Library Support in an Actor-based Parallel Programming Platform

Hae-woo Park and Soonhoi Ha

Co-design and Parallel-processing Lab.
Seoul National University
Table of Contents

Introduction
- "Programming Platform"-based Design
- Actor-based Modeling
- Design Restrictions

Library Support
- Specification
- Mapping
- Code Generation

Experiments

Conclusion
Introduction

Background: MPSoC becomes a promising solution.

- Evolution of semiconductor process technology
  - Possible to integrate tens or hundreds of processors in a single chip
- Problem
  - Utilization of a large amount of cores
  - Development of parallel embedded software

Parallel Software Design Methods

- Compiler-based approaches
  - ex) MAPS project from Aachen Univ.
- Language-based approaches
  - ex) OpenMP for IBM Cell, CUDA for Nvidia GPGPU
- Model-based approaches
  - ex) SDF model, KPN (Kahn Process Network) model
- Platform-based approaches
Introduction

**Programming Platform**

Design of a Parallel Embedded System

- **Model-based Design**
- **Manual Design**

```
Model-based Design  →  Manual Design
↓                ↓
Software Platform
↓                ↓
Hardware Platform
```

- "Programming Platform"
  - Enhances retargetability
  - Provides a common layer for analysis and code synthesis
Programming Platform

Design Flow

Model-based Design
- UML
- KPN
- SDF

Manual Design
- Task
- Task
- Task

Common Intermediate Code (CIC)
(Architecture Independent Specification)

Architecture-aware Code Translation
(Exploits task-, temporal-, and data-parallelism)

Final Code optimized for Target-architecture
**CIC task model**

- Basically an actor-oriented model
  - Rich enough to accommodate various actor models
- Tasks: computation task, control task
  - Time driven or data driven
- Channels: FIFO, Array, or Buffer

**CIC Task Codes**

```c
TASK_INIT {
...
}
TASK_GO {
...
}
TASK_WRAPUP {
...
}
```

**Model Info.**
- Algorithm
- Architecture
- Mapping
- Control
- Profile

**Channel Types:**
- FIFO or Array
  - Modeling a shared memory (indexed slots)

**CIC Tasks**

- \( T_1 \)
- \( T_2 \)
- \( T_3 \)
- \( T_4 \)
Commonly used for
- Model-based Design
- Platform-based Design
  - "Programming Platform"

Advantages
- Potential parallelism is explicitly exposed.
  → Parallelization is nothing but mapping the actors to the processing elements.
- Various analysis methods and design tools have been developed.
  - Scheduling / mapping
  - Buffer optimization
  - Code synthesis
1. Single programming model fails to deal with various type of applications.
   - Pierre G. Paulin suggests the mixture of the three programming models:
     - Platform Programming
     - Symmetric multi-processing (SMP)
     - Stream programming model
     - Client-server programming model
   - Successfully applied to software design in STMicroelectronics

2. Layered Software
   - Layering software modules is a very popular concept.
   - However, it is not easy to express in previous actor-oriented models
Restrictions of Actor Models

Assumptions that causes the restrictions

- **Data Channel:** the only shareable object between actors
- **Actor:** self-contained
- **Algorithm Specification in Actor-oriented Model**
- **Flat Graph:** actors compose a system in a single flat layer
  - though hierarchy enables horizontal decomposition or mixture of multiple models

Library Support in an Actor-based Parallel Programming Platform
Resource sharing is desired

Actors are self-contained.

- May cause a redundant code copy or
- Code matching / implicit sharing
- Low productivity
  - High cost for maintenance
- Isolation disturbs sharing of functions.

```c
TASK_GO {
  ... r1 = dct(p); ...
}
dct(p) {
  ... ...
}
```

```c
TASK_GO {
  ... r2 = dct(p); ...
}
dct(p) {
  ... ...
}
```
Resource sharing is desired

Data channels are the only sharable object.

- We have many kinds of shared resources. (e.g. H/W IP's)
- We have to maintain sub-systems' internal states that are shared by many tasks.

• Representation of resource sharing is not intuitive in previous actor-oriented models.

• Task $T_s$ in the example should be modified when a new user task is added.
Vertically layered SW is needed

Actors compose a system in a single flat layer.

- It is not easy to specify multiple application/system layers only with the current actor-oriented models.

- Layering on multi-processors should be considered explicitly.

< Actor-oriented Specification >

< What we want >

Introduction

Library Support in an Actor-based Parallel Programming Platform

Seoul National University
Proposed Solution

New type of an actor: Library Task
- To represent resource sharing
- To specify vertically-layered software
- To model client-server applications

CIC Library Task
- A new type of an actor in the programming platform
- A sharable and mappable object that defines a set of function interfaces
- The concept may be easily applicable to the preexisting actor models.
  - Such as SDF or KPN

New types of ports
- Library master port: the caller-side port, which has a name
- Library slave port: the callee-side port
  - A library task has a single library slave port.
- N:1 master-slave connections are permitted.
An application example

A library task can be shared by multiple masters.

