Motivation
Application Model
Deployment using SMT
Symmetry elimination
Distributed memory scheduling
Design Tools
Conclusions

Mapping and Scheduling Streaming Applications using SMT Solvers

Pranav Tendulkar

Supervisors:
Dr. Oded Maler
Dr. Peter Poplavko

Verimag, FRANCE

13 October 2014
Multi-core Processors Everywhere

- Tablets
- Smart-TV
- Space-shuttle
- Cars
- Laptops
- Cameras
Motivation Application Model Deployment using SMT Symmetry elimination Distributed memory scheduling Design Tools Conclusions

Multi-core Processors Everywhere

source: http://www.csl.cornell.edu/courses/ece5745/handouts.html
Multi-core Processors Everywhere

source: http://www.csl.cornell.edu/courses/ece5745/handouts.html
Software on Single Processor System
Software on Single Processor System
Software on Single Processor System

- Processor
- Memory
- Code
- Compiler Tools
Software on Single Processor System
Software on Single Processor System

Motivation

Application Model

Deployment using SMT

Symmetry elimination

Distributed memory scheduling

Design Tools

Conclusions
Multi-core systems

How To:

1. Deploy the application to the platform
2. Decide the number of processors to use
3. Allocate tasks to processors and schedule them

Tendulkar  Mapping/scheduling for many-core
How To:

- **Deploy** the application to the platform
Multi-core systems

How To:

- **Deploy** the application to the platform
- Decide number of **processors** to use?
Multi-core systems

How To:

- **Deploy** the application to the platform
- Decide number of **processors** to use?
- **Allocate** tasks to processors and **schedule** them
Our Deployment Framework

Application Model
Our Deployment Framework
Our Deployment Framework

Motivation Application Model Deployment using SMT Symmetry elimination Distributed memory scheduling Design Tools Conclusions

#Processors

Platform Model

Performance

Constraints

Memory

#Processors

Tendulkar Mapping/scheduling for many-core
Our Deployment Framework

- Motivation
- Application Model
- Deployment using SMT
- Symmetry elimination
- Distributed memory scheduling
- Design Tools
- Conclusions

Our Deployment Framework

Constraints:
- Performance
- Memory
- #Processors

Optimization Techniques

Platform Model

Application Model

Mapping/scheduling for many-core
Our Deployment Framework

- Motivation
- Application Model
- Deployment using SMT
- Symmetry elimination
- Distributed memory scheduling
- Design Tools
- Conclusions

Our Deployment Framework

- Constraints
- Performance
- Memory
- #Processors

Application Model

Platform Model

Optimization Techniques

Solution

Mapping/scheduling for many-core
Our Deployment Framework

Motivation
Application Model
Deployment using SMT
Symmetry elimination
Distributed memory scheduling
Design Tools
Conclusions

Our Deployment Framework

Constraints
Performance
Memory
#Processors

Platform Model
Application Model

Optimization Techniques

Solution

Code

Mapping/scheduling for many-core

Tendulkar
Application Model

Task Graph

- A
- B
- C
- D
- E
- F
- G
- H
- I
- J

Tasks: Software procedure
Edges: Precedence relations annotated with execution time
Application Model

Task Graph

- **Tasks**: Software procedure
**Application Model**

**Task Graph**

- **Tasks**: Software procedure
- **annotated with execution time**
**Application Model**

**Task Graph**

- **Tasks**: Software procedure
- **Edges**: Precedence relations
**Deployment Problem**

**Task Graph**

- **Tasks**: Software procedure
- **Edges**: Precedence relations

**Deployment Solution**

- **Processors**
  - P1: C, F, E, D, H, J
  - P2: A, B, G, I

- **Time**
Deployment Problem

Task Graph

- **Tasks**: Software procedure
- **Edges**: Precedence relations

Deployment Solution

- **Mapping**: Task $\Rightarrow$ Processor

The diagram illustrates a task graph and a deployment solution with processors and time.
Task Graph

- **Tasks**: Software procedure
- **Edges**: Precedence relations

Deployment Solution

- **Mapping**: Task $\Rightarrow$ Processor
- **Scheduling**: Task $\Rightarrow$ Time

Deployment Problem

Motivation | Application Model | Deployment using SMT | Symmetry elimination | Distributed memory scheduling | Design Tools | Conclusions
---|---|---|---|---|---|---

Deployment Problem

Tasks: Software procedure

Edges: Precedence relations

Mapping: Task $\Rightarrow$ Processor

Scheduling: Task $\Rightarrow$ Time
Deployment Problem

Solution 1:

<table>
<thead>
<tr>
<th>Processors</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td></td>
</tr>
</tbody>
</table>

P1: C F E D H J
P2: A B G I
Deployment Problem

Solution1:

<table>
<thead>
<tr>
<th>Processors</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td></td>
</tr>
</tbody>
</table>

Solution2:

<table>
<thead>
<tr>
<th>Processors</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td></td>
</tr>
</tbody>
</table>
Solution space is large

2 proc., 10 tasks \(\approx\) 1000+ potential solutions
Deployment problem

- The difficulty in the deployment is that the design space is **exponential**
The difficulty in the deployment is that the design space is **exponential**.

One needs to model **complex hardware**: Processors, Network, DMA.
Deployment problem

- The difficulty in the deployment is that the design space is exponential.
- One needs to model complex hardware: Processors, Network, DMA.
- Multiple Evaluation Criteria
  - Latency
  - Memory used
  - Processors used
  - ...
Research Questions

How to:

- model the **software**
How to:

- model the **software**
- model the **hardware** (Processors, Network, DMA)
How to:

- model the **software**
- model the **hardware** (Processors, Network, DMA)
- Optimize deployment while dealing with design space **explosion**
Overview

1. Motivation
2. Application Model
3. Deployment using SMT
4. Symmetry elimination
5. Distributed memory scheduling
6. Design Tools
7. Conclusions
Synchronous Dataflow graphs (SDF)
by Edward Lee and David Messerschmitt in 1987
Synchronous Dataflow graphs (SDF)

by Edward Lee and David Messerschmitt in 1987

represents **Streaming Applications**
Synchronous Dataflow graphs (SDF)

by Edward Lee and David Messerschmitt in 1987

represents **Streaming Applications**
Synchronous DataFlow
Synchronous DataFlow

Motivation
Application Model
Deployment using SMT
Symmetry elimination
Distributed memory scheduling
Design Tools
Conclusions

