uniform dependences

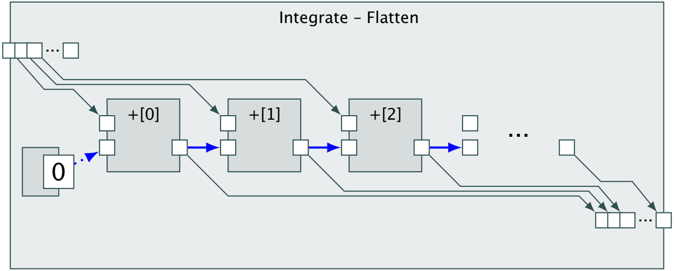
State construction

- *Transfer data between different instances of the same repetition.*
- Examples: Sum, **Integrate**

Need for uniform dependences

State construction

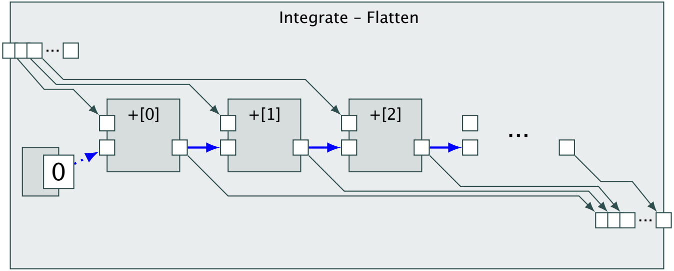
- *Transfer data between different instances of the same repetition.*
- Examples: Sum, **Integrate**



Need for uniform dependences

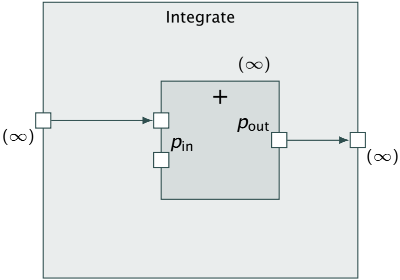
State construction

- *Transfer data between different instances of the same repetition.*
- Examples: Sum, **Integrate**



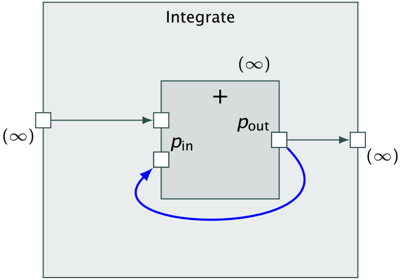
- Uniform data dependences between instances of a repetition

Uniform dependences



Inter-repetition dependence

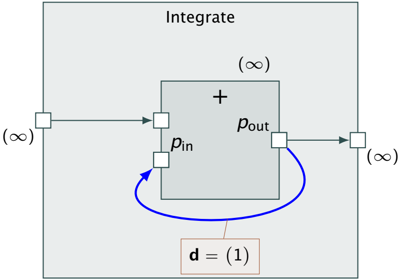
Uniform dependences



Inter-repetition dependence

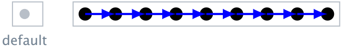
1. Data dependence: $p_{out} \rightarrow p_{in}$

Uniform dependences

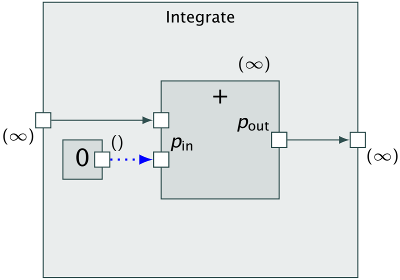


Inter-repetition dependence

1. Data dependence: $p_{out} \rightarrow p_{in}$
2. Dependence vector inside the repetition space

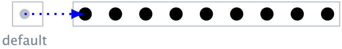


Uniform dependences

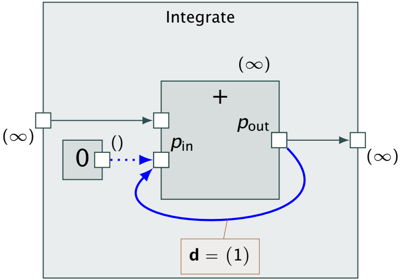


Inter-repetition dependence

1. Data dependence: $p_{out} \rightarrow p_{in}$
2. Dependence vector inside the repetition space
3. Initial values: default link for dependences that exit the repetition space

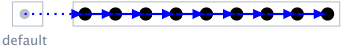


Uniform dependences



Inter-repetition dependence

1. Data dependence: $p_{out} \rightarrow p_{in}$
2. Dependence vector inside the repetition space
3. Initial values: default link for dependences that exit the repetition space



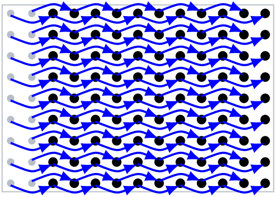
Calin Glitia, Philippe Dumont, and Pierre Boulet.

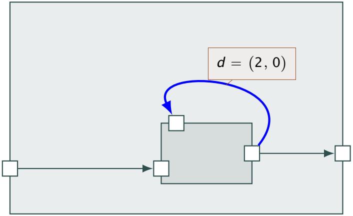
Array-OL with delays, a domain specific specification language for multidimensional intensive signal processing.

Multidimensional Systems and Signal Processing, 2009.

