Programming-Platform Based Design of MPSoC: HOPES Approach

June 29, 2009

Soonhoi Ha
Seoul National University
Contents

1. Introduction
2. Proposed Design Flow
3. Key Techniques
4. Preliminary Experiments
5. Discussion and Conclusion
Parallel Embedded SW Design Challenge

Parallel programming for non-trivial heterogeneous systems with diverse design constraints (time, power, cost, and so on)

Parallel programming: current practice

• Manual multi-tasking/multi-threading programming
• General purpose parallel programming models
  • MPI: message passing model
  • OpenMP: shared address space model
• Target-specific programming models
  • CUDA for Nvidia GPGPU
Problem Statement

Target: MPSoC with high degree of parallelism
- Scalability
- Heterogeneous processors with diverse communication architecture
- Power/resource-constrained system

Problem: parallel programming for MPSoC
- Parallelism extraction (multiple use case, multi-tasking apps.)
  - Functional parallelism, data-parallelism, temporal-parallelism
- Partitioning and mapping
- Parallel code generation
- Performance estimation and verification
- Design space exploration

Need of sound (scalable and robust) methodology
Basic Idea

Applications (Manual design)

Software Platform
Hardware Platform

Model-based design

Programming platform (CIC)
Software Platform
Hardware Platform

(Manual design)
HOPES Design Flow

- Dataflow Model
- UML
- KPN

Automatic Code Generation

Manual Coding

Common Intermediate Code

- Task Codes (Algorithm)
- XML File (Architecture)

Task Mapping

Performance Lib./ Constraints

CIC Translation

Target-Executable C Code

Virtual Prototyping System

HOPES project, SNU
**CIC (Common Intermediate Code)**

- **Separation of target-independent task codes (C-code) and target-dependent architecture information (xml-file)**

<table>
<thead>
<tr>
<th>CIC (tasks)</th>
<th>CIC (architecture)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interfaces:</td>
<td>Hardware information:</td>
</tr>
<tr>
<td>* _init()</td>
<td>Processor information</td>
</tr>
<tr>
<td>* _go()</td>
<td>H/W accelerator info.</td>
</tr>
<tr>
<td>* _wrapup()</td>
<td>OS, Memory maps</td>
</tr>
<tr>
<td>Depends on:</td>
<td>Constraints:</td>
</tr>
<tr>
<td>Generic APIs</td>
<td>Timing requirement</td>
</tr>
<tr>
<td>(including comm. APIs)</td>
<td>Memory constraint</td>
</tr>
<tr>
<td>Additional information:</td>
<td>Power consumption</td>
</tr>
<tr>
<td>OpenMP pragmas</td>
<td>Structure:</td>
</tr>
<tr>
<td>Hardware pragmas</td>
<td>Task structures</td>
</tr>
<tr>
<td></td>
<td>Channel &amp; Task mapping</td>
</tr>
</tbody>
</table>

* CIC (CIC) and CIC (architecture) are separated with target-independent task codes (C-code) and target-dependent architecture information (xml-file).

* Interfaces include *(init, go, wrapup)*.

* Depends on Generic APIs (including communication APIs).

* Additional information includes OpenMP pragmas and Hardware pragmas.

* Hardware information includes Processor information, H/W accelerator info, OS, Memory maps.

* Constraints include Timing requirement, Memory constraint, Power consumption.

* Structure includes Task structures, Channel & Task mapping.
CIC Tasks

Express the "potential functional parallelism" of the application

- The designer defines the task granularity
- CIC tasks are concurrent tasks: default channel is FIFO channel
  - Data-driven task: # of required samples, relative execution rate

Dataflow model

CIC task model

Clustering
CIC code generation
A CIC task is defined by three methods

- \_init(): before main loop
- \_go(): in the main loop
- \_wrapup(): after main loop

Use generic APIs for target independence

A CIC task may be a data-parallel task

- Different specification of data parallelism from functional parallelism

```c
void taskC_init () { /* task initialization code */ };  
void taskC_go (void) {
  l = MQ_RECEIVE("mq0", (char *)(ld_106->rdbfr), 2048);
  ...
  //task_body()
  MQ_SEND("output", (char *)(st_107->buf), 2048);
}
void taskC_wrapup () { /* task wrapup code */ };  
```
Data Parallelism in CIC