Tendulkar

Mapping/scheduling for many-core

16 / 52
Synchronous DataFlow

SDF Graph

Task Graph
**Motivation**

**Application Model**

**Deployment using SMT**

**Symmetry elimination**

**Distributed memory scheduling**

**Design Tools**

**Conclusions**

---

**Synchronous DataFlow**

---

**Actors**
- Pre-processing
- Blur
- Post-processing

---

**SDF Graph**

**Task Graph**

---

Tendulkar

Mapping/scheduling for many-core
Synchronous DataFlow

**Actors**
- Pre-processing
- Blur
- Post-processing

**Edges**
- Blur executes only after Pre-processing finishes
Synchronous DataFlow

**Actors**
- Pre-processing
- Blur
- Post-processing

**Edges**
- Blur executes only after Pre-processing finishes

**Rates**
- Pre-processing produces 4 pieces of an image (tokens)
- Each Blur consumes 1 piece
Actor Blur is **compact representation** of data parallel tasks.
Actor Blur is **compact representation** of data parallel tasks.
- All Blur tasks have **same properties** such as execution time.
we use **split-join graphs** : restriction of SDF
still covering perhaps 90% of use cases in the literature
we use **split-join graphs** : restriction of SDF
still covering perhaps 90% of use cases in the literature

a simple example:

\[ \begin{array}{ccc}
A & \xrightarrow{\alpha} & B \\
& & \xrightarrow{1/\alpha} \\
& & C
\end{array} \]

$\alpha$: spawn and split
$1/\alpha$: wait and join
we use **split-join graphs**: restriction of SDF
still covering perhaps 90% of use cases in the literature

a simple example:

\[
\begin{array}{ccc}
A & \xrightarrow{\alpha} & B & \xrightarrow{1/\alpha} & C \\
\end{array}
\]

\(\alpha\): spawn and split

\(1/\alpha\): wait and join
Restrictions compared to general SDF

- Stateful actors
- Non-proportional rates
- Initial tokens
- Cyclic paths

\[ A \rightarrow B \]
Split-join does not support:

- **Stateful** actors
Split-join does not support:

- **Stateful** actors
- **Non-proportional** rates
Restrictions compared to general SDF

Split-join does not support:

- **Stateful** actors
- **Non-proportional** rates
- **Initial tokens** and **cyclic** paths
Overview

1. Motivation
2. Application Model
3. Deployment using SMT
4. Symmetry elimination
5. Distributed memory scheduling
6. Design Tools
7. Conclusions
SATisfiability solver (SAT / SMT)
Boolean variables
- \( \text{in}_0, \text{in}_1, \text{in}_2 \ldots \)
- \( \text{out}_0, \text{out}_1, \text{out}_2 \ldots \)
**SATisfiability solver (SAT / SMT)**

- **Boolean variables**
  - $in_0, in_1, in_2, \ldots$
  - $out_0, out_1, out_2, \ldots$

- **Constraints**
  - $out_0 = in_0 \lor in_1 \oplus in_2, \ldots$
SATisfiability solver (SAT / SMT)

- **Boolean variables**
  - \( \text{in}_0, \text{in}_1, \text{in}_2 \ldots \)
  - \( \text{out}_0, \text{out}_1, \text{out}_2 \ldots \)

- **Constraints**
  - \( \text{out}_0 = \text{in}_0 \lor \text{in}_1 \oplus \text{in}_2 \ldots \)
SATisfiability solver (SAT / SMT)

- Boolean variables
  - $in_0, in_1, in_2 \ldots$
  - $out_0, out_1, out_2 \ldots$

- Constraints
  - $out_0 = in_0 \lor in_1 \oplus in_2 \ldots$

Variables

Constraints

SAT solver
SATisfiability solver (SAT / SMT)

- Boolean variables
  - $in_0, in_1, in_2 \ldots$
  - $out_0, out_1, out_2 \ldots$

- Constraints
  - $out_0 = in_0 \lor in_1 \oplus in_2 \ldots$

SAT solver

$\begin{align*}
out_0 &= true \\
&\&
out_1 &= true
\end{align*}$
SATisfiability solver (SAT / SMT)

- **Boolean variables**
  - $in_0$, $in_1$, $in_2$ ...
  - $out_0$, $out_1$, $out_2$ ...

- **Constraints**
  - $out_0 = in_0 \lor in_1 \oplus in_2$ ...

- $out_0 = true$
  - $out_1 = true$

SAT solver

UNSAT
SATisfiability solver (SAT / SMT)

- **Boolean variables**
  - \( \text{in}_0, \text{in}_1, \text{in}_2 \ldots \)
  - \( \text{out}_0, \text{out}_1, \text{out}_2 \ldots \)

- **Constraints**
  - \( \text{out}_0 = \text{in}_0 \lor \text{in}_1 \oplus \text{in}_2 \ldots \)

The diagram illustrates a circuit with inputs and outputs, showing how boolean variables and constraints are used in SAT/SMT solvers. The SAT solver is used to determine the truth values of the variables and constraints.
SATisfiability solver (SAT / SMT)

**Boolean variables**
- \( \text{in}_0, \text{in}_1, \text{in}_2 \ldots \)
- \( \text{out}_0, \text{out}_1, \text{out}_2 \ldots \)

**Constraints**
- \( \text{out}_0 = \text{in}_0 \lor \text{in}_1 \oplus \text{in}_2 \ldots \)

Variables
- \( \text{out}_0 = \text{false} \)
- \( \text{out}_1 = \text{true} \)

SAT solver
- \( \text{in}_0 = \text{true}, \quad \text{in}_1 = \text{false}, \ldots \)
SATisfiability solver (SAT / SMT)

- **Boolean variables**
  - \( \text{in}_0, \text{in}_1, \text{in}_2 \ldots \)
  - \( \text{out}_0, \text{out}_1, \text{out}_2 \ldots \)

- **Constraints**
  - \( \text{out}_0 = \text{in}_0 \lor \text{in}_1 \oplus \text{in}_2 \ldots \)

SMT deals with **numeric** variables and constants
Encoding deployment with constraints

Task Graph

<table>
<thead>
<tr>
<th>Actor</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks</td>
<td>A₀</td>
<td>B₀</td>
<td>B₁</td>
</tr>
<tr>
<td>Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variables</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Precedence Constraints

\[ x_B^0 \geq (x_A^0 + d_A) \]

Mutual Exclusion Constraints

\[ \text{if } p_B^1 = p_B^2 \text{ then } \]
\[ x_B^1 \geq (x_B^2 + d_B) \lor x_B^2 \geq (x_B^1 + d_B) \]
Encoding deployment with constraints

**Task Graph**

![Task Graph Image]

**Actor** | A | B | C
---|---|---|---
**Tasks** | A₀ | B₀ | B₁ | B₂ | B₃ | C₀
**Description** | **Variables**
Start time | xA₀ | xB₀ | xB₁ | xB₂ | xB₃ | xC₀

**Precedence Constraints**

\[x_B \geq (x_A + d_A)\]

**Mutual Exclusion Constraints**

\[
\text{if } (p_B = p_B) \text{ then } x_B \geq (x_B + d_B) \lor x_B \geq (x_B + d_B)
\]