Initial values

Initial values – default link

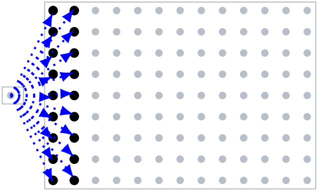


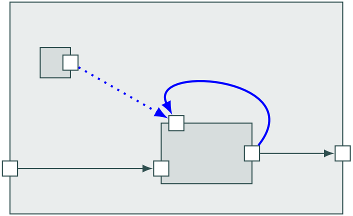


Initial values

Initial values – default link

- Same initial value

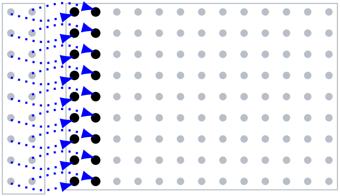


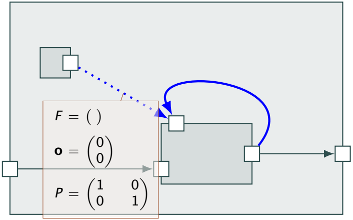


Initial values

Initial values – default link

- Same initial value
- Different values – Tiler

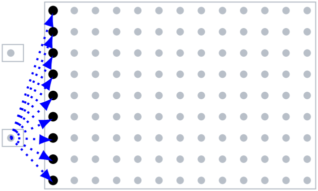


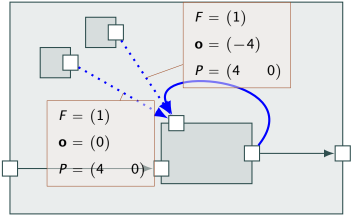

$F = (\ )$
$\mathbf{o} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$
$P = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

Initial values

Initial values – default link

- Same initial value
- Different values – Tiler
- Different default links – Exclusive tilers

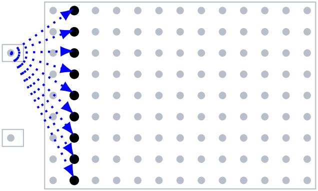


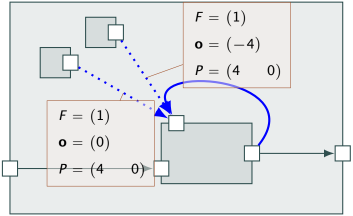


Initial values

Initial values – default link

- Same initial value
- Different values – Tiler
- Different default links – Exclusive tilers

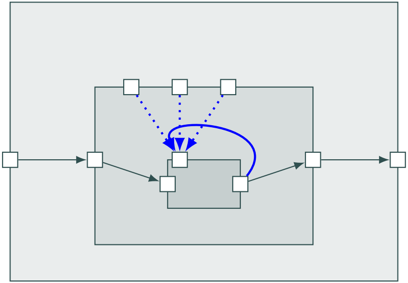




Complex dependences

Dependence constructions:

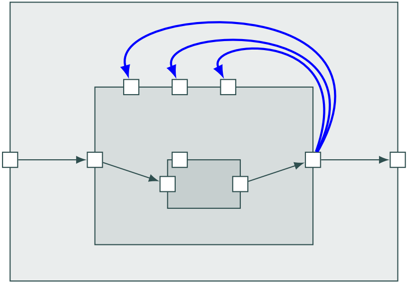
- Multiple default links



Complex dependences

Dependence constructions:

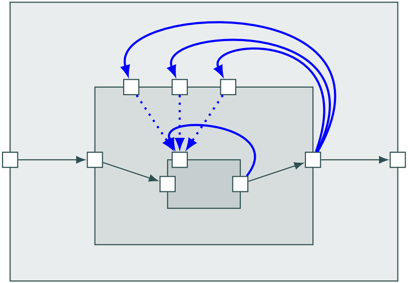
- Multiple default links
- Multiple dependences on a repetition space



Complex dependences

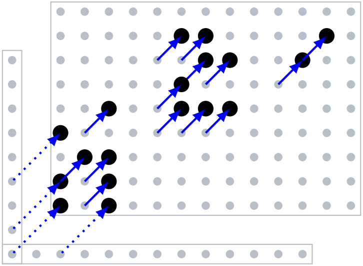
Dependence constructions:

- Multiple default links
- Multiple dependences on a repetition space
- Dependences connected through the hierarchy

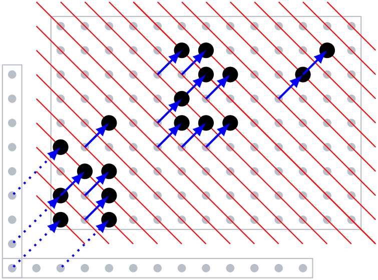


Dependences on the complete repetition space

Impact on the parallelism

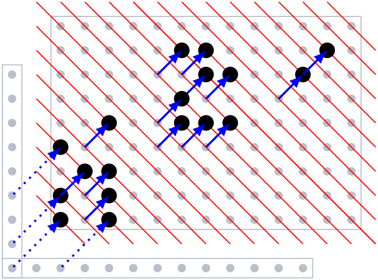


Impact on the parallelism



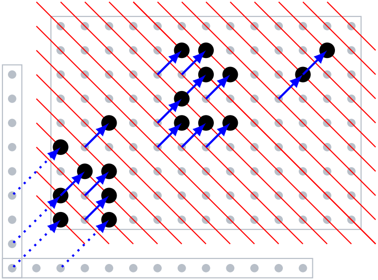
- The repetition space is split in parallel hyper-planes

Impact on the parallelism



- The repetition space is split in parallel hyper-planes
- Pipeline execution following the distance vector

Impact on the parallelism

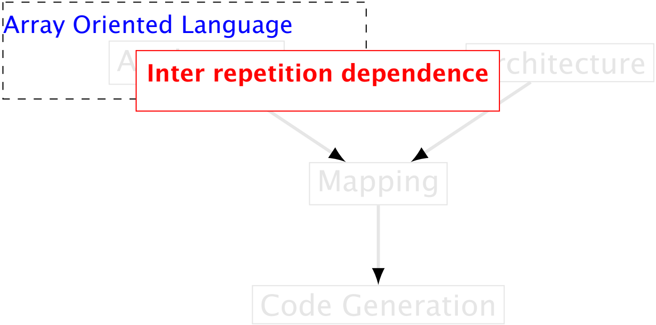


- The repetition space is split in parallel hyper-planes
- Pipeline execution following the distance vector

Scheduling uniform loops

Alain Darte and Yves Robert.
Constructive methods for scheduling uniform loop nests.
IEEE Trans. Parallel Distributed Systems, 5(8):814–822, 1994.