Data parallelism is specified by a data-parallel CIC task

Two types of data parallelism

- **openMP-style parallelism**: invocations are independent
  - (example) Macroblock decoding of H263 decoder
  - Specification method: openMP pragma

- **Wavefront parallelism**: invocations are dependent
  - (example) Macroblock analysis of H264 encoder
  - Specification method: CIC definition of wavefront vector

```c
#pragma omp parallel for
for(i=0; i<99; i++) {
    //T2_main()
    ....
}
```
CIC Architecture Information

Hardware information
- Processor kinds
- Address space: local and shared memory
- OS information

Design constraints
- Timing constraints per task: Period and deadline
- Power and memory constraints

Task structure
- Mode information: support different use cases
- CIC hierarchy information

```xml
<?xml version="1.0"?>
<CIC_XML>
  <hardware>
    <processor name="arm926ej-s0">
      <index>0</index>
      <localMem name="lmap0">
        <addr>0x0</addr> <size>0x10000</size>  // 64KB
      </localMem>
      <sharedMem name="shmap0">
        <addr>0x10000</addr> <size>0x40000</size>  // 256KB
        <sharedWith>1</sharedWith>
      </sharedMem>
    </processor>
    <OS>
      <support>TRUE</support>
    </OS>
  </hardware>
</CIC_XML>
```
Contents

1. Introduction

2. Proposed Design Flow

3. Key Techniques
   - Partition and Mapping
   - CIC translator

4. Preliminary Experiments

5. Discussion and Conclusion
Functional Specification:
(multi-rate task graph)

Target Architecture:
Set of Processor Pools
* Processor Pool: Set of Homogeneous PEs

Partitioning and Mapping

Constraint

Task Parallelism

Parallelism

+ Data Parallelism

+ Temporal Parallelism

HOPES project, SNU
Proposed Solution - QEA

Quantum-inspired Evolutionary Algorithm

- Adjust Q-bits according to the current best solution
- Evaluate them generated solutions & pick up the best solution in each group and the global best
- Generate solutions according to the probabilistic condition of quantum bits & repair them
- Probability that the according bit goes to '0'
- Probability that the according bit goes to '1'

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

\((\alpha, \beta)\)
### Mapping Information on each subtask

- **Length** of the string: number of processors in the target architecture
- Initially generated according to Q-stream probabilistic status

### Repairing for valid mapping

1. Choose a processor pool among chosen – invalidate all mappings on other pools
2. Number of chosen processors in the pool should be less than or equal to the maximum data-parallelism degree

### Task and Data parallelisms are considered here.

#### Q-string structure

<table>
<thead>
<tr>
<th>subtask_1</th>
<th>subtask_2</th>
<th>...</th>
<th>subtask_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>pool_1</td>
<td>pool_2</td>
<td>...</td>
<td>pool_m</td>
</tr>
</tbody>
</table>

```
011110 001010000 00000000
010110 00000000
```

* Maximum data-parallelism degree of subtask_2 is 3
**Valid pipeline set**

- Should **NOT** have non-delayed cycle between consecutive pipeline stages
  - At most \((n-1)\) pipelines can exist on \(n\) delayed feedback edge
  - Do not cross each other

**Enumerating all possible pipelines**

- Pipeline Ordering Graph (POG)
  - Each node of the POG denotes a valid pipeline cut
  - A path from the source to the destination in the POG guarantees the valid pipeline set
Proposed Solution - Pipeline

Pipeline information
- **Length** of the string: # of POG nodes
- Initially generated according to Q-stream probabilistic status

Repairing for valid pipelining
1. Enforce at most (n-1) pipelines on n-delayed edge
2. Find the ‘most chosen path’ in POG

Temporal Parallelism considered here

---

Q-string structure

<table>
<thead>
<tr>
<th>Mapping</th>
<th>Pipelining</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>1</td>
</tr>
<tr>
<td>CE</td>
<td>1</td>
</tr>
<tr>
<td>DE</td>
<td>1</td>
</tr>
<tr>
<td>BCE</td>
<td>1</td>
</tr>
<tr>
<td>CDE</td>
<td>0</td>
</tr>
<tr>
<td>BCDE</td>
<td>1</td>
</tr>
</tbody>
</table>