**Latency Cost**

\[\text{Latency} = (x_C + d_C)\]
Encoding deployment with constraints

Task Graph

<table>
<thead>
<tr>
<th>Actor</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks</td>
<td>A₀</td>
<td>B₀</td>
<td>B₁</td>
</tr>
<tr>
<td>Description</td>
<td>Variables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>----------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start time</td>
<td>xA₀</td>
<td>xB₀</td>
<td>xB₁</td>
</tr>
<tr>
<td>Allocated proc.</td>
<td>pA₀</td>
<td>pB₀</td>
<td>pB₁</td>
</tr>
</tbody>
</table>
Encoding deployment with constraints

Task Graph

<table>
<thead>
<tr>
<th>Actor</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks</td>
<td>A₀</td>
<td>B₀</td>
<td>B₁</td>
</tr>
<tr>
<td>Description</td>
<td>Start time</td>
<td>xA₀</td>
<td>xB₀</td>
</tr>
<tr>
<td></td>
<td>Allocated proc.</td>
<td>pA₀</td>
<td>pB₀</td>
</tr>
<tr>
<td></td>
<td>Duration</td>
<td>dA</td>
<td>dB</td>
</tr>
</tbody>
</table>
Motivation Application Model Deployment using SMT Symmetry elimination Distributed memory scheduling Design Tools Conclusions

Encoding deployment with constraints

Task Graph

- **Precedence Constraints**
  - $xB_0 \geq (xA_0 + dA)$

<table>
<thead>
<tr>
<th>Actor</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tasks</strong></td>
<td>$A_0$</td>
<td>$B_0$</td>
<td>$B_1$</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td><strong>Variables</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start time</td>
<td>$xA_0$</td>
<td>$xB_0$</td>
<td>$xB_1$</td>
</tr>
<tr>
<td>Allocated proc.</td>
<td>$pA_0$</td>
<td>$pB_0$</td>
<td>$pB_1$</td>
</tr>
<tr>
<td>Duration</td>
<td>$dA$</td>
<td>$dB$</td>
<td>$dC$</td>
</tr>
</tbody>
</table>

Precedence Constraints

Mutual Exclusion Constraints

if ($pB_1 = pB_2$) then
  $xB_1 \geq (xB_2 + dB)$ \lor $xB_2 \geq (xB_1 + dB)$

Latency Cost

$Latency = (xC_0 + dC)$
Encoding deployment with constraints

**Task Graph**

- **Precedence Constraints**
  - \( xB_0 \geq (xA_0 + dA) \)

- **Mutual Exclusion Constraints**
  - if \((pB_1 = pB_2)\) then 
    \( xB_1 \geq (xB_2 + dB) \lor xB_2 \geq (xB_1 + dB) \)

### Table: Actor \( A \) vs. \( B \) vs. \( C \)

<table>
<thead>
<tr>
<th>Description</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tasks</strong></td>
<td>( A_0 \quad B_0 \quad B_1 \quad B_2 \quad B_3 \quad C_0 )</td>
</tr>
<tr>
<td><strong>Start time</strong></td>
<td>( xA_0 \quad xB_0 \quad xB_1 \quad xB_2 \quad xB_3 \quad xC_0 )</td>
</tr>
<tr>
<td><strong>Allocated proc.</strong></td>
<td>( pA_0 \quad pB_0 \quad pB_1 \quad pB_2 \quad pB_3 \quad pC_0 )</td>
</tr>
<tr>
<td><strong>Duration</strong></td>
<td>( dA \quad dB \quad dC )</td>
</tr>
</tbody>
</table>
Encoding deployment with constraints

- **Precedence Constraints**
  - \(x_B \geq (x_A + d_A)\)

- **Mutual Exclusion Constraints**
  - If \((p_B = p_B)\) then
    \(x_B \geq (x_B + d_B) \lor x_B \geq (x_B + d_B)\)

- **Latency Cost**
  - Latency = \((x_C + d_C)\)
Multi-criteria Problem

Latency

Processors

Latency

Pareto Set

Mapping/scheduling for many-core

Tendulkar
Multi-criteria Problem

Latency = 4
#Proc = 2

(4,2)
Multi-criteria Problem

Latency = 4  
#Proc = 2

Latency = 3  
#Proc = 4
Multi-criteria Problem

Conflicting Criteria

Latency = 4
#Proc = 2

Latency = 3
#Proc = 4
Multi-criteria Problem

Latency = 4  
#Proc = 2

Latency = 3  
#Proc = 4

Pareto Set

(3,4)  
(4,2)
Problem Monotonicity

![Diagram showing monotonicity concept with axes labeled 'Processors' and 'Latency', and upper bounds indicated.]

Upper Bound

Upper Bound
Problem Monotonicity

Latency = 4
#Proc = 2
Problem Monotonicity

Latency = 4  
#Proc = 2

Latency = 5  
#Proc = 3
Problem Monotonicity

\[ \begin{array}{c|ccc}
P_2 & A_0 & B_0 & B_2 \\
P_1 & B_1 & B_3 & C_0 \\
\end{array} \]

Time

Latency = 4

#Proc = 2

\[ \begin{array}{c|ccc}
P_3 & A_0 \\
P_2 & B_0 & B_2 \\
P_1 & B_1 & B_3 & C_0 \\
\end{array} \]

Time

Latency = 5

#Proc = 3

Upper Bound

SAT

Processors

Latency
Problem Monotonicity

Latency $\leq 4$

$\#\text{Proc} \leq 2$

Not Possible
Problem Monotonicity

Latency \leq 4
\#Proc \leq 2
Not Possible

Latency = 2
\#Proc = 1
Also Not Possible
Problem Monotonicity

Latency ≤ 4
#Proc ≤ 2
Not Possible

Latency = 2
#Proc = 1
Also Not Possible

Upper Bound

UNSAT
Design Space Exploration
Tendulkar

Mapping/scheduling for many-core
Design Space Exploration

Split-join Graph

SMT Constraints
**Design Space Exploration**

- **Split-join Graph**
- **SMT Constraints**
- **SMT Solver**

**Design Space Exploration Algorithm**

\[
\begin{align*}
\text{cost} & = x_1 + y_1 \\
\text{constraints} & = \{ x_2 + y_2, x_3 + y_3 \} \\
\text{solutions} & = \{ (x_1, y_1), (x_2, y_2), (x_3, y_3) \}
\end{align*}
\]

**SAT**

**UNSAT**

**TIMEOUT**

**Timeout:** Cannot decide SAT / UNSAT in a given TIME-BUDGET.
Design Space Exploration

- Split-join Graph
- SMT Constraints
- Design Space Exploration Algorithm
- Cost constraints
- SMT Solver

Motivation | Application Model | Deployment using SMT | Symmetry elimination | Distributed memory scheduling | Design Tools | Conclusions
Design Space Exploration

- Split-join Graph
- SMT Constraints
- Design Space Exploration Algorithm
- cost constraints
- SMT Solver
- solutions
Design Space Exploration