Outline



Array Oriented Language

A

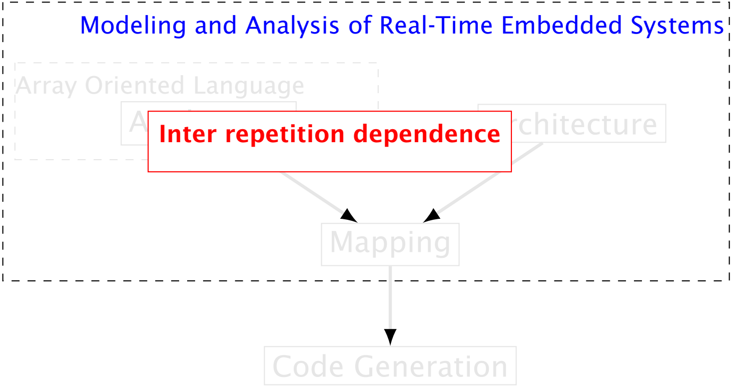
Inter repetition dependence

rchitecture

Mapping

Code Generation

Outline


Modeling and Analysis of Real-Time Embedded Systems
Array Oriented Language
A
Inter repetition dependence
rchitecture
Mapping
Code Generation

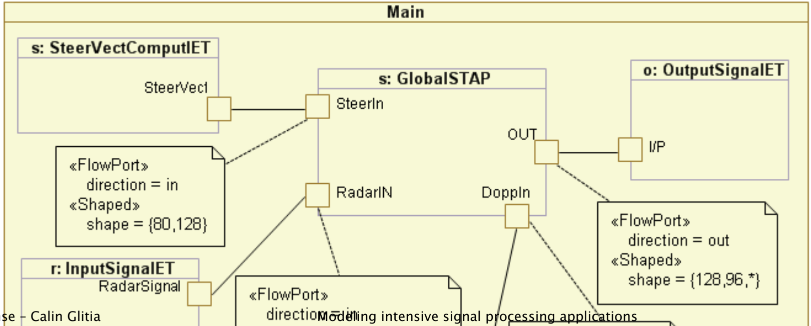
Modeling and Analysis of Real-Time Embedded Systems

Profile UML – standard OMG

- Model Driven Engineering
- Co-design: application, architecture, mapping

Repetitive Structure Modeling

- All the ARRAY-OL concepts are included
- Proposed by the DaRT team


Main
s: SteerVectComputIET
SteerVect
s: GlobalSTAP
SteerIn
OUT
o: OutputSignalET
I/P
«FlowPort» direction = in «Shaped» shape = {80,128}
RadarIN
DoppIn
r: InputSignalET
RadarSignal
«FlowPort» direction = out «Shaped» shape = {128,96,*}
«FlowPort» direction = in

Model repeated inter-connected architecture topologies

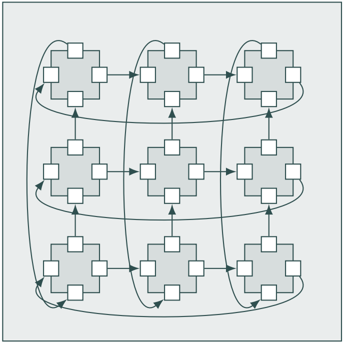
Physical connections between architecture components

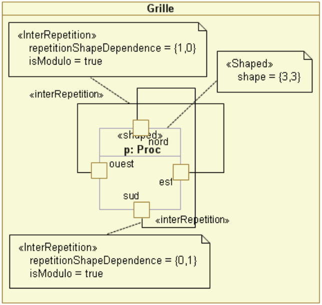
- Compact expression

Model repeated inter-connected architecture topologies

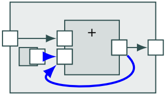
Physical connections between architecture components

- Compact expression
- Cyclic uniform inter-connections





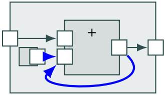
Summary inter-repetition dependences



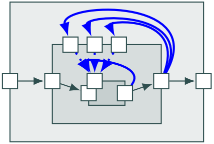
1. Expression of state constructions

Summary inter-repetition dependences

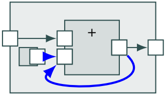
1. Expression of state constructions



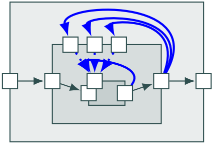
2. Complex dependences through the hierarchy



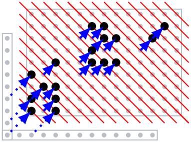
Summary inter-repetition dependences



1. Expression of state constructions



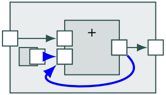
2. Complex dependences through the hierarchy



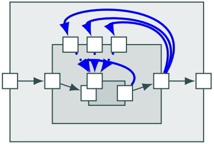
3. Parallelism – pipeline

Summary inter-repetition dependences

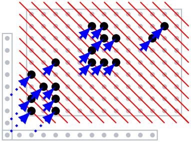
1 Expression of state constructions



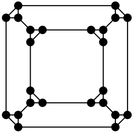
2 Complex dependences through the hierarchy



