Designer-Controlled Re-Coding for Parallel and Flexible MPSoC Specification

Rainer Dömer and Pramod Chandraiah
Center for Embedded Computer Systems
University of California, Irvine

Multi-Processor System-on-Chip Design

- MPSoCs are promising, but challenging
  - Heterogeneous by nature
  - Strict requirements
    - Low power
    - High performance
    - Low cost
- System level design
  - Specification to implementation
    - Reference code (sequential, flat)
    - System model (parallel, flexible)
    - MPSoC platform (optimized implementation)
- Our research
  - How to get from flat and sequential C code to a flexible and parallel system model?
Outline

- Introduction and Motivation
- Problem Definition
- Re-Coding Transformations
  - Code and Data Partitioning
  - Creating Explicit Communication
- Interactive Re-Coding
  - Designer-controlled Source Re-coder
- Experiments and Results
- Summary and Conclusions

Introduction and Motivation

- System Level Design
  - Increase productivity by using higher abstraction levels
- System design flow
  - Top-down
  - Refinement-based
  - Starts from Specification Model
    - Using System Level Design Languages (SystemC, SpecC)
  - Successive automatic refinement steps result in end implementation
Introduction and Motivation

- Need quality input model
  - Estimation, exploration, refinement, synthesis
- However, actual design starts with reference model (flat C code)
  - Gap to create Specification Model
- Existing design flows do not automatically create this model [Gerstlauer et.al.’01, Groetker et.al.’02]
- Manual re-coding needed
  - Case study: MP3 decoder
  - >90% of design time needed to create Specification Model

Problem Definition

- How do we go from flat C code to a flexible specification model?
- Re-coding
  - Introduce concurrency
  - Create structural hierarchy
  - Expose communication
  - Make the code compliant to the design tools
  - ...
- Our approach
  - Designer-controlled Re-coder
    - Interactive source code transformations
Overcoming the Specification Gap

- Re-Coding Transformations

---

Overcoming the Specification Gap

- Re-Coding Transformations
  - Code and data partitioning [DAC’07]
  - Creating explicit communication [ASPDAC’07]
  - ...

---
Code and Data Partitioning

- Today's parallelizing compilers (e.g., Intel) are ineffective for MPSoC platforms
  - Are fully automatic
  - Ineffective for heterogeneous architectures
  - Ineffective on embedded source codes
  - Cannot effectively utilize application-specific knowledge
- Our Approach
  - Designer-controlled programming
  - Designer-in-the-loop
    - Designer utilizes his application knowledge
    - Designer makes decisions
  - Programming by automatic transformations
    - Tools automatically transform the source code

Desirable model features
- Enable parallel execution
- Allow mapping to different PEs

Recoding tasks
- Partition code
- Partition data
- Synchronize dependents

Designer-controlled transformations
1. Loop splitting
2. Cumulative Access Type analysis
3. Partitioning of vector dependents
4. Synchronizing dependent variables
1. Loop Splitting

- **Code Partitioning**
  - Designer selects loops to be split

- **Re-coder creates multiple incarnations of the loop**
  - Adjusted loop parameters

- **If loop parameters cannot be determined statically**
  - Designer specifies loop parameters
  - E.g. *while* loops

2. Cumulative Access Type Analysis

- **Identify dependents between loop partitions**

- **Accumulated accesses to variable in the loop**
  - Eg: Cumulative access of *x* is *Write-Read*

- **Cumulative Access Types (CATs)**
  - *Read* (*c, d*)
  - *Write* (*b*)
  - *Write-Read* (*x*)
  - *Read-Write* (*a*)

- **R, W, WR don’t need synchronization**
- **RW types need to be synchronized**
- **What about composite structures?**
  - Vector Data Partitioning (next)
3. Vector Data Partitioning

- Breakdown vectors to loop partitions
- Vector variables can be partitioned
  - Array access analysis needed
  - In general, not analyzable
    - E.g. A[B[I]]
  - However, affine references are analyzable
    - E.g. i, 2i, 2i+1
- Analysis of affine references of the form \( mx+b \)
  - Check if two affine references \( m_1 x+b_1 \) and \( m_2 x+b_2 \)
    point to same location

\[
\begin{align*}
    & m_1 x+b_1 \neq m_2 (S+k\Delta x)+b_2 \\
    \forall k: & 1 \leq k \leq ((E-S+1)/\Delta x) - 1 \quad \forall x: S \leq x \leq E \\
    & (m_1 x-m_2 S+b_1-b_2)/(m_2 \Delta x) \neq k \\
    \forall k: & 1 \leq k \leq ((E-S+1)/\Delta x) - 1 \quad \forall x: S \leq x \leq E
\end{align*}
\]

(c) R. Doemer, P. Chandraiah

---

3. Vector Data Partitioning

- Example analysis result
  - Vector \( b \) can be partitioned
    - Create new set of smaller vectors

(c) R. Doemer, P. Chandraiah
3. Vector Data Partitioning

- Example analysis result
  - Vector \( b \) can be partitioned
    - Create new set of smaller vectors
    - Adjust index expressions
  - Vector \( a \) cannot be partitioned
    - Same elements are accessed across partitions

for \((i=0; i<4; i++)\)
- \( x = i + i; \)
- \( a[i] += c; \)
- \( a[2i] = c + d; \)
- \( b1[i] = c^*d - x; \)

for \((i=4; i<8; i++)\)
- \( x = i + i; \)
- \( a[i] += c; \)
- \( a[2i] = c + d; \)
- \( b2[i-4] = c^*d - x; \)

for \((i=8; i<12; i++)\)
- \( x = i + i; \)
- \( a[i] += c; \)
- \( a[2i] = c + d; \)
- \( b3[i-8] = c^*d - x; \)

for \((i=12; i<16; i++)\)
- \( x = i + i; \)
- \( a[i] += c; \)
- \( a[2i] = c + d; \)
- \( b4[i-12] = c^*d - x; \)

4. Synchronizing Dependents

- Variables with CAT RW
  - Need synchronization
- Insert abstract channels
  - \( \text{send()} \)
  - \( \text{recv()} \)
Creating Explicit Communication

• Why explicitly expose communication?
  • Quality of Communication Exploration
    – Number of explorations
    – Extent of automation
    – Time
  • Shared-Memory Model
    – Global variables limit the number of possible automatic explorations
  • Explicit Communication Model
    – Enables automatic exploration of more design alternatives

Creating Explicit Communication

• Expose Communication
  – Transformations:
    1. Localize global variables
    2. Expose communication
    3. Synchronize accesses
  – Interactive Re-coding approach
    • Designer iteratively invokes transformations
    • No manual coding
Explicit Communication: 1. Localize

- Localize global variables to partitions
  - To enable multiple explorations
- Procedure
  - Find the global variable
  - Determine the functions and behaviors accessing it
  - If only one behavior is accessing it, migrate the variable into this behavior

Explicit Communication: 2. Expose

- Localize global variables to common parent and provide explicit access
  - Simplifies subsequent analysis of models
- Procedure
  - Find the global variable
  - Determine the functions and behaviors accessing it
  - If multiple behaviors are accessing it, find the lowest common parent
  - Migrate the variable to the parent
  - Provide access to the variable by recursively inserting ports in behaviors
Explicit Communication: 3. Synchronize

- Use message passing channels instead of variables
  - Defines synchronization scheme
  - Guides exploration tools

- Procedure
  - Create a typed synchronization channel
  - Replace the ports corresponding to the original variable with the channel interface type
  - Modify each access to the variable to call the appropriate interface function of the channel
    - read() / receive()
    - write() / send()

Explicit Communication: Example Code

- Transformations require significant recoding!

(a) Model-1: Original Model
(b) Model-2: After Localization
(c) Model-3: Exposed connectivity
(d) Model-4: Synchronized Model

Paderborn, Germany, 08/03/2007
Interactive Re-Coding Environment

- Why interactive?
  - Re-coding is not a monolithic transformation
    - Multiple iterative transformations
  - Designer-control necessary
to create a properly structured model
    - Structure of the model is determined by
      - Type of code transformations
      - Order in which the transformations are applied
  - Re-coding involves multiple hard problems
    - Handling pointers
    - Exposing concurrency
    - …

Existing Interactive Environments

- Notepad
- Vim/Emacs
- Visual Studio C++ [7]
- Eclipse JDT [6]
- D Editor [5]
- SUFI Explorer [3]
- Parascope [4]
- Our Re-coder

<table>
<thead>
<tr>
<th>Text</th>
<th>Syntax</th>
<th>Semantics</th>
<th>Analysis</th>
<th>Program Transformations</th>
</tr>
</thead>
</table>

(c) R. Doemer, P. Chandraiah
Source Re-coder

- Controlled interactive approach
- Automatic programming
- It’s a union of
  - Editor
  - AST
  - Parser
  - Transformations
  - Code generator