Motivation  Application Model  Deployment using SMT  Symmetry elimination  Distributed memory scheduling  Design Tools  Conclusions

Split-join Graph  SMT Constraints  Design Space Exploration Algorithm  SMT Solver  SAT  solutions

cost constraints  \((x_1, y_1)\)

Tendulkar
Mapping/scheduling for many-core

Timeout: Cannot decide SAT / UNSAT in a given time-budget.
Design Space Exploration

Design Space Exploration Algorithm

- Split-join Graph
- SMT Constraints
- cost constraints
- (x₂, y₂)
- SMT Solver
- UNSAT
- solutions

SAT (x₁, y₁)

UNSAT (x₃, y₃)

Timeout: Cannot decide SAT / UNSAT in a given time-budget.
Design Space Exploration

Split-join Graph

Design Space Exploration Algorithm

SMT Constraints

SMT Solver

solutions

Timeout:
Cannot decide SAT / UNSAT in a given TIME-BUDGET.
Design Space Exploration

- **Split-join Graph**
- **SMT Constraints**
- **Design Space Exploration Algorithm**
  - cost constraints
- **SMT Solver**
- **solutions**

**Motivation**

**Application Model**

**Deployment using SMT**

**Symmetry elimination**

**Distributed memory scheduling**

**Design Tools**

**Conclusions**
Exploration Algorithm
**Exploration Algorithm**

- **Divide cost space** using grids

- **sat points**
- **unsat points**
- **not yet explored points**
Exploration Algorithm

- **Divide cost space** using grids
- **One SMT query** per point on the grid

- sat points
- unsat points
- not yet explored points
Exploration Algorithm

- **Divide cost space** using grids
- **One SMT query** per point on the grid
- **Finer** grid after every iteration

![Grids](image)

- ● sat points
- ■ unsat points
- ● not yet explored points
Exploration Algorithm

- **Divide cost space** using grids
- **One SMT query** per point on the grid
- **Finer** grid after every iteration
- Don’t query in **known area**

- ⬤ sat points
- ⬤ unsat points
- ⬤ not yet explored points
Overview

1. Motivation
2. Application Model
3. Deployment using SMT
4. Symmetry elimination
5. Distributed memory scheduling
6. Design Tools
7. Conclusions
Task Symmetry

All instances of actor $C$ are similar (symmetric).

No change in latency!

Huge number of such symmetric solutions.

Add constraints to eliminate all but one.
Task Symmetry

- all instances of actor $C$ are similar (symmetric)
**Task Symmetry**

- **Motivation**
- **Application Model**
- **Deployment using SMT**
- **Symmetry elimination**
- **Distributed memory scheduling**
- **Design Tools**
- **Conclusions**

---

- **Task Symmetry**

  - **Task Graph**
  - **Schedule**
  - **Time**

- **A schedule**

  - **P2**
    - A0, B0, C11, D1
  - **P1**
    - B1, C10, C01, C00, D0, E0

- **Task graph**

- **All instances of actor C are similar (symmetric)**
Task Symmetry

- **Task graph**

<table>
<thead>
<tr>
<th></th>
<th>P₂</th>
<th>P₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>A₀</td>
<td>B₀</td>
<td>C₁₁</td>
</tr>
<tr>
<td>D₁</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₀₀</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B₀</td>
<td>C₀₁</td>
<td>D₀</td>
</tr>
<tr>
<td>E₀</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C₀₁</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B₁</td>
<td>C₁₀</td>
<td>C₀₁</td>
</tr>
<tr>
<td>D₀</td>
<td>C₀₀</td>
<td>E₀</td>
</tr>
<tr>
<td>C₁₁</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **a schedule**
  - P₂: A₀ B₀ C₁₁ D₁
  - P₁: B₁ C₁₀ C₀₁ C₀₀ D₀ E₀
  - time

- **a permuted schedule**
  - P₂: A₀ B₀ C₁₁ D₁
  - P₁: B₁ C₁₀ C₀₀ C₀₁ D₀ E₀
  - time

- all instances of actor C are similar (symmetric)
Task Symmetry

- All instances of actor $C$ are similar (symmetric)

**Task graph**

**a schedule**

<table>
<thead>
<tr>
<th>Time</th>
<th>P2</th>
<th>P1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A0</td>
<td>B0</td>
</tr>
<tr>
<td></td>
<td>B1</td>
<td>C10</td>
</tr>
</tbody>
</table>

**a permuted schedule**

<table>
<thead>
<tr>
<th>Time</th>
<th>P2</th>
<th>P1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A0</td>
<td>B0</td>
</tr>
<tr>
<td></td>
<td>B1</td>
<td>C10</td>
</tr>
</tbody>
</table>

**Motivation**
- Application Model
- Deployment using SMT
- Symmetry elimination
- Distributed memory scheduling
- Design Tools
- Conclusions

**Tendulkar**
Mapping/scheduling for many-core

28 / 52
Task Symmetry

- all instances of actor $C$ are similar (symmetric)
- No change in latency!
Task Symmetry

- all instances of actor $C$ are similar (symmetric)
- No change in latency!
- Huge number of such symmetric solutions
Task Symmetry

- all instances of actor $C$ are similar (symmetric)
- **No change** in latency!
- **Huge number** of such symmetric solutions
- Add constraints to **eliminate** all but one
Task Symmetry

**Task Graph**

A directed acyclic graph representing the dependencies between tasks.

**A Schedule**

A list of tasks scheduled for execution on two processors.

- **Processor P1**:
  - Time sequence: B1, C10, C01, C00, D0, E0

- **Processor P2**:
  - Time sequence: A0, B0, C11, D1

**Lexicographic Order**

Enforce lexicographic order in the schedule.

- For u ≪ u′, s(u) ≤ s(u′)

- Example:
  - s(C00) ≤ s(C01) ≤ s(C10) ≤ s(C11)
**Task Symmetry**

- **Lexicographic order**: $C_{00} \ll C_{01} \ll C_{10} \ll C_{11}$
Motivation  
Application Model  
Deployment using SMT  
Symmetry elimination  
Distributed memory scheduling  
Design Tools  
Conclusions

Task Symmetry

- **lexicographic order**: $C_{00} \ll C_{01} \ll C_{10} \ll C_{11}$
- enforce lexicographic order in schedule:
  
  $s(u) \leq s(u')$ for $u \ll u'$

![Task graph](image)

a schedule

<table>
<thead>
<tr>
<th>A₀</th>
<th>B₀</th>
<th>C₁₁</th>
<th>D₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>B₁</td>
<td>C₁₀</td>
<td>C₀₁</td>
<td>C₀₀</td>
</tr>
</tbody>
</table>

time
Task Symmetry

- **lexicographic order**: $C_{00} \ll C_{01} \ll C_{10} \ll C_{11}$
- Enforce lexicographic order in schedule:
  $s(u) \leq s(u')$ for $u \ll u'$
  $s(C_{00}) \leq s(C_{01}) \leq s(C_{10}) \leq s(C_{11})$
Task Symmetry

**Lexicographic Order**: $C_{00} \ll C_{01} \ll C_{10} \ll C_{11}$

- Enforce lexicographic order in schedule: $s(u) \leq s(u')$ for $u \ll u'$
- $s(C_{00}) \leq s(C_{01}) \leq s(C_{10}) \leq s(C_{11})$
Task Symmetry: Theorem

Theorem: Every group has a lexicographic schedule.