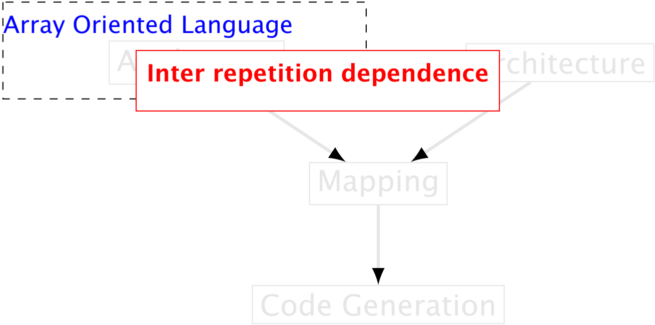
3 Parallelism – pipeline



4 Repeated inter-connected architectures



Outline



Array Oriented Language

A

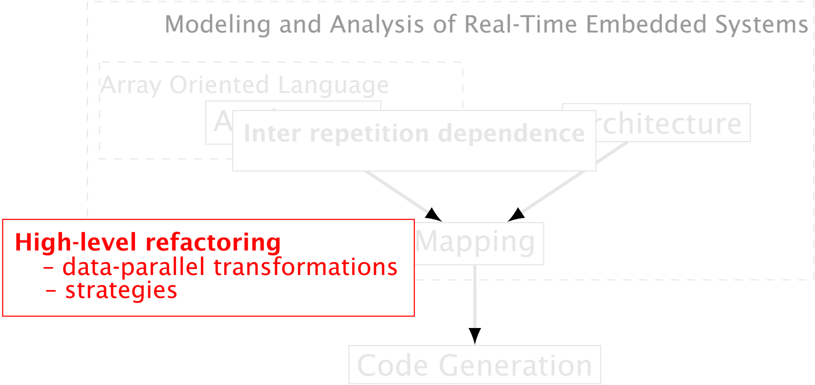
Inter repetition dependence

rchitecture

Mapping

Code Generation

Outline


Modeling and Analysis of Real-Time Embedded Systems

Array Oriented Language

A

Inter repetition dependence

rchitecture

Mapping

Code Generation

High-level refactoring

- data-parallel transformations
- strategies

Execution

Logical space and time as mixed dimensions of multidimensional structure

- Specification: expresses the data dependences
  - between all the data elements that transits the system
- And a partial execution order
  - between all the execution of the tasks in of the system

Execution

Logical space and time as mixed dimensions of multidimensional structure

- Specification: expresses the data dependences
  - between all the data elements that transits the system
- And a partial execution order
  - between all the execution of the tasks in of the system

EFFICIENT execution

- Optimized code generation
- Projection of specification into physical space and time

Execution

Logical space and time as mixed dimensions of multidimensional structure

- Specification: expresses the data dependences
    - between all the data elements that transits the system
- And a partial execution order
    - between all the execution of the tasks in of the system

EFFICIENT execution

- Optimized code generation
- Projection of specification into physical space and time

Adapt a specification to the execution

- High-level refactoring
- Execution that reflects the specification

Projection into space and time

Multi-dimensional structures

repetition spaces

data structures

Projection into space and time

Multi-dimensional structures

repetition spaces
⇓
in space

data structures
⇓
in time

Projection into space and time

Multi-dimensional structures

repetition spaces

⇓

in space

⟷

linked

(trade-off)

data structures

⇓

in time

Projection into space and time

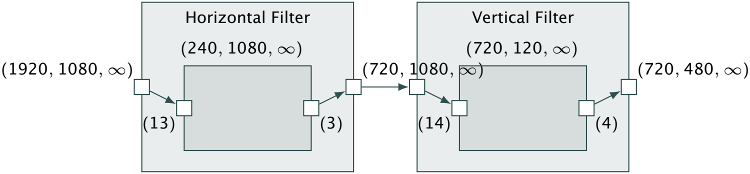
Multi-dimensional structures

repetition spaces ⇓ in space ⟷ linked (trade-off) data structures ⇓ in time

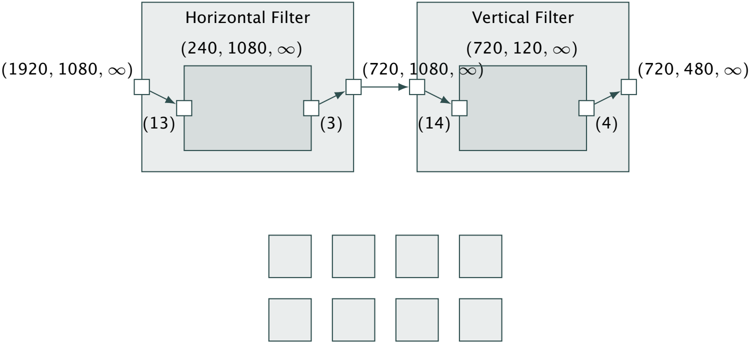
Take into account the execution constraints

