Many-Core Scheduling of Data Parallel Applications using SMT Solvers

Pranav Tendulkar    Peter Poplavko    Ioannis Galanommmatis    Oded Maler

Verimag, FRANCE

August 2014
Multi-core Processors Everywhere

- Tablets
- Phones
- Space-shuttle
- Cars
- Smart-TV
- Laptops
- Cameras

Tendulkar, Poplavko, Galanommatis, Maler

Mapping/scheduling for many-core
Mapping and Scheduling solutions is exponential

Many-core platforms involve extra complexity factors

- Explicit modeling of network communication is necessary
- Orchestration of processor and network resources is non-trivial
Design Problems

How to:

- Maximize the performance of the application
- Optimally utilize memory resources
- Orchestrate shared resources such as Processors, DMA etc.
  - Load balance the processors
  - Minimize communication costs
  - Schedule tasks in parallel sharing limited resources
Overview

1. Motivation
2. Application Model
3. Hardware Platform
4. Scheduling
5. Experiments
6. 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 use split-join graphs : restriction of SDF

still covering perhaps 90% of use cases
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 use split-join graphs : restriction of SDF

still covering perhaps 90% of use cases

a simple split-join graph example:

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

\( \alpha \) : spawn and split

\( 1/\alpha \) : wait and join
Overview

1. Motivation
2. Application Model
3. Hardware Platform
4. Scheduling
5. Experiments
6. Conclusions
Many-core platform = network of clusters
Many-core platform = network of clusters

Efficient orchestration of network communication and cluster scheduling is non-trivial.
Platform characteristics

- 16 symmetric processors in a cluster
- Shared Memory within a cluster (2 MB)
- Data cache 8KB per core (disabled)
- Inter-cluster communication using DMA and NoC
- NoC with Toroidal 2D topology
Overview

1. Motivation
2. Application Model
3. Hardware Platform
4. Scheduling
5. Experiments
6. Conclusions
Design Flow

Application Graph
Design Flow

Application Graph

Partitioning

max workload per group

#groups

estimated comm. cost

(3D Pareto solutions)
Design Flow

Partitioning

**Output:**
- Application graph partitioned into groups

![Diagram of Application Graph Partitioning](image)
Design Flow

Partitioning

Output:
- Application graph partitioned into groups

Goals:
- Load balance the groups
- Minimize communication between groups
Partitioning

Output:
- Application graph partitioned into groups

Goals:
- Load balance the groups
- Minimize communication between groups

Problem Inputs:
- Application Graph
- Hardware Architecture Model
Design Flow

Application Graph

Partitioning

max workload per group

#groups

estimated comm. cost

(3D Pareto solutions)
Design Flow

Application Graph

Partitioning

Placement

max workload per group

#groups

estimated comm. cost

minimal solution

communication cost

(3D Pareto solutions)
Design Flow

**Placement**

**Output:**
- Group to platform cluster assignment
Design Flow

**Placement**

**Output:**
- Group to platform cluster assignment

**Goals:**
- Place communicating groups on closely located hardware clusters
### Design Flow

**Placement**

**Output:**
- Group to platform cluster assignment

**Goals:**
- Place communicating groups on closely located hardware clusters

**Problem Inputs:**
- Application Graph
- Hardware Architecture Model
- Partitioning scheme

![Diagram of Application Graph and Placement](image-url)
Design Flow

Application Graph

Partitioning

Placement

max workload per group

#groups

estimated comm. cost

minimal solution

communication cost

(3D Pareto solutions)
Design Flow

- Application Graph
- Partitioning
- Placement
- Multi-cluster Scheduling

Scheduling

- max workload per group
- estimated comm. cost
- minimal solution
- communication cost
- latency
- comm. buffer size

(3D Pareto solutions)
(2D Pareto solutions)
Design Flow

Scheduling

Output:
- A mapping of every task to a processor or DMA channel
- Start time for every task
- Comm. buffer size per channel

Output of mapping and scheduling:
- A mapping of every task to a processor or DMA channel
- Start time for every task
- Comm. buffer size per channel

Cluster 0:
- P1
- DMA0

Cluster 1:
- P2
- P1

Time:
- P1
- P2

Tasks: A0, B0, C0, D0, E0, F0

Buffers: fifo, tx

Diagram: Nodes and arrows representing task dependencies and scheduling decisions.
**Design Flow**

**Scheduling**

**Output:**
- A mapping of every task to a processor or DMA channel
- Start time for every task
- Comm. buffer size per channel

**Goals:**
- Minimize application latency
- Minimize comm. buffer space
**Design Flow**

### Scheduling

**Output:**
- A mapping of every task to a processor or DMA channel
- Start time for every task
- Comm. buffer size per channel

**Goals:**
- Minimize application latency
- Minimize comm. buffer space

**Problem Inputs:**
- partitioning solution
- placement solution
- hardware architecture model
Design Space Exploration

- Motivation
- Application Model
- Hardware Platform
- Scheduling
- Experiments
- Conclusions

SMT Constraints

Design Space Exploration Algorithm

scheduling
placement
partition

SMT Solver

cost constraints

SMT Constraints

problem constraints

solutions

Tendulkar, Poplavko, Galanommati, Maler

Mapping/scheduling for many-core
Design Space Exploration

SMT Constraints → SMT Solver → SAT → solutions

Design Space Exploration Algorithm

Scheduling

Motivation Application Model Hardware Platform Scheduling Experiments Conclusions

Tendulkar, Poplavko, Galanommati, Maler

Mapping/scheduling for many-core
Design Space Exploration

- SMT Constraints
- Design Space Exploration Algorithm
- SMT Solver

Mapping/scheduling for many-core
Design Space Exploration

Motivation Application Model Hardware Platform Scheduling Experiments Conclusions

SMT Constraints

Design Space Exploration Algorithm

cost constraints

SMT Solver

solutions

scheduling

placement

partition

SMT Constraints

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

- \(C_L\) - latency
- \(C_B\) - comm. buffer

\[
\begin{array}{c}
\text{sat points} & \text{unsat points} & \text{unexplored points}
\end{array}
\]
Tasks communicating via DMA:

- **A**
- **I**
- **G**
- **B**

<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:

\[ \hat{e} : [\alpha(\hat{e}), \omega(\hat{e})] \]
An example application graph:

\[ \hat{e} : [\alpha(\hat{e}), \omega(\hat{e})] \]

**Model Transformation**
Model Transformation

An example application graph:

\[ \hat{e} : [\alpha(\hat{e}), \omega(\hat{e})] \]

Partition-Aware graph:

\[ e_{wt}(\hat{e}) : [1, w^\uparrow(\hat{e})], \quad e_{wn}(\hat{e}) : [1], \quad e_{rt}(\hat{e}) : [\alpha(\hat{e}), \omega(\hat{e})] \]
Model Transformation

An example application graph:

![Diagram of an application graph showing nodes A and B with edges labeled with time intervals.]

Partition-Aware graph:

![Diagram of a partition-aware graph with nodes A, I, G, F, St, Gsd, Lrd, and B, showing edges with time intervals.]

Buffer-Aware graph:

![Diagram of a buffer-aware graph with nodes A, I, G, F, St, Gsd, Lrd, and B, showing edges with time intervals.]
VLD : Variable Length Decoder
IQ / IDCT : Inverse Quantization / Inverse Discrete Cosine Transform
Color : Color Conversion
JPEG Decoder

Partitioning Solutions:

<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>$C_T$  $C_\eta$  $C_Z$</td>
</tr>
<tr>
<td>$P_{s0}$</td>
<td>0  1  2</td>
<td>424012  12384  3</td>
</tr>
<tr>
<td>$P_{s1}$</td>
<td>0  0  1</td>
<td>758116  2736  2</td>
</tr>
<tr>
<td>$P_{s2}$</td>
<td>0  0  0</td>
<td>934288  0  1</td>
</tr>
<tr>
<td>$P_{s3}$</td>
<td>0  1  1</td>
<td>510276  9648  2</td>
</tr>
</tbody>
</table>

$C_T$ : Maximum workload per group  
$C_\eta$ : Total communication cost  
$C_Z$ : Number of Groups
**Partitioning Solutions**

<table>
<thead>
<tr>
<th>Solution</th>
<th>Allocated Group</th>
<th>Exploration Cost</th>
<th>Allocation Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{s0}$</td>
<td>0</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>$P_{s1}$</td>
<td>0</td>
<td>0.5</td>
<td>0.7</td>
</tr>
<tr>
<td>$P_{s2}$</td>
<td>0</td>
<td>0.6</td>
<td>0.8</td>
</tr>
<tr>
<td>$P_{s3}$</td>
<td>0</td>
<td>0.7</td>
<td>0.9</td>
</tr>
</tbody>
</table>

$C_\tau$: Maximum workload per group
$C_\eta$: Total communication cost
$C_z$: Number of Groups

**Scheduling Solutions**

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

- Latency (cycles)
- Buffer Size (bytes)
Measurements on the Kalray processor

JPEG decoder latency measured on Kalray platform
## Other Applications

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#Actors</th>
<th>#Channels</th>
<th>#Tasks</th>
<th>Total Exec. Time (cycles)</th>
<th>Total Comm. Data (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG Decoder</td>
<td>3</td>
<td>2</td>
<td>25</td>
<td>934288</td>
<td>12384</td>
</tr>
<tr>
<td>Beam Former</td>
<td>8</td>
<td>7</td>
<td>53</td>
<td>342816</td>
<td>944</td>
</tr>
<tr>
<td>Insertion Sort</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>40033</td>
<td>320</td>
</tr>
<tr>
<td>Merge Sort</td>
<td>12</td>
<td>11</td>
<td>31</td>
<td>102347</td>
<td>704</td>
</tr>
<tr>
<td>Radix Sort</td>
<td>13</td>
<td>12</td>
<td>13</td>
<td>85464</td>
<td>768</td>
</tr>
<tr>
<td>Dct1</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>127496</td>
<td>768</td>
</tr>
<tr>
<td>Dct2</td>
<td>7</td>
<td>6</td>
<td>21</td>
<td>215525</td>
<td>1536</td>
</tr>
<tr>
<td>Dct3</td>
<td>5</td>
<td>4</td>
<td>12</td>
<td>129105</td>
<td>1024</td>
</tr>
<tr>
<td>Dct4</td>
<td>7</td>
<td>6</td>
<td>21</td>
<td>183890</td>
<td>1536</td>
</tr>
<tr>
<td>Dct5</td>
<td>7</td>
<td>6</td>
<td>21</td>
<td>216079</td>
<td>1536</td>
</tr>
<tr>
<td>Dct6</td>
<td>8</td>
<td>7</td>
<td>36</td>
<td>258304</td>
<td>1792</td>
</tr>
<tr>
<td>Dct7</td>
<td>8</td>
<td>7</td>
<td>29</td>
<td>218577</td>
<td>1792</td>
</tr>
<tr>
<td>Dct8</td>
<td>10</td>
<td>9</td>
<td>38</td>
<td>272514</td>
<td>2304</td>
</tr>
<tr>
<td>DctCoarse</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>74401</td>
<td>512</td>
</tr>
<tr>
<td>DctFine</td>
<td>6</td>
<td>5</td>
<td>20</td>
<td>163708</td>
<td>1280</td>
</tr>
<tr>
<td>Comparison Count</td>
<td>5</td>
<td>5</td>
<td>20</td>
<td>141397</td>
<td>1280</td>
</tr>
<tr>
<td>Matrix multiplication</td>
<td>11</td>
<td>11</td>
<td>79</td>
<td>1087840</td>
<td>10656</td>
</tr>
<tr>
<td>Fft</td>
<td>13</td>
<td>12</td>
<td>96</td>
<td>640109</td>
<td>6144</td>
</tr>
</tbody>
</table>
Other Applications

- 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

The graph shows the number of solutions (#Solutions) and the percentage of error (%error) for various applications. The applications are listed on the x-axis, and the y-axis represents the number of solutions and percentage of error. The graph indicates the mapping/scheduling for many-core architectures.
Other Applications

- 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
Conclusions:

- Automated design flow using SMT solvers
- Communication tasks for modeling explicit DMA communication
- Many-core scheduling of tasks on Processors and DMA

Contributions:
Conclusions

Contributions:

- Automated design flow using SMT solvers
- Communication tasks for modeling explicit DMA communication
- Many-core scheduling of tasks on Processors and DMA

Future Work:

- Spreading task instances of an actor over multiple clusters
- Network route selection and communication scheduling
- Pipelined scheduling on the platform
Questions?