Corollary: No feasible schedule is lost.
Task Symmetry: Theorem

Theorem: Every group has a lexicographic schedule.

Corollary: No feasible schedule is lost.
Task Symmetry: Theorem

**Theorem:** Every group has a lexicographic schedule

**Corollary:** No feasible schedule is lost

---

**a schedule**

<table>
<thead>
<tr>
<th>P₂</th>
<th>P₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>A₀</td>
<td>B₀</td>
</tr>
<tr>
<td>B₁</td>
<td>C₁₀</td>
</tr>
</tbody>
</table>

---

**a permuted schedule**

<table>
<thead>
<tr>
<th>P₂</th>
<th>P₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>A₀</td>
<td>B₀</td>
</tr>
<tr>
<td>B₁</td>
<td>C₁₀</td>
</tr>
</tbody>
</table>
Theorem: Every group has a **lexicographic schedule**

**Theorem:** Every group has a lexicographic schedule
Task Symmetry: Theorem

Theorem: Every group has a lexicographic schedule

Corollary: No feasible schedule is lost
Processor Symmetry
Processor Symmetry

Task graph

P1

P2

B1 C10 C11 D1

A0 B0 C00 C01 D0 E0

Time

schedule
Processor Symmetry

Task graph:

- A₀
- B₀
- C₀₀
- C₀₁
- D₀
- E₀
- A₁
- B₁
- C₁₀
- C₁₁
- D₁

Schedule:

P₁:

- A₀
- B₀
- C₀₀
- C₀₁
- D₀
- E₀

P₂:

- B₁
- C₁₀
- C₁₁
- D₁

Time

Swap P₁ and P₂
Processor Symmetry

task graph

[Diagram showing a task graph with nodes labeled A₀, B₀, C₀₀, D₀, E₀, A₁, B₁, C₀₁, D₁, E₁, C₁₀, C₁₁ and arrows indicating dependencies and flows.]

Schedule:

- **P₁**: A₀, B₀, C₀₀, C₀₁, D₀, E₀
- **P₂**: B₁, C₁₀, C₁₁, D₁

Swapped P₁ and P₂

[Diagram showing the swapped schedule with a red cross indicating the swapped order of tasks on P₁ and P₂.]
Exploration: Processors vs Latency $\alpha = 30$
Pareto Exploration

Exploration : Processors vs Latency $\alpha = 30$

without symmetry breaking
Pareto Exploration

Exploration: Processors vs Latency $\alpha = 30$

without symmetry breaking

with symmetry breaking
Pareto Exploration

**Exploration : Processors vs Latency** $\alpha = 30$

**Solver Performance**
- **Timeouts reduce**!
- The gap between SAT and UNSAT points is smaller.
Video Decoder

3D cost space \((C_L, C_P, C_B)\) exploration, \(C_B\) - total buffer size

MPEG video decoder:

122 Tasks
Video Decoder

3D cost space \((C_L, C_P, C_B)\) exploration, \(C_B\) - total buffer size

MPEG video decoder:

122 Tasks

![Diagram showing 3D cost space exploration for MPEG video decoder with and without symmetry constraints. The diagram includes nodes and edges with values for latency, buffer size, and processor count, highlighting better Pareto points with symmetry constraints.]
**Video Decoder**

**3D cost space** \((C_L, C_P, C_B)\) exploration, \(C_B\) - total buffer size

MPEG video decoder:

- Tasks: 122
- 3D cost space exploration
- Without symmetry constraints
- With symmetry constraints

![Diagram with 3D cost space and nodes](Image)
**Video Decoder**

**3D cost space** $(C_L, C_P, C_B)$ exploration, $C_B$ - total buffer size

MPEG video decoder:

- 122 Tasks

Better Pareto points
Video Decoder

**3D cost space** $\langle C_L, C_P, C_B \rangle$ exploration, $C_B$ - total buffer size

**MPEG video decoder:**

```
1500 20 150 20 1 100 1/20 3400
5 130
4 300
40 1 200
4 30
260 1 40

122 Tasks
```

Better Pareto points in same TIME-Budget!
So far we ignored the communication costs. For distributed memory, communication needs to be modeled.
So far we ignored the **communication** costs
Distributed memory scheduling

- So far we ignored the **communication** costs
- For **distributed memory**, communication needs to be modeled
Overview

1. Motivation
2. Application Model
3. Deployment using SMT
4. Symmetry elimination
5. Distributed memory scheduling
6. Design Tools
7. Conclusions
Kalray MPPA-256

- 16 compute clusters
Motivation

Application Model

Deployment using SMT

Symmetry elimination

Distributed memory scheduling

Design Tools

Conclusions

Kalray MPPA-256

- 16 compute clusters

Motivation

Application Model

Deployment using SMT

Symmetry elimination

Distributed memory scheduling

Design Tools

Conclusions
16 compute clusters

16 processors
Kalray MPPA-256

- 16 compute clusters
  - 16 processors
  - 2 MB Shared Memory
Kalray MPPA-256

- **16 compute clusters**
  - 16 processors
  - 2 MB Shared Memory
  - DMA
16 compute clusters

- 16 processors
- 2 MB Shared Memory
- DMA

Toroidal 2D network
Scheduling of Tasks and Communication is **not trivial.**

- **16 compute clusters**
  - **16 processors**
  - 2 MB Shared Memory
  - DMA
- Toroidal 2D network
Design Flow

- Application Graph
Design Flow

Application Graph

Partitioning

Group the Actors

Mapping/scheduling for many-core
Design Flow

Goals
- Load balance the groups
- Minimize data exchange
Design Flow

Application Graph

Partitioning

Placement

A
B
C
D
E
F

Place the Groups
Design Flow

Application Graph

Partitioning

Placement

Goals

Minimize distance between communicating groups

Place the Groups
Design Flow

Application Graph

Partitioning

Placement
Design Flow

Application Graph

Partitioning

Placement

Multi-cluster Scheduling

Schedule
- Tasks
- Transfer
Motivation Application Model Deployment using SMT Symmetry elimination Distributed memory scheduling Design Tools Conclusions

Design Flow

Application Graph

Partitioning

Placement

Multi-cluster Scheduling

Schedule

- Tasks
- Transfer

Goals

- Minimize Latency
- Minimize Buffer size
Output of Design Flow
Output of Design Flow