- Data dependences
- Available resources

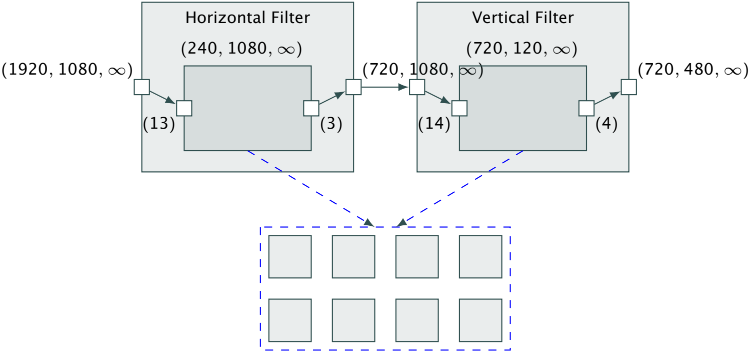
Projection example



Projection example



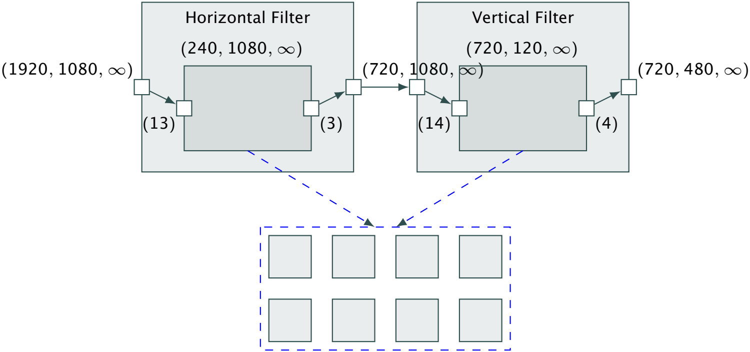
Projection example



Maximal parallelism

- Memory size
- Infinite data structures – Blocking points

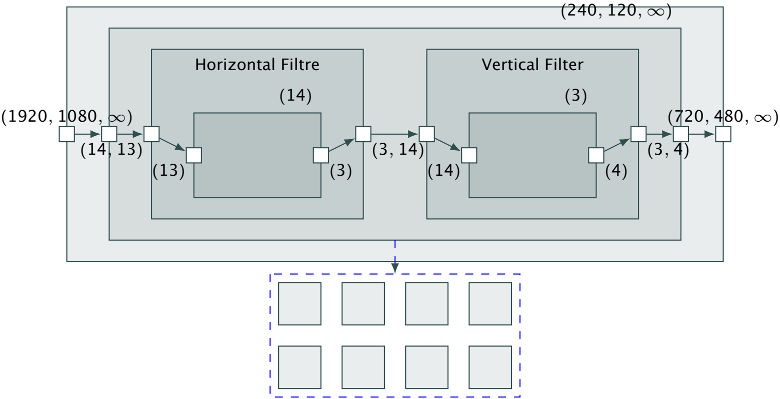
Projection example



Pipeline

- Execution Order

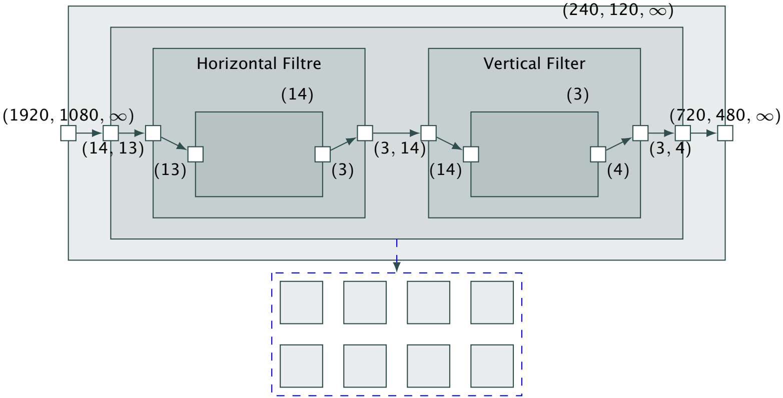
Projection example



Fusion of successive repetitions

- Minimize the arrays – macro-patterns
- Distribution of the common repetition
- Each processor its macro-patterns in memory

Projection example



Re-computations

- When intermediate values are consumed by multiple repetitions
- Trade-off
  - Recompute values
  - Keep in memory – increase of memory size

High-level transformations

Adapt a specification to execution

- change the granularity of the repetitions
- array sizes reductions

High-level transformations

Adapt a specification to execution

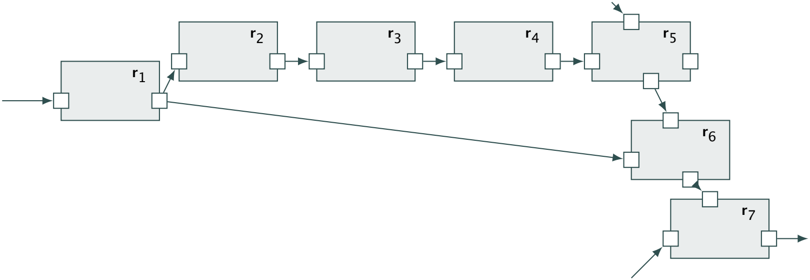
- change the granularity of the repetitions
- array sizes reductions

“High-level” loop transformations

- repetition = visual representation of data-parallel loop nest
- fusion, change paving, tiling, collapse, . . .

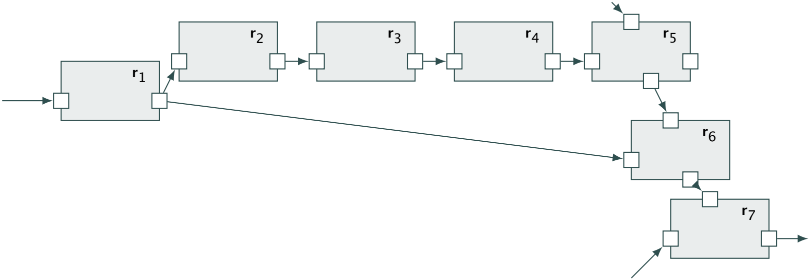
Calin Glitia and Pierre Boulet.
High level loop transformations for multidimensional signal processing embedded applications.
In *International Symposium on Systems, Architectures, Modeling, and Simulation (SAMOS VIII)*, Samos, Greece, July 2008.