---

HOPES project, SNU
CIC Translation

**CIC to Multi-thread codes for functional simulation**
- Generated codes are run on a host machine

**CIC to target C codes**
- Target specific code generation
  - For virtual prototyping
  - For MPCore
  - For Cell processor [planned]
  - DSP array
  - Reconf. Hardware
- Per-processor code generation based on mapping information
  - Multi-threaded task codes
  - Interface code generation
  - Scheduler code generation
CIC Translator Procedure

1. Task Codes (Algorithm)
2. XML File (Architecture Info.)
3. Generic API Translation
4. Is OpenMP compiler available?
   - Yes
     - OpenMP Translation
   - No
     - Target dependent translation
5. Task Scheduling Code Generation
6. Target Dependant Parallel Code
Generic API

- OS-independent API
- Abstraction from IEEE POSIX 1003.1-2004
- Selected OS services + C library functions

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

file = open("input.dat", O_RDONLY);
...
read(file, data, 100);
...
close(file);
```

```c
#include <stdio.h>
...
file = fopen("input.dat", "r");
...
read(file, data, 100);
...
close(file);
```
typedef struct {
  void (*init)();
  int (*go)();
  void (*wrapup)();
  int period;
  int time_count; /* this variable indicates the next invocation time of a task */
} task;

int taskNum = 2;
task taskInfo[] = {{task1_init, task1_go, task1_wrapup, 100, 0}, {task2_init, task2_go, task2_wrapup, 200, 0}};
...

int main() {
  init();    /* {task_name}_init() functions of all tasks are called */
  scheduler(); /* scheduler code */
  wrapup();  /* {task_name}_wrapup() functions of all tasks are called */
  return 0;
}
### Scheduling Code (2): w/o OS

```c
void scheduler() {
    while(all_task_done()==FALSE) {
        int taskId = get_next_task();
        if (taskInfo[taskId]->go()==0) {
            taskInfo[taskId]->timeCount += taskInfo[taskId]->period;
        }
    }