- Tasks and Transfers
Output of Design Flow

- Tasks and Transfers
  - Cluster Mapping
Output of Design Flow

- Tasks and Transfers
  - **Cluster** Mapping
  - **Processor** and **DMA** Mapping
Output of Design Flow

- Tasks and Transfers
  - Cluster Mapping
  - Processor and DMA Mapping
  - Start time
Output of Design Flow

- Tasks and Transfers
  - Cluster Mapping
  - Processor and DMA Mapping
  - Start time

- Edges
Output of Design Flow

- **Tasks and Transfers**
  - **Cluster** Mapping
  - **Processor** and **DMA** Mapping
  - **Start time**

- **Edges**
  - **Communication buffer** size
Output of Design Flow

- Tasks and Transfers
  - **Cluster** Mapping
  - **Processor** and **DMA** Mapping
  - **Start time**

- Edges
  - **Communication buffer** size

- Application
Output of Design Flow

- Tasks and Transfers
  - **Cluster** Mapping
  - **Processor** and **DMA** Mapping
  - **Start time**

- Edges
  - **Communication buffer** size

- Application
  - **Latency**
Tasks communicating via DMA:
Tasks communicating via DMA:
Tasks communicating via DMA:

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>Resources used</th>
<th>Task duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Initialization</td>
<td>Processor and DMA</td>
<td>Constant</td>
</tr>
</tbody>
</table>
Tasks communicating via DMA:

**Diagram:**

- **Cluster 0:**
  - P1
  - DMA0

- **Cluster 1:**
  - P1

**Task Description, Resources used, Task duration:**

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>Resources used</th>
<th>Task duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Initialization</td>
<td>Processor and DMA</td>
<td>Constant</td>
</tr>
<tr>
<td>G</td>
<td>Network Transfer</td>
<td>Only DMA</td>
<td>Transfer size dependent</td>
</tr>
</tbody>
</table>
An example application graph:
Model Transformation

An example application graph:

```
A -> B
[α, ω]
```

Buffer-Aware graph:

```
A

I

wr

G

wr

B

e
wt:
[1, w↑]

e
wn:
[1]

e
rt:
[α, ω]

e
ws:
[1]

e
wb:
[1, 0, b(ewt)]

e
rs:
[1]

e
rn:
[1]

e
rb:
[α - 1, 0, b(ert)]
```

DMA: Data flow-control

DMA: DMA-Completion
Model Transformation

An example application graph:

Partition-Aware graph:

\[
\begin{align*}
A & \xrightarrow{e_{wt} : [1, w^\uparrow]} I_{wr} & e_{wr} : [1] & \xrightarrow{e_{rt} : [\alpha, \omega]} B
\end{align*}
\]
Model Transformation

**An example application graph:**

Motivation  
Application Model  
Deployment using SMT  
Symmetry elimination  
Distributed memory scheduling  
Design Tools  
Conclusions

![Graph](image)

**Partition-Aware graph:**

![Partition-Aware Graph](image)

**Buffer-Aware graph:**

![Buffer-Aware Graph](image)
Model Transformation

An example application graph:

Partition-Aware graph:

Buffer-Aware graph:

DMA : Data

Tendulkar
Mapping/scheduling for many-core
Model Transformation

An example application graph:

Partition-Aware graph:

Buffer-Aware graph:

DMA : Data

DMA : flow-control
Model Transformation

An example application graph:

Partition-Aware graph:

Buffer-Aware graph:

DMA : Data

DMA-Completion

DMA : flow-control

Tendulkar

Mapping/scheduling for many-core
JPEG Decoder Example

VLD → IQ/IDCT → COLOR
12 → 12
1

VLD: Variable Length Decoder
IQ/IDCT: Inverse Quantization / Inverse Discrete Cosine Transform
COLOR: Color Conversion
JPEG Decoder Example

VLD : Variable Length Decoder
JPEG Decoder Example

VLD : Variable Length Decoder

IQ / IDCT : Inverse Quantization / Inverse Discrete Cosine Transform
JPEG Decoder Example

VLD : Variable Length Decoder
IQ / IDCT : Inverse Quantization / Inverse Discrete Cosine Transform
Color : Color Conversion
JPEG Decoder Example

Partitioning Solutions:

- $C_\tau$: Max. workload per group
- $C_\eta$: Total communication cost
- $C_z$: No. of Groups

JPEG Decoder

Partitioning

Placement

Multi-cluster Scheduling
JPEG Decoder Example

Partitioning Solutions:

- $C_\tau$: Max. workload per group
- $C_\eta$: Total communication cost
- $C_z$: No. of Groups

<table>
<thead>
<tr>
<th>Solution $P_s$</th>
<th>Allocated group</th>
<th>Exploration Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>vld</td>
<td>iq</td>
</tr>
<tr>
<td>$P_{s0}$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$P_{s1}$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$P_{s2}$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$P_{s3}$</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
### JPEG Decoder Example

#### Partitioning Solutions:

- $C_\tau$: Max. workload per group
- $C_\eta$: Total communication cost
- $C_z$: No. of Groups

#### Partitioning Solutions Table:

<table>
<thead>
<tr>
<th>Solution</th>
<th>Allocated group</th>
<th>Exploration Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>vld</td>
<td>iq</td>
</tr>
<tr>
<td>$P_{s0}$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$P_{s1}$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$P_{s2}$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$P_{s3}$</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
JPEG Decoder Example

Partitioning Solutions:

- $C_\tau$: Max. workload per group
- $C_\eta$: Total communication cost
- $C_z$: No. of Groups

### Table

<table>
<thead>
<tr>
<th>Solution</th>
<th>Allocated group</th>
<th>Exploration Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>vld iq color</td>
<td>$C_\tau$</td>
</tr>
<tr>
<td>$P_{s0}$</td>
<td>0 1 2</td>
<td>424012</td>
</tr>
<tr>
<td>$P_{s1}$</td>
<td>0 0 1</td>
<td>758116</td>
</tr>
<tr>
<td>$P_{s2}$</td>
<td>0 0 0</td>
<td>934288</td>
</tr>
<tr>
<td>$P_{s3}$</td>
<td>0 1 1</td>
<td>510276</td>
</tr>
</tbody>
</table>
JPEG Decoder Example

Scheduling Solutions:

Solution

\[ P_{s0} \]
\[ P_{s1} \]
\[ P_{s2} \]
\[ P_{s3} \]

Buffer Size (bytes)

Latency (cycles)

\[ \cdot 10^4 \]

\[ \cdot 10^6 \]
JPEG decoder latency on Kalray platform

- $P_{s0}$
- $P_{s1}$
- $P_{s2}$
- $P_{s3}$

- model
- measured-min.
- measured-max.
StreamIt Benchmarks

- JPEG Dec.
- Beam Former
- Insertion Sort
- Merge Sort
- Radix Sort
- Dct1
- Dct2
- Dct3
- Dct4
- Dct5
- Dct6
- Dct7
- Dct8
- Dct Coarse
- Dct Fine
- Comp. count
- Matrix Mult.
- Fit