Optimization strategies – memory size reduction



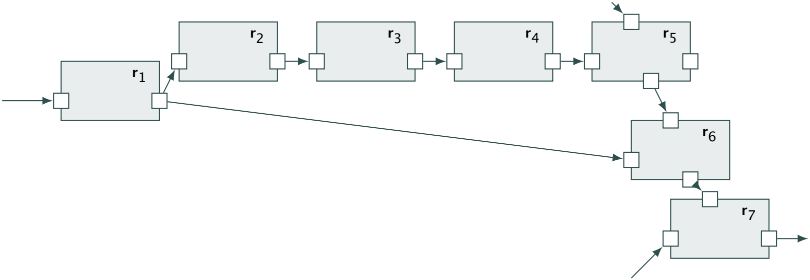
- MAXIMAL reduction of the intermediate arrays

Optimization strategies – memory size reduction



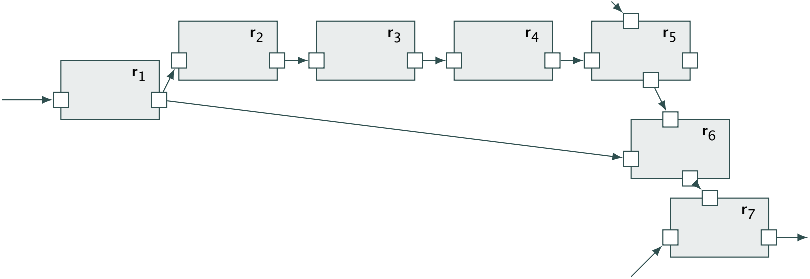
- MAXIMAL reduction of the intermediate arrays
- Fusion of **multiple repetitions**
  - Minimizes only the last intermediate array
  - Re-computations!

Optimization strategies – memory size reduction



- MAXIMAL reduction of the intermediate arrays
- Fusion of **multiple repetitions**
  - Minimizes only the last intermediate array
  - Re-computations!
- Complete fusion?
  - Too much re-computations
  - Limited array reduction

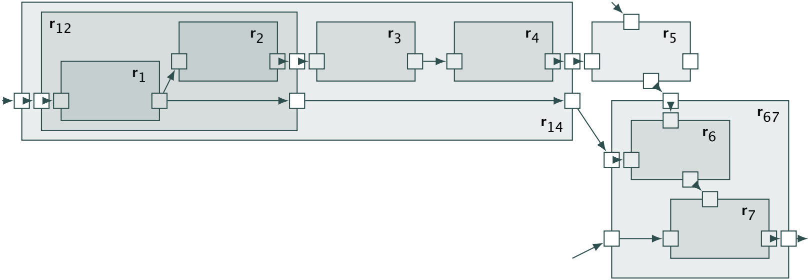
Optimization strategies – memory size reduction



- MAXIMAL reduction of the intermediate arrays

- Strategy that limits the re-computations
  - using result from complete fusion and two-by-two fusions
  - where re-computations are introduces and minimal achievable array reduction

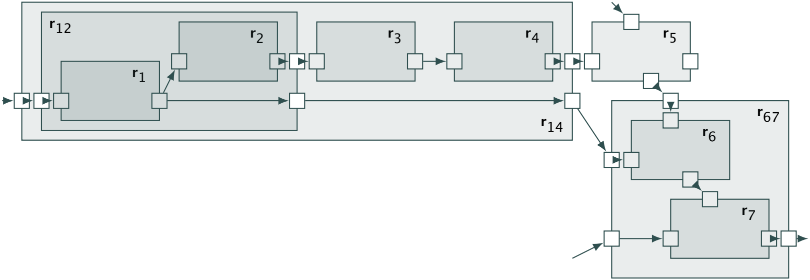
Optimization strategies – memory size reduction



- MAXIMAL reduction of the intermediate arrays

Repetitions before fusion	Repetitions after fusion	Re-computations (product)	Reduction factor of the output arrays
$8 \times 128 \times 96$	$96 \times \begin{pmatrix} 119 \times \begin{pmatrix} 10 \times 8 \\ 1 \end{pmatrix} \\ 80 \times 80 \\ 1 \end{pmatrix}$	9.29	1228.8
119×96		1	96
$80 \times 80 \times 96$		1	96
96		1	1
$128 \times 96 \times 80$	$128 \times 96 \times 80$	1	1
$119 \times 128 \times 96$	$128 \times 96 \times \begin{pmatrix} 119 \\ 1 \end{pmatrix}$	1	12288
128×96		1	1

Optimization strategies – memory size reduction



- MAXIMAL reduction of the intermediate arrays

Calin Glitia, Pierre Boulet, Éric Lenormand, and Michel Barreteau.
Repetitive model refactoring strategy for the design space exploration of intensive signal processing applications.
Journal of Systems Architecture, Special Issue: Hardware/Software CoDesign.

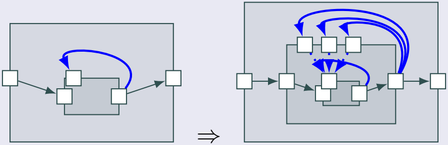
And the inter-repetition dependences?

Why ?

- To allow the use of the refactoring tools on models with uniform dependences

And the inter-repetition dependences?

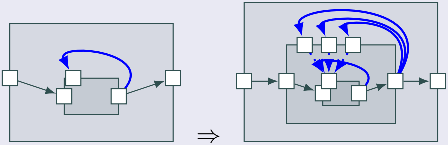
Why ?

- To allow the use of the refactoring tools on models with uniform dependences
- Typically: 

And the inter-repetition dependences?

Why ?

- To allow the use of the refactoring tools on models with uniform dependences
- Typically:



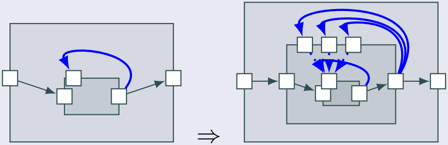
Algorithm

- The global accesses and dependences MUST remain unchanged
- Automatically compute new dependences after a transformation

And the inter-repetition dependences?

Why ?

