Symmetry Breaking for Multi-Criteria Mapping and Scheduling on Multicores

Pranav Tendulkar  Peter Poplavko  Oded Maler

Verimag, FRANCE

August 2013
Typical in parallel programming: spawn multiple identical tasks

- data parallelism
- obtain hyperperiod of a multi-periodic system
- duplicate tasks for fault-tolerance
Typical in parallel programming: spawn multiple identical tasks
- data parallelism
- obtain hyperperiod of a multi-periodic system
- duplicate tasks for fault-tolerance

Often the platform have multiple identical processors.
Typical in parallel programming: spawn multiple identical tasks
- data parallelism
- obtain hyperperiod of a multi-periodic system
- duplicate tasks for fault-tolerance

Often the platform have multiple identical processors.

Hence, symmetry in the solution space.
Multi-criteria Optimization

minimize latency using minimal number of processors

![Diagram showing a network with nodes labeled A, B, C, D, and E, with arrows indicating connections and latency values.]
Motivation

Multi-criteria Optimization

minimize latency using minimal number of processors
Motivation

Contribution

context:

static mapping and scheduling for programs with data-parallelism
multi-criteria optimization using SMT solvers
Motivation

Contribution

classification:
stead mapping and scheduling for programs with data-parallelism
multi-criteria optimization using SMT solvers

symmetry breaking in solution space for identical tasks and processors
Motivation

**Motivation**

**Contribution**

**context:**

static mapping and scheduling for programs with data-parallelism
multi-criteria optimization using **SMT solvers**

**symmetry breaking** in solution space for identical tasks and processors

**goal:** increase the tractable problem size of SMT solvers

**experiments** : problem size increase from 20 to 50 tasks
Outline

1. Motivation
2. Application Model
3. Problem Formulation - SMT
4. Symmetry Breaking
5. Cost Space Exploration
6. Experiments and Results
7. Conclusions
Outline

1. Motivation
2. Application Model
3. Problem Formulation - SMT
4. Symmetry Breaking
5. Cost Space Exploration
6. Experiments and Results
7. Conclusions
synchronous dataflow graphs (SDF)
   by E. Lee and D. Messerschmitt in 1987
   task graph + symbolic representation of data parallelism
   signal-processing, video-coding applications

a ‘standard’ in academic multicore compilers:
   StreamIt compiler of MIT
synchronous dataflow graphs (SDF) 
by E. Lee and D. Messerschmitt in 1987

- task graph + symbolic representation of data parallelism
- signal-processing, video-coding applications

a ‘standard’ in academic multicore compilers:
- StreamIt compiler of MIT

we introduce split-join graphs : restriction of SDF

- still covering perhaps 90% of use cases
Split-Join Graphs

A simple split-join graph example:

\[ A \xrightarrow{\alpha} B \xrightarrow{1/\alpha} C \]

\( \alpha \): spawn and split

\( 1/\alpha \): wait and join
Definition (Split-Join Graph)

\[ S = (V, E, d, \alpha), (V, E): \text{DAG}, \quad V: \text{actors}, \ E: \text{channels} \]

\[ d: V \rightarrow \mathbb{R}_+: \text{actor execution time}, \]

\[ \alpha: E \rightarrow \mathbb{Q}: \text{channel counter: split} \ (> 1), \ \text{join} \ (< 1) \ \text{or neutral} \ (= 1) \]
Split-Join Graphs

Definition (Split-Join Graph)

\[ S = (V, E, d, \alpha), (V, E) : \text{DAG}, \quad V: \text{actors, } E: \text{channels} \]
\[ d : V \to \mathbb{R}_+ : \text{actor execution time,} \]
\[ \alpha : E \to \mathbb{Q} : \text{channel counter: split (}> 1\), join (< 1\) or neutral (= 1) \]
Split-Join Graphs

Definition (Split-Join Graph)

\[ S = (V, E, d, \alpha), \quad (V, E) : \text{DAG}, \quad V: \text{actors, } E: \text{channels} \]

\[ d : V \rightarrow \mathbb{R}_+ : \text{actor execution time,} \]

\[ \alpha : E \rightarrow \mathbb{Q} : \text{channel counter: split (> 1), join (< 1) or neutral (= 1)} \]

**balanced-parenthesis:** 

\( (1,3,2,1/2,1/3) \)
Well-behaved Graphs

Definition (Well-behaved)

\[ S = (V, E, d, \alpha) \] is well-behaved if any complete path has balanced-parenthesis signature

Such a graph can be unfolded to a task graph.
Unfolding to Task Graph

A → B → C

A → B → D → E → F → G
Unfolding to Task Graph
Unfolding to Task Graph
Unfolding to Task Graph

Application Model

Tendulkar, Poplavko, Maler

Symmetry Breaking for mapping/scheduling
Unfolding to Task Graph

Application Model

Tendulkar, Poplavko, Maler

Symmetry Breaking for mapping/scheduling
Unfolding to Task Graph

Application Model
Unfolding to Task Graph

Application Model

Symmetry Breaking for mapping/scheduling
Unfolding to Task Graph

Application Model

Symmetry Breaking for mapping/scheduling

Tendulkar, Poplavko, Maler
Unfolding to Task Graph
Unfolding to Task Graph
Actors, Tasks, Lexicographic Order

split-join graph: actors e.g., A, B, C

[Diagram of a split-join graph with nodes labeled A, B, C, D, E, F, G and arrows indicating connections and labels for order and splits/joins.]

notation for actors: \( v, v \in V \)

notation for tasks: \( u \in U \)

\( u \vdash u = \{ v_h \} \): lexicographically ordered (\( \ll \)) set of instances of \( v \)

\( E_0 \ll E_0 \ll E_1 \ll E_1 \ll E_2 \ll E_2 \)
Actors, Tasks, Lexicographic Order

notation for actors: $v, v \in V$

![Diagram showing a graph of actors and tasks with lexicographic order relations]
Actors, Tasks, Lexicographic Order

notation for actors: \( v, v \in V \)

unfolded task graph: tasks e.g., \( E_{0,1}, B, C_2 \)
Actors, Tasks, Lexicographic Order

notation for actors: $v, v \in V$

notation for tasks: $u \in U$
Application Model

Actors, Tasks, Lexicographic Order

notation for actors: \( v, v \in V \)

notation for tasks: \( u \in U \)

\( u = v_h, v \in V \) and \( h \) - hier. index, e.g., \( v_h = E_{0,1} \)
Actors, Tasks, Lexicographic Order

notation for actors: \( v, v \in V \)

\[
\begin{align*}
A & \xrightarrow{1} B & 1 \\
B & \xrightarrow{3} C & 2 \\
C & \xrightarrow{3} D & 1/2 \\
D & \xrightarrow{2} E & 1/3 \\
E & \xrightarrow{1/3} F \\
F & \xrightarrow{1/3} G
\end{align*}
\]

notation for tasks: \( u \in U \)

\( U_v = \{v_h\} : \text{lexicographically ordered (\( \ll \)) set of instances of } v \)

\( U_E : E_{0,0} \ll E_{0,1} \ll E_{1,0} \ll E_{1,1} \ll E_{2,0} \ll E_{2,1} \)
Outline

1. Motivation
2. Application Model
3. Problem Formulation - SMT
4. Symmetry Breaking
5. Cost Space Exploration
6. Experiments and Results
7. Conclusions
Multi-criteria Optimization Strategy

Given a split-join graph $S$, we perform the following steps:

1. Check whether $S$ is well-behaved
2. Unfold $S$ into task graph $T = (U, E, \delta)$
3. Generate the mapping and scheduling constraints:
   - Precedence
   - Mutual Exclusion
   - Buffer Capacity
   (Extended Problem - see the paper)

Decision variables:
- $\mu(u)$, $u \in U$ - the mapping: processor (1, 2, ..., $M$) for $u$
- $s(u)$ - the schedule: start time of $u$
Multi-criteria Optimization Strategy

Given a split-join graph $S$, we perform the following steps:

1. Check whether $S$ is well-behaved
2. Unfold $S$ into task graph $T = (U, \mathcal{E}, \delta)$
Multi-criteria Optimization Strategy

Given a split-join graph $S$, we perform the following steps:

1. Check whether $S$ is well-behaved
2. Unfold $S$ into task graph $T = (U, E, \delta)$
3. Generate the mapping and scheduling constraints:
   - Precedence
   - Mutual Exclusion
   - Buffer Capacity
Multi-criteria Optimization Strategy

Given a split-join graph $S$, we perform the following steps:

1. Check whether $S$ is well-behaved
2. Unfold $S$ into task graph $T = (U, E, \delta)$
3. Generate the mapping and scheduling constraints:
   - Precedence
   - Mutual Exclusion
   - Buffer Capacity (Extended Problem - see the paper)
Given a split-join graph $S$, we perform the following steps:

1. Check whether $S$ is well-behaved
2. Unfold $S$ into task graph $T = (U, E, \delta)$
3. Generate the mapping and scheduling constraints:
   - Precedence
   - Mutual Exclusion
   - Buffer Capacity (Extended Problem - see the paper)

Decision variables:
Multi-criteria Optimization Strategy

Given a split-join graph $S$, we perform the following steps:

1. Check whether $S$ is well-behaved
2. Unfold $S$ into task graph $T = (U, \mathcal{E}, \delta)$
3. Generate the mapping and scheduling constraints:
   - Precedence
   - Mutual Exclusion
   - Buffer Capacity (Extended Problem - see the paper)

Decision variables:

- $\mu(u), u \in U$ - the mapping: processor $(1,2,\ldots,M)$ for $u$
Multi-criteria Optimization Strategy

Given a split-join graph $S$, we perform the following steps:

1. Check whether $S$ is well-behaved
2. Unfold $S$ into task graph $T = (U, E, \delta)$
3. Generate the mapping and scheduling constraints:
   - Precedence
   - Mutual Exclusion
   - Buffer Capacity (Extended Problem - see the paper)

Decision variables:

- $\mu(u), u \in U$ - the mapping: processor $(1,2,\ldots,M)$ for $u$
- $s(u)$ - the schedule: start time of $u$
Constraints

Predicate $\varphi(u, u')$: task $u'$ starts after the completion of task $u$

$$\varphi(u, u') : s(u') \geq s(u) + \delta(u)$$
Constraints

Predicate $\varphi(u, u')$:

Task $u'$ starts after the completion of task $u$

$$\varphi(u, u') : s(u') \geq s(u) + \delta(u)$$

Precedence:

$$\bigwedge_{(u, u') \in E} \varphi(u, u')$$
Problem Formulation - SMT

Constraints

Predicate $\varphi(u, u')$:

task $u'$ starts after the completion of task $u$

$$\varphi(u, u') : s(u') \geq s(u) + \delta(u)$$

Precedence:

$$\bigwedge_{(u, u') \in E} \varphi(u, u')$$

Mutual exclusion:

$$\bigwedge_{u \neq u' \in U} (\mu(u) = \mu(u')) \Rightarrow \varphi(u, u') \lor \varphi(u', u)$$
Outline

1. Motivation
2. Application Model
3. Problem Formulation - SMT
4. Symmetry Breaking
5. Cost Space Exploration
6. Experiments and Results
7. Conclusions
Task Symmetry

- all instances of given actor $v$ are similar (symmetric)
Task Symmetry

- all instances of given actor $v$ are similar (symmetric)
Task Symmetry

- All instances of given actor \( v \) are similar (symmetric)
- Permutation of symmetric tasks does not change the latency,
- ... but extends the solution space exponentially
Task Symmetry

- **Symmetry Breaking**

- **Task Symmetry**

  

  - Task graph

  - Schedule:

    - Schedule compatible with lexicographic order:
      \[ s(C_{00}) \leq s(C_{01}) \leq s(C_{10}) \leq s(C_{11}) \]

  - Theorem: Adding constraints 
    \[ s(u) \leq s(u') \] 
    for \( u < u' \) does not eliminate optimality.

  - **References:**
    - Tendulkar, Poplavko, Maler
      - Symmetry Breaking for mapping/scheduling
**Task Symmetry**

- **enforce the schedule to be compatible with lexicographic order:**
  \[ s(C_{00}) \leq s(C_{01}) \leq s(C_{10}) \leq s(C_{11}) \]
Task Symmetry

- enforce the schedule to be compatible with lexicographic order:
  \[ s(C_{00}) \leq s(C_{01}) \leq s(C_{10}) \leq s(C_{11}) \]
- **Theorem**: adding constraints \( s(u) \leq s(u') \) for \( u \ll u' \) does not eliminate optimality

Symmetry Breaking for mapping/scheduling
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 (\( v_h, v'_h \))

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 \((v_h, v'_h)\)

lexicographic order

start-time compatible

new hier. index;
new precedence relation

Tendulkar, Poplavko, Maler
Symmetry Breaking for mapping/scheduling
Proof Sketch

[Diagram showing a sequence of numbers with arrows indicating predecessor-successor relationships.]

take successor $[j]$
Proof Sketch

\[ \begin{array}{c}
0 & 0 \\
1 & 1 \\
2 & 2 \\
3 & 3 \\
4 & 4 \\
5 & 5 \\
\end{array} \]

take successor \([ j ]\)

\[ \begin{array}{c}
1[0] & 2[0] \\
0[1] & 1[1] \\
\end{array} \]

\([ j ] = [3] \)
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
take successor \([j]\)
by definition there exist \(j + 1\) same or earlier successors
their original predecessors finish before successor \([j]\):
Proof Sketch

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

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

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1[0]</td>
<td>2[0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0[1]</td>
<td>1[1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4[3]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 \([j]\)
Processor Symmetry

task graph

![Diagram]

Schedule:

<table>
<thead>
<tr>
<th>P3</th>
<th></th>
<th>C_{01}</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>B_1</td>
<td>C_{10}</td>
</tr>
<tr>
<td>P1</td>
<td>A</td>
<td>B_0</td>
</tr>
</tbody>
</table>

Time

Swap P1 and P2

![Swapped Diagram]
Outline

1 Motivation
2 Application Model
3 Problem Formulation - SMT
4 Symmetry Breaking
5 Cost Space Exploration
6 Experiments and Results
7 Conclusions
Exploring the Design Space

One SMT query for a given point \((C_L, C_M)\) in the cost space:

- \(C_L\) - latency
- \(C_M\) - processor count

\[\text{sat points} \quad \text{unsat points} \quad \text{unexplored points}\]
Exploring the Design Space

One SMT query for a given point \((C_L, C_M)\) in the cost space:

- \(C_L\) - latency
- \(C_M\) - processor count

Precedence and Mutual Exclusion Constraints

Cost Constraints

\[
\bigwedge_{u \in U} s(u) + \delta(u) \leq C_L \land \bigwedge_{u \in U} \mu(u) \leq C_M
\]
Outline

1. Motivation
2. Application Model
3. Problem Formulation - SMT
4. Symmetry Breaking
5. Cost Space Exploration
6. Experiments and Results
7. Conclusions
Synthetic-Graph Experiments

- Fix processor cost $C_M$ and perform binary search for optimal $C_L$
- Increase $\alpha$ and measure increase in computation time
- With(out) breaking of task symmetry and processor symmetry
Synthetic-Graph Experiments

- Fix processor cost $C_M$ and perform binary search for optimal $C_L$
- Increase $\alpha$ and measure increase in computation time
- With(out) breaking of task symmetry and processor symmetry
- Z3 solver v4.1 on i7 core at 1.73GHz
Experiments and Results

Synthetic-Graph Experiments

Exploration Time (seconds)

\( \alpha \)

0 10 20 30 40 50

0 1,000

timeout

- - no sym
- - task sym
- - proc sym
- - task & proc sym

5-processor deployments
Pareto Exploration

without symmetry breaking

cost space \((C_L, C_M)\) exploration for \(\alpha = 30\)

evaluate task and processor symmetry breaking
Pareto Exploration

without symmetry breaking

with symmetry breaking

cost space \((C_L, C_M)\) exploration for \(\alpha = 30\)
evaluate task and processor symmetry breaking
Video Decoder

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

MPEG video decoder:
Experiments and Results

Video Decoder

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

MPEG video decoder:

- \(1500\) to \(20\) to \(150\) to \(100\) to \(3400\)
- \(130\) to \(4\) to \(300\) to \(200\) to \(30\)
- \(40\) to \(4\) to \(260\) to \(40\)

- Without symmetry constraints
- With symmetry constraints

Latency (\(10^3\))

- \(8\)
- \(12\)
- \(16\)
- \(20\)
- \(24\)
- \(28\)
- \(32\)
- \(36\)
- \(40\)

Buffer Size

- \(150\)
- \(200\)
- \(250\)
- \(300\)
- \(350\)

Processor

- \(0\)
- \(20\)
- \(40\)
- \(60\)
- \(80\)
- \(100\)
- \(120\)
- \(140\)

Tendulkar, Poplavko, Maler

Symmetry Breaking for mapping/scheduling
Conclusions

- Symbolic representation of data-parallel programs
  - a useful subclass of SDF model
- Framework for multi-criteria optimal deployment
- Symmetry breaking: prove task symmetry and use processor symmetry

Future work:
- More symmetry breaking, also approximation and heuristics
- More refined data communication: data transfer delays
- Pipelined scheduling
- Scheduling under uncertainty
- Multistage design flow
Conclusions

- Symbolic representation of data-parallel programs
  - a useful subclass of SDF model
- Framework for **multi-criteria optimal deployment**
- Symmetry breaking: prove task symmetry and use processor symmetry
- Future work:
Conclusions

- Symbolic representation of data-parallel programs
  - a useful subclass of SDF model
- Framework for multi-criteria optimal deployment
- Symmetry breaking: prove task symmetry and use processor symmetry
- Future work:
  - More symmetry breaking, also approximation and heuristics
  - More refined data communication: data transfer delays
  - Pipelined scheduling
  - Scheduling under uncertainty
  - Multistage design flow
QUESTIONS?