#Solutions

- %error

Tendulkar

Mapping/scheduling for many-core
StreamIt Benchmarks

- JPEG Dec.
- Beam Former
- Insertion Sort
- Merge Sort
- Radix Sort
- Dct1
- Dct2
- Dct3
- Dct4
- Dct5
- Dct6
- Dct7
- Dct8
- Dct Coarse
- Dct Fine
- Comp. count
- Matrix Mult.
- Fft

#Solutions
%error
Overview

1. Motivation
2. Application Model
3. Deployment using SMT
4. Symmetry elimination
5. Distributed memory scheduling
6. Design Tools
7. Conclusions
Our Framework

Runtime
Our Framework

Runtime

Run-time
Our Framework

Runtime

XML \rightarrow Run-time
Our Framework

Runtime

Application Code

Run-time

XML
Our Framework

Runtime

Application Code

Run-time

XML

Tendulkar

Mapping/scheduling for many-core
Our Framework

Runtime

Application Code

Run-time

XML

FIFO
Our Framework

Runtime

- Application Code
- Run-time
- XML
- FIFO
- P₀
- P₁
Our Framework

Runtime

- Application Code
- Run-time
- XML

- FIFO
- $P_0$
- $P_1$
Our Framework

Runtime

Application Code

A ()

Run-time

XML

FIFO

P₀

P₁

Tendulkar

Mapping/scheduling for many-core
Our Framework

Runtime

XML

Application Code

A()

Run-time

FIFO

P0

P1
Our Framework

Runtime

Application Code

B ()

FIFO

P_0

P_1

XML

Run-time
Our Framework

Runtime

Application Code
B ()

Run-time

FIFO

P₀

P₁

XML
Our Framework

StreamExplorer
Our Framework

StreamExplorer

Run-time
Our Framework

StreamExplorer

XML Parser
- Split-join
- Platform

Run-time

profile
Our Framework

StreamExplorer

- XML Parser
  - Split-join
  - Platform

- Graph Models
  - Split-join
  - Task Graph

- Run-time
Our Framework

StreamExplorer

- XML Parser
  - Split-join
  - Platform

- Run-time

- Property Analyzer
  - Consistency
  - Cost bounds

- Graph Models
  - Split-join
  - Task Graph

- Output Generator
  - DotGraph
  - Cost-space Explorer
  - Z3 Solver
  - query
  - result

- Design Tools

- Conclusions
Our Framework

StreamExplorer

Property Analyzer
Consistency Cost bounds

XML Parser
Split-join Platform

Graph Models
Split-join Task Graph

Output Generator
DotGraph

Run-time

profile
Our Framework

StreamExplorer

- **Property Analyzer**
  - Consistency
  - Cost bounds

- **XML Parser**
  - Split-join
  - Platform

- **Graph Models**
  - Split-join
  - Task Graph

- **Cost-space Explorer**
  - Grid
  - Binary search

- **Run-time**
  - Output Generator
    - DotGraph

- **Profile**
Our Framework

StreamExplorer

- XML Parser
  - Split-join
  - Platform
- Property Analyzer
  - Consistency
  - Cost bounds
- Graph Models
  - Split-join
  - Task Graph
- Output Generator
  - DotGraph
- Cost-space Explorer
  - grid
  - binary search
- Z3 Solver

Run-time

Profile

---

Tendulkar
Mapping/scheduling for many-core
Our Framework

StreamExplorer

- XML Parser
  - Split-join
  - Platform
- Property Analyzer
  - Consistency
  - Cost bounds
- Graph Models
  - Split-join
  - Task Graph
- Cost-space Explorer
  - Grid
  - Binary search
- Z3 Solver
- Output Generator
  - DotGraph
  - Gantt Chart
  - Schedule XML
- Run-time
- Profile

Tendulkar
Mapping/scheduling for many-core
Our Framework

StreamExplorer

- **XML Parser**
  - Split-join
  - Platform

- **Property Analyzer**
  - Consistency
  - Cost bounds

- **Graph Models**
  - Split-join
  - Task Graph

- **Cost-space Explorer**
  - grid
  - binary search

- **Z3 Solver**

- **Output Generator**
  - DotGraph
  - Gantt Chart
  - Schedule XML

- **Run-time**

Query -> Result

Profile -> Schedule XML
Our Framework

- StreamExplorer

Written in Java
32k+ lines of Code.

Runtime
Written in C++
14k+ lines of Code.
Our Framework

StreamExplorer
- Written in Java
Our Framework

StreamExplorer

- Written in Java
- 32k+ lines of Code.
Our Framework

- **StreamExplorer**
  - Written in Java
  - 32k+ lines of Code.

- **Runtime**
  - Written in C++
Our Framework

- **StreamExplorer**
  - Written in Java
  - 32k+ lines of Code.

- **Runtime**
  - Written in C++
  - 14k+ lines of Code.
Overview

1. Motivation
2. Application Model
3. Deployment using SMT
4. Symmetry elimination
5. Distributed memory scheduling
6. Design Tools
7. Conclusions
Conclusions and Future Work

Conclusions:

Symmetry elimination finds better solutions.
Combined Optimization with Communication modeling.
Automated design flow for distributed memory.

Future Work:
Spread actor over multiple clusters.
Network route selection and scheduling.
Pipelined scheduling.
Scheduling under uncertainty.
Conclusions:

- **Symmetry elimination** finds better solutions
Conclusions:

- **Symmetry elimination** finds better solutions
- Combined Optimization with **Communication modeling**
Conclusions:

- **Symmetry elimination** finds better solutions
- Combined Optimization with **Communication modeling**
- **Automated design flow** for distributed memory

Future Work:
- Spread actor over multiple clusters
- Network route selection and scheduling
- Pipelined scheduling
- Scheduling under uncertainty
Conclusions and Future Work

Conclusions:

- **Symmetry elimination** finds better solutions
- Combined Optimization with **Communication modeling**
- **Automated design flow** for distributed memory

Future Work:
Conclusions and Future Work

Conclusions:

- **Symmetry elimination** finds better solutions
- Combined Optimization with **Communication modeling**
- **Automated design flow** for distributed memory

Future Work:

- Spread actor over **multiple clusters**
Conclusions:

- **Symmetry elimination** finds better solutions
- Combined Optimization with **Communication modeling**
- **Automated design flow** for distributed memory

Future Work:

- Spread actor over **multiple clusters**
- **Network route** selection and scheduling
Conclusions and Future Work

Conclusions:

- **Symmetry elimination** finds better solutions
- Combined Optimization with **Communication modeling**
- **Automated design flow** for distributed memory

Future Work:

- Spread actor over **multiple clusters**
- **Network route** selection and scheduling
- **Pipelined** scheduling
Conclusions:

- **Symmetry elimination** finds better solutions
- Combined Optimization with **Communication modeling**
- **Automated design flow** for distributed memory

Future Work:

- Spread actor over **multiple clusters**
- **Network route** selection and scheduling
- **Pipelined** scheduling
- Scheduling under **uncertainty**
Contributions


- P. Tendulkar and S. Stuijk. “A Case Study into Predictable and Composable MPSoC Reconfiguration”. In: IPDPS RAW Workshop. 2013


Questions?
Overview

8 SDF and Split Join graphs

9 Symmetry Breaking

10 Design Flow Details

11 DMA transfer granularity

12 Run-time Management
Hypothesis supported by StreamIt.\textsuperscript{1}

- Total 763 actors analyzed in various applications
  - 94\% are stateless
  - 6\% are stateful
    - 45\% have states due to algorithm
    - 55\% have avoidable states
- Odd rates exist but are rare

CD-DAT benchmark used as an example

Converts CD audio (44.1 kHz) to digital audio tape (48 kHz)

---

\textsuperscript{1} W. Thies and S. Amarasinghe. “An Empirical Characterization of Stream Programs and Its Implications for Language and Compiler Design”. In: PACT. 2010
Overview

- SDF and Split Join graphs
- Symmetry Breaking
- Design Flow Details
- DMA transfer granularity
- Run-time Management
Proof Sketch

modify a feasible schedule such that:
\[ s(v_0) \leq s(v_1) \leq s(v_2) \leq ... \]
prove that precedence constraints are satisfied
here: for neutral channels (\( \alpha = 1 \)), unfolded to \((v_h, v'_h)\)

lexicographic order

start-time compatible

new hier. index;
new precedence relation
Proof Sketch

modify a feasible schedule such that:
\( s(v_0) \leq s(v_1) \leq s(v_2) \leq \ldots \)

prove that precedence constraints are satisfied

here: for neutral channels (\( \alpha = 1 \)), unfolded to \((v_h, v'_h)\)

$\xrightarrow{\text{lexicographic order}}$

$\xrightarrow{\text{start-time compatible}}$

$\xrightarrow{\text{new hier. index; new precedence relation}}$
Proof Sketch

take successor $[j]$
Proof Sketch

take successor \([j]\)
take successor \([j]\) by definition there exist \(j + 1\) same or earlier successors
Proof Sketch

take successor $[j]$ by definition there exist $j + 1$ same or earlier successors
Proof Sketch

take successor \([j]\) by definition there exist \(j + 1\) same or earlier successors their original predecessors finish before successor \([j]\):
take successor \([ j ]\) by definition there exist \( j + 1 \) same or earlier successors

their original predecessors finish before successor \([ j ]\):
take successor $[j]$ by definition there exist $j + 1$ same or earlier successors their original predecessors finish before successor $[j]$: 
take successor \([ j ]\) by definition there exist \( j + 1 \) same or earlier successors

their original predecessors finish before successor \([ j ]\):

\( j + 1 \) predecessors finish before, hence the earliest \( j + 1 \) ones as well
Proof Sketch

Take successor \([ j ]\) by definition there exist \(j + 1\) same or earlier successors. Their original predecessors finish before successor \([ j ]\):

\(j + 1\) predecessors finish before, hence the earliest \(j + 1\) ones as well.

Predecessor \([ j ]\) finishes before successor \([ j ]\).
Overview

- SDF and Split Join graphs
- Symmetry Breaking
- Design Flow Details
- DMA transfer granularity
- Run-time Management
Design Flow

Application Graph

Partitioning

Placement

Multi-cluster Scheduling

max workload per group

#groups

estimated comm. cost

communication cost

minimal solution

(3D Pareto solutions)

communication cost

communication cost

(2D Pareto solutions)

comm. buffer size

latency

Tendulkar
Overview

- SDF and Split Join graphs
- Symmetry Breaking
- Design Flow Details
- DMA transfer granularity
- Run-time Management
Buffering Algorithm

For architectures with DMA and limited local memory
Data transfer granularity

Transfer Granularity less than optimal

Time

Transfer Granularity greater than optimal

Time

Mapping/scheduling for many-core
Data transfer granularity

Additional complexity with multiple processors

Output Transfer

Prologue

$Proc_2$

idle

$b_2$

$b_5$

$b_8$

$Proc_1$

idle

$b_1$

$b_4$

$b_7$

$Proc_0$

idle

$b_0$

$b_3$

$b_6$

Input Transfer

$Proc_2$

$b_0, b_1, b_2$

$Proc_1$

$b_3, b_4, b_5$

$Proc_0$

$b_6, b_7, b_8$

Time

Prologue

Epilogue

Mapping/scheduling for many-core

Tendulkar

DMA transfer granularity

Run-time Management

Design Flow Details

Symmetry Breaking

SDF and Split Join graphs

11 / 20
DMA transfer granularity optimization

\[ I(s) \]
\[ C(s) \]
\[ T_p(s) \]
\[ T_1(s) \]
\[ \nu(s) \]

Transfer / computation time per block

Total Exec time

Transfer Regime
Computation Regime

Transfer Domain
Computation Domain

local mem. size
block size

\[ s_1^* \]
\[ s_p^* \]
Characterization of DMA of IBM Cell B.E.:

- **Time to read/write block**
  - Graph showing time in clock cycles vs. block size for 1 SPU, 2 SPU, 4 SPU, and 8 SPU.

- **Cost per byte**
  - Graph showing cost per byte vs. block size for 1 SPU, 2 SPU, 4 SPU, and 8 SPU.
Experimental Evaluation

Synthetic Application Benchmark:

![Graph showing execution time vs block size for different SPU (Single Processing Unit) configurations.]

- Execution Time (clock cycles)
- Block size (bytes)
- 2 SPU-pred
- 4 SPU-pred
- 8 SPU-pred
- 2 SPU-meas
- 4 SPU-meas
- 8 SPU-meas
Overview

8. SDF and Split Join graphs
9. Symmetry Breaking
10. Design Flow Details
11. DMA transfer granularity
12. Run-time Management
The context

- Multiple configurations for each application
- Applications start / stop dynamically
- **How to:**
  - select a configuration for each application?
  - re-configure the applications?
Features:

- CompOSe real-time operating system
- Predictable Æthereal network-on-chip
- TDM application scheduling for composability
- **composable**: The changes in an application don’t affect other running applications
Resource Manager conceptual view

Resource manager Design:

- System RM: takes re-configuration decisions
- Application RM: implements re-configuration decisions
Resource Manager on the platform

Resource manager Implementation:

- **System RM**: is a separate application
- **Application RM**:
  - organized in master-slave(s) configuration
  - is a part of user application
Experiment with JPEG Decoder

Steps for Re-configuration:

1. Instruct application RM to reconfigure
2. Request removal of application from TDM
3. Remove application from TDM and ack.
4. Resize TDM allocation
5. Add new FIFO(s)
6. Remove old FIFO(s)
7. Add application to TDM
8. Inform system RM about completion