- To allow the use of the refactoring tools on models with uniform dependences
- Typically:



Algorithm

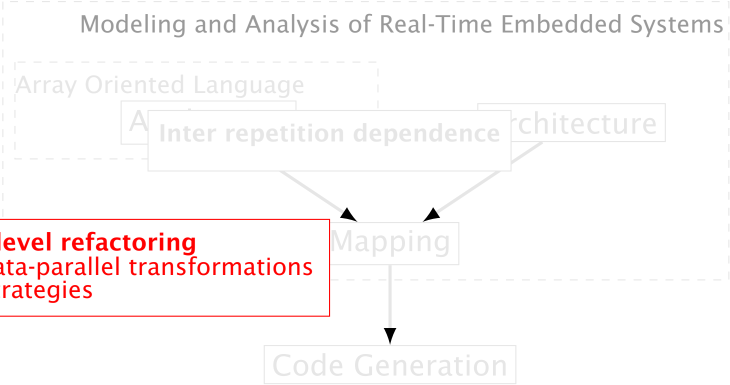
- The global accesses and dependences MUST remain unchanged
- Automatically compute new dependences after a transformation

Calin Glitia and Pierre Boulet.

Interaction between inter-repetition dependences and high-level transformations in array-ol.

In *Conference on Design and Architectures for Signal and Image Processing (DASIP 2009)*, Sophia Antipolis, France, September 2009.

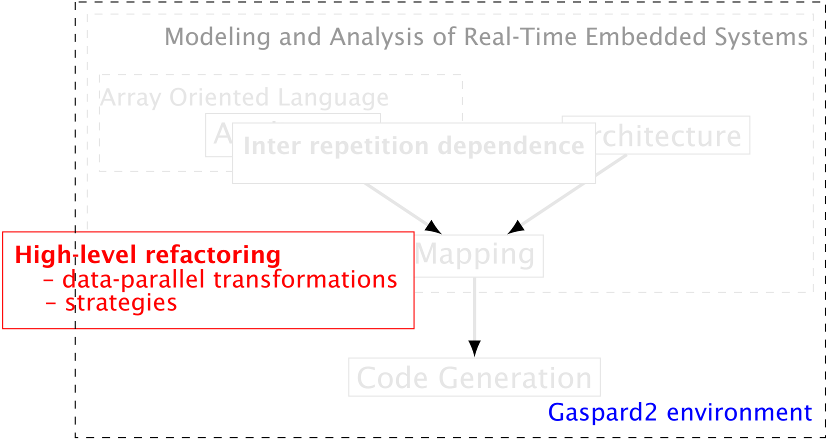
Outline


Modeling and Analysis of Real-Time Embedded Systems
Array Oriented Language
A
Inter repetition dependence
rchitecture
Mapping
Code Generation

High-level refactoring

- data-parallel transformations
- strategies

Outline



Modeling and Analysis of Real-Time Embedded Systems

Array Oriented Language

A

Inter repetition dependence

rchitecture

Mapping

Code Generation

Gaspard2 environment

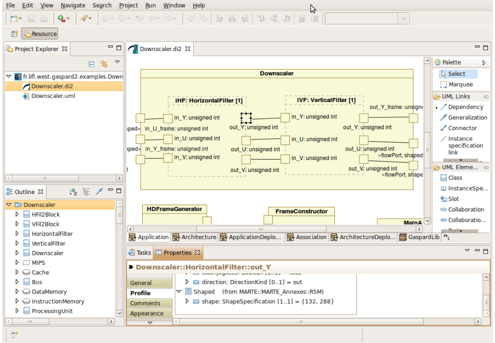
High-level refactoring

- data-parallel transformations
- strategies

Gaspard2 environment

SoC visual co-design

1. allows modeling, simulation and code generation of SoC
2. approach Model Driven Engineering
3. Subset of the MARTE UML profile



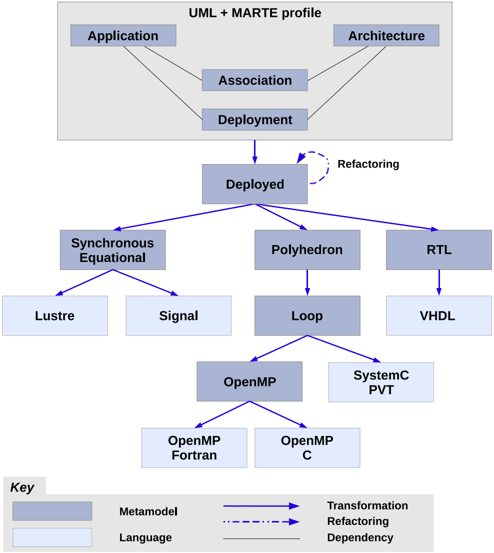
Downscaler::HorizontalFilter::out_Y

direction: DirectionKind [0..1] = out

Shaped (from MARTE::MARTE_Annexes::RSM)

shape: ShapeSpecification [1..1] = {132, 288}

Gaspard2 conception flow

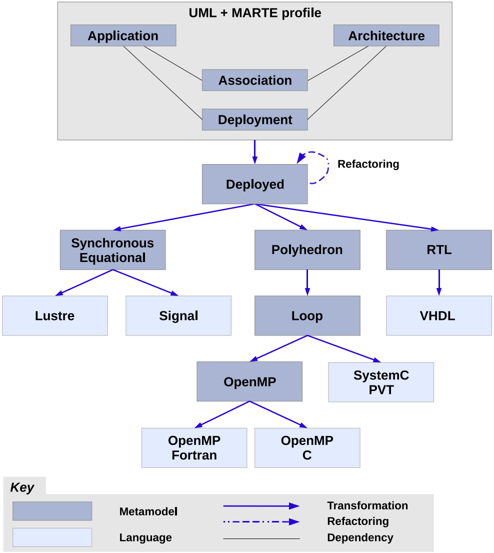


high-level specification

specializations

code generation

Gaspard2 conception flow



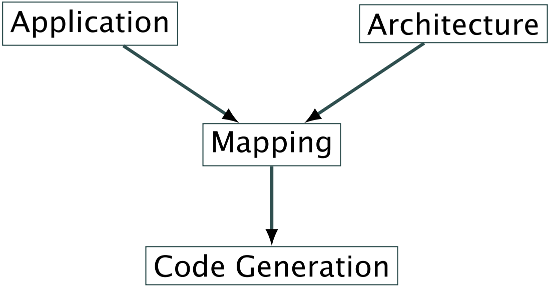
high-level specification

- inter-repetition dependences
- refactoring tools: implementation and integration
- MDE contributions to Gaspard2