    /* return the number of task id that should be executed */
    int get_next_task() {
        for all task in this processor
            find the tasks that has the smallest value of time_count variable
        if (the number of found tasks > 1) {
            for found tasks that has the smallest value of time_count variable
                select the task that is not executed for the longest time in found tasks
        }
        return task_id;
    }
}
```
void *thread_task_0_func (void *argv) {
    ...
    task_0_go();
    get_time(&time);
    sleep(task_0->next_period – time); //sleep during remained time
    ...
}

int main() {
    ...
    pthread_t thread_task_0;
    sched_param thread_task_0_param;
    ...
    thread_task_0_param.sched_priority = 0;
    pthread_attr_setschedparam (…, &thread_task_0_param);
    ...
    task_init(); /* In this function, task_name_init of each is called */
    pthread_create (&thread_task_0, &thread_task_0_attr, thread_task_0_func, NULL);
    ...
    task_wrapup(); /* In this function, task_name_wrapup of each task is called */
}
Contents

1. Introduction

2. Proposed Design Flow

3. Key Techniques

4. Preliminary Experiments

5. Discussion and Conclusion
(1) Model-based Design: H.263 CODEC

H263 Encoder/decoder in extended SDF (FRDF) specification
Clustering SDF nodes into CIC tasks to keep functional parallelism as much as possible. (Note data parallel execution of a task in H.263 decoder)
Generate CIC files from the clustered SDF graph automatically.
CIC Mapping (Manual Mapping)

Mapping CIC tasks into the target processing elements
CIC Translation

Target C codes

H.263 encoder

H.263 decoder (parallelized code)
Prototyping
(2) Manual Design: X264 Encoder on CELL

Objective
- Shows that CIC programming is a feasible solution for non-trivial parallel programming

Challenges
- Heterogeneous MPSoC
  - 1 PPE and 6 SPEs on Cell
- Complex communication architecture
- Non-trivial wave-front parallelism

Solution
- Code generation for both PPE & SPEs
- x2.0 speed-up
Manual Design: X264 Encoder on CELL

- **Init / File read**
- **ME**
- **Encoder**
- **Deblocking Filter**

Variable Length Coding / File write

Feedback port with index

Array channel

(QVGA)

MB Encode 84%

MB Analysis 84%

Write File 4%

VLC 3%

Read File 1%

Frame Init 2%

(time)

(ratio of parallelization)

HOPES project, SNU
X264 Encoder on CELL(2)

CIC translator synthesizes wrapper automatically

- Init / File read
- Wrapper
- Encoder
- Decoder
- Deblocking Filter
- Variable Length Coding / File write

feedback port with index
Objective
- Shows that CIC programming platform is able to specify and implement control-oriented applications as well

Challenges
- Control-oriented target: NXT Robot
  - 4 sensors, 3 servo motors
  - Bluetooth communication with PC
- CIC translation with dynamic task scheduling

Solution
- CIC control task with control APIs
  - Set task parameters
  - Change status with timing condition
- Code generation for both PC & Robot
CIC Task Structure

- ControlPC
- KeyDetect
- ControlNXT
- SensorDetect
- Move
- Grab
- LCD
Control NXT robot by both a PC and the robot itself.

1. ControlPC task is a PC-side NXT-controller.
2. ControlNXT task is an NXT-side NXT-controller.
3. SensorDetect task reads sensor values and sends them to two control tasks: ControlPC and ControlNXT.
4. KeyDetect task reads key input value and sends it to ControlPC task
5. Controlled by ControlPC task and ControlNXT task, Move task and Grab task run motors.
6. LCD task displays the current status of NXT
(4) Design Productivity Test: H.263 Decoder

Task graph and partitioning

0. Variable Length Decoding
1. Macroblock Decoding Y
2. Macroblock Decoding U
3. Macroblock Decoding V
4. Motion Compensation
5. Display Frame

Dequantize → Inverse Zigzag → IDCT
# Configurations and Execution Cycles

## Task mapping

<table>
<thead>
<tr>
<th>Processor id</th>
<th>Configuration (task mapping)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Task 0, 1, 2, 3, 4, 5</td>
</tr>
<tr>
<td>1</td>
<td>Task 0, 2, 3, 4, 5</td>
</tr>
<tr>
<td>2</td>
<td>Task 0, 3, 4, 5</td>
</tr>
</tbody>
</table>

## Execution Cycles

<table>
<thead>
<tr>
<th># processors for data parallelism</th>
<th>Configuration (task mapping)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No OpenMP</td>
<td>158,099,172</td>
</tr>
<tr>
<td>2</td>
<td>167,119,458</td>
</tr>
<tr>
<td>4</td>
<td>168,640,527</td>
</tr>
<tr>
<td>2</td>
<td>152,753,214</td>
</tr>
<tr>
<td>4</td>
<td>154,159,995</td>
</tr>
<tr>
<td>3</td>
<td>146,464,503</td>
</tr>
<tr>
<td>2</td>
<td>153,127,710</td>
</tr>
<tr>
<td>4</td>
<td>155,415,942</td>
</tr>
</tbody>
</table>
Comparison with Manual Coding

Number of code lines for functional parallelism

<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Communication</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code for processor 1</td>
<td>2393</td>
<td>226</td>
<td>2619</td>
</tr>
<tr>
<td>Code for processor 1</td>
<td>400</td>
<td>226</td>
<td>626</td>
</tr>
<tr>
<td>Original code</td>
<td></td>
<td></td>
<td>2507</td>
</tr>
</tbody>
</table>

Number of code lines for data parallelism

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>Master</th>
<th>Slave</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of code line</td>
<td>56</td>
<td>97</td>
<td>249</td>
</tr>
<tr>
<td>Time to write</td>
<td></td>
<td></td>
<td>12 man hours</td>
</tr>
</tbody>
</table>

Performance comparison with manual coding

- Overhead: 10.2 %
1. Introduction
2. Proposed Design Flow
3. Key Techniques
4. Preliminary Experiments
5. Discussion and Conclusion
Currently two ways of CIC generation are supported

- PeaCE models to CIC: FSM model to CIC is under development
- Manual specification of CIC tasks
- (Simulink Model to CIC, UML to CIC might be interesting)

CIC model refinement

- X264 specification uncovers need of extensions.
- Control behavior specification

Retargetable and scalable virtual prototyping

- Debugging and monitoring features will be added
- Fast simulator based on virtual synchronization technique is under development
HOPES (http://peace.snu.ac.kr/hopes) is a newly launched project to make an embedded software development environment for MPSoCs.

- Support of diverse models
- Target independent environment + target specific libraries

Programming platform idea seems working!

- CIC is in-between specification and implementation

New challenges: CIC translator

- New types of compiler considering all design constraints and target peculiarities