A master-slave connection explicitly reveals the relation between the caller and the callee.
## Library Support Specification

### Internal Specification

The diagram illustrates the internal specification of a library support in an actor-based parallel programming platform. Two tasks, $T_1$ and $T_2$, are involved, with interactions denoted by dashed lines.

- **Task $T_1$**
  - `void init(void) { .. }`
  - `void wrapup(void) { .. }
  - `int getValue(void) { .. }
  - `void setValue(int v) { .. }

- **Task $T_2$**

- **Library Definitions**
  - `extern LIBFUNC(void, init, void);`
  - `extern LIBFUNC(void, wrapup, void);`
  - `extern LIBFUNC(int, getValue, void);`
  - `extern LIBFUNC(void, setValue, int value);`

- **CIC Library Header File (.cicl.h)**
  - `CIC Library Header File (.cicl.h)`

- **CIC Library Source File (.cicl)**
  - `static int my_value;`
  - `LIBFUNC(void, init, void) { .. }`
  - `LIBFUNC(void, wrapup, void) { .. }`
  - `LIBFUNC(int, getValue, void) {
      return my_value;
    }
  - `LIBFUNC(void, setValue, int value) {
      my_value = value;
    }

- **Calling Code Example (From $T_1$)**
  ```
  TASK_GO {
    val = LIBCALL(MP1, getValue);
    LIBCALL(MP1, setValue, newVal);
  }
  ```

---

Library Support in an Actor-based Parallel Programming Platform

Seoul National University
## Directives and APIs for Library Task Specification

<table>
<thead>
<tr>
<th>Directive / API</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>LIBFUNC(return_type, function_name, type0 arg0, ...)</code></td>
<td>Define or declare a function that is exposed to the master task.</td>
</tr>
<tr>
<td><code>LIBCALL(lib_master_port, function_name, arg0, ...)</code></td>
<td>Call a function through the given library master port.</td>
</tr>
<tr>
<td><code>Lock()</code></td>
<td>Enter the critical section. It guarantees that only one critical section in a library can be executed at a time.</td>
</tr>
<tr>
<td><code>Unlock()</code></td>
<td>Exit the critical section.</td>
</tr>
</tbody>
</table>

To avoid data-race problem that can be caused by library task sharing
Previous task mapping algorithms

- There are numerous task mapping algorithms.
- None of them can be applied to a task graph with library tasks.
  : A library task has different characteristics from a normal task both in functional and timing behaviors.

Characteristics of Library Tasks

1) Containing multiple functions
   - A library task provides multiple functions.
   - Each function consumes different length of time.
     → The execution time of each function should be profiled or annotated for timing analysis.

2) Execution pattern
   - The execution pattern of a library task cannot be directly obtained from the task graph.
     → The execution rates of library tasks and the dependencies between master and slave tasks should be known.
Characteristics of Library Tasks (cont'd)

3) Discontinuous time-line of a task execution
   - A task or a library task can call a library function during its execution and wait for the result.
   - The time-line of the caller may be split into multiple slices.
     → The number of slices and their lengths should be identified.

4) Data-race problem caused by library task sharing
   - In case a library task has internal states and it is shared by multiple master tasks, the schedule should be carefully managed to avoid a data-race problem.
5) Library task replication
- To improve the performance, a library task that has no internal state, can be replicated on multiple processors.
Graph Transformation

- To a normal task graph
- The application behavior is obtained by profiling or annotation.

Transformation Result

< Application Graph >

< Application Behavior >

< Transformation Result >
**Additional Constraints**

- Slices of a single task should be mapped onto the same processor.

- Nodes associated with the same single library instance should be mapped onto the same processor.

- Library task replication
  - Up to whether there is any internal state or not
  → If not replicable, the nodes from a single library task should be mapped onto the same processor.
**Proposed Mapping Algorithm**

- Based on QEA (quantum-inspired evolutionary algorithm)

### Design of Q-stream

- '1' means the node is mapped onto the processor

\[ |\alpha|^2 + |\beta|^2 = 1 \]

- m quantum bits

\[ (\alpha, \beta) \]
Mapping Algorithm (cont'd): Our implementation

- Based on QEA (quantum-inspired evolutionary algorithm)

- The repair function alters illegal Q-streams into valid ones.
  - Also considers the additional constraints.

- Evaluation of a Q-stream
  - By list scheduling considering the communication overhead.

- Termination conditions
  - The number of generations reaches the given maximum number.
  - All individuals converge.
Automatic Code Generation Flow

- Architecture-neutral task codes are translated into target-specific codes.
- Scheduler code synthesis

Extensions for library task support

- Translation of the LIBCALL() / LIBFUNC() directives
- Code synthesis for remote function calls
Code Generation for Library Task Support

- A caller stub and a library wrapper task are created.

Library Support in an Actor-based Parallel Programming Platform

Seoul National University
Code Generation for Library Task Support (cont'd)

T1

T1_go {
  val = L_stub_getValue();
  L_stub_setValue(newVal);
}

L'

void L_stub_init(void) {}
void L_stub_wrapup(void) {}
int L_stub_getValue(void) {
  send_request(chid_TW_a,
              callerId_T1,
              fid_getValue, args);
  result = recv_result(chid_TW_r);
  return result.v;
} void L_stub_setValue(int v) {
  send_request(chid_TW_a,
              callerId_T1,
              fid_setValue, args);
  recv_ack(chid_TW_r);
}

Library Wrapper Task

void L_init(void) { .. }
void L_wrapup(void) { .. }
int L_getValue(void) { .. }
void L_setValue(int v) { .. }
Math Library Example

- For a given function module, there may be many implementations.
  - Different cost-performance trade-off
- Math library in this example: Running as a calculation server
  - Standard library-based or table-based
MPEG4 SP Decoder Example

- Represent a shared resource or a device driver of a hardware-IP

Experiments

A shared IDCT module that is accelerated by a hardware-IP

Library Support in an Actor-based Parallel Programming Platform

Seoul National University
Motor Control Example

- Simple test example for vertically-layered software
- The motors' status are shared by the controller and the monitor.
### Effect of Replication of Library Tasks

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Math</td>
<td>5+1</td>
<td>Math1_L</td>
<td>1</td>
<td>51611</td>
<td>34580</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>228478</td>
<td>136417</td>
<td></td>
</tr>
<tr>
<td>Mpeg4 SP Dec.</td>
<td>29+1</td>
<td>IDCT4_L</td>
<td>1</td>
<td>206211</td>
<td>206211</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>957811</td>
<td>957811</td>
<td></td>
</tr>
</tbody>
</table>

Library replication is effective.

Library replication is not effective.

< Arch. 1 >

< Arch. 2 >
**Automatic Mapping**

**Design Space Exploration**
- Finding pareto-optimal points in architectural configurations
- Latency vs. Number of Processors (with arch. 2)

*Experiments*

<table>
<thead>
<tr>
<th>Ex.</th>
<th>Library Task</th>
<th># Proc.</th>
<th># Replications</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math</td>
<td>Math1_L</td>
<td>1</td>
<td>1</td>
<td>287236</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>2</td>
<td>150818</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>2</td>
<td>136417</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>2</td>
<td>136417</td>
</tr>
<tr>
<td>Mpeg4 SP Dec.</td>
<td>IDCT4_L</td>
<td>1</td>
<td>1</td>
<td>1643170</td>
</tr>
<tr>
<td>(b)</td>
<td></td>
<td>2</td>
<td>2</td>
<td>957818</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>2</td>
<td>957811</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>2</td>
<td>957811</td>
</tr>
</tbody>
</table>

- The time for obtaining the result is *below ten seconds* for each case.
Automatic Code Synthesis

Screen-shot of our design environment (MPEG4 SP Decoder example)

Foreman (QCIF)
The following codes are automatically generated

- Scheduler and communication libraries
- Task wrappers and etc. (including build-related codes)

Pros

- Enhancing development productivity
- Accelerating design space exploration

<table>
<thead>
<tr>
<th>Ex.</th>
<th>Arch.</th>
<th>Scheduler/Comm.</th>
<th>Task Wrappers</th>
<th>Etc</th>
<th>Total</th>
<th>Ratio (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math</td>
<td>1</td>
<td>2167</td>
<td>1049</td>
<td>55</td>
<td>3271 (/3641)</td>
<td>89.8</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5775</td>
<td>207</td>
<td>210</td>
<td>6192 (/6553)</td>
<td>94.5</td>
</tr>
<tr>
<td>Mpeg4 SP Dec. (b)</td>
<td>1</td>
<td>3850</td>
<td>1374</td>
<td>107</td>
<td>5331 (/11414)</td>
<td>46.7</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>6515</td>
<td>280</td>
<td>362</td>
<td>7157 (/13286)</td>
<td>53.9</td>
</tr>
<tr>
<td>Motor</td>
<td>1</td>
<td>2065</td>
<td>1511</td>
<td>55</td>
<td>3631 (/3841)</td>
<td>94.5</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4707</td>
<td>285</td>
<td>191</td>
<td>5183 (/5523)</td>
<td>93.8</td>
</tr>
</tbody>
</table>
Conclusion

We propose a new type of an actor – a library task

- A sharable and mappable object
- Enhancing the modularity of the system
- Enabling the specification of
  - vertically-layered software
  - client-server type of applications
- Preserving the benefits of actor-oriented modeling

Supporting the library tasks in our design framework

- Automatic mapping algorithm
- Automatic code synthesis
Thank you!