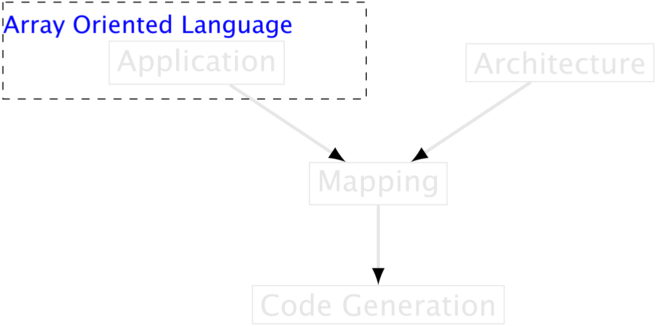
specializations

code generation

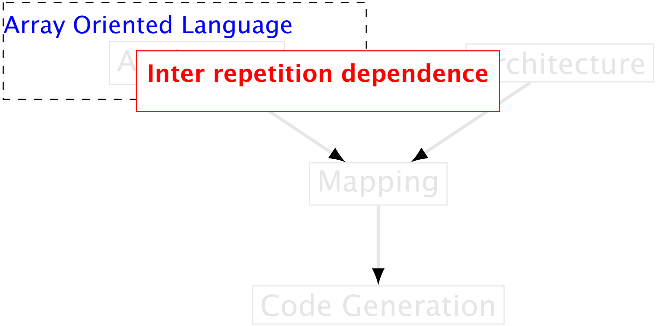
Conclusion



Conclusion


Array Oriented Language
Application
Architecture
Mapping
Code Generation

Conclusion



Array Oriented Language

A

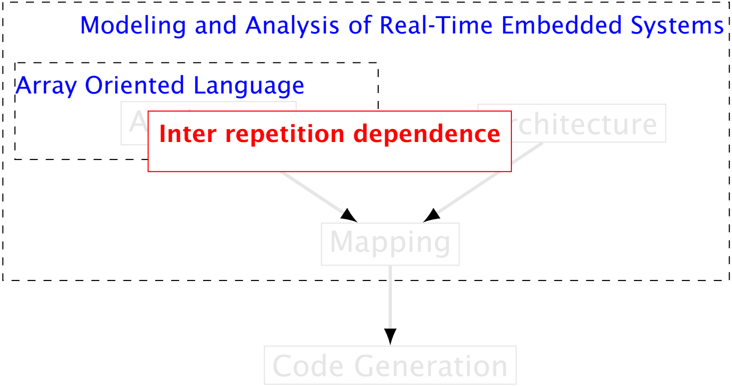
Inter repetition dependence

rchitecture

Mapping

Code Generation

Conclusion

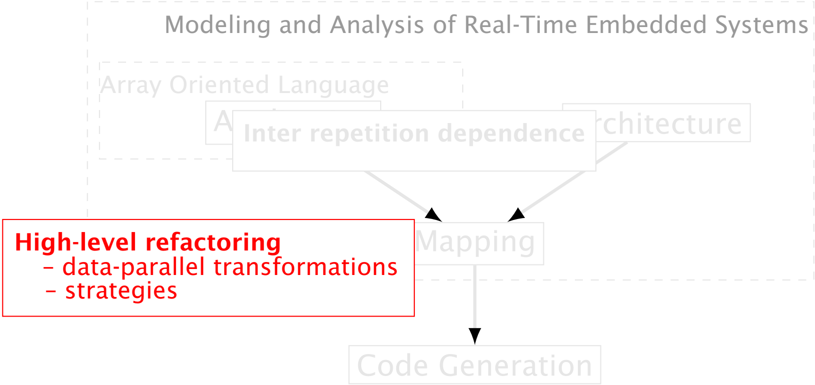


Modeling and Analysis of Real-Time Embedded Systems

Array Oriented Language

Inter repetition dependence

Conclusion



Modeling and Analysis of Real-Time Embedded Systems

Array Oriented Language

A

Inter repetition dependence

rchitecture

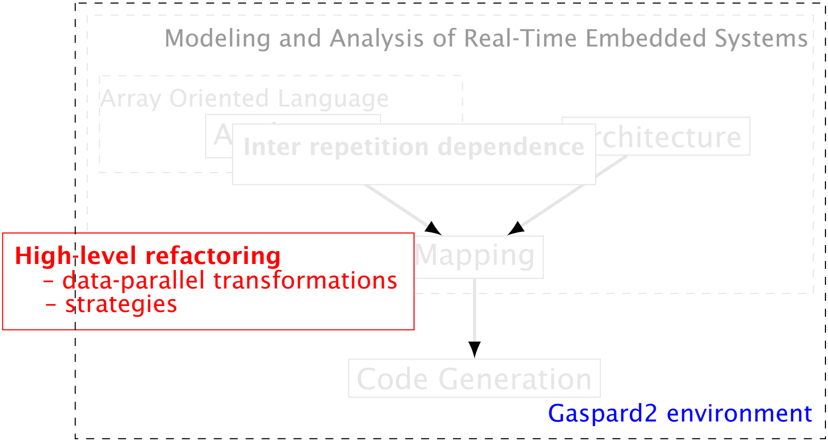
Mapping

Code Generation

High-level refactoring

- data-parallel transformations
- strategies

Conclusion



Modeling and Analysis of Real-Time Embedded Systems

Array Oriented Language

A

Inter repetition dependence

rchitecture

Mapping

Code Generation

Gaspard2 environment

High-level refactoring

- data-parallel transformations
- strategies

Perspectives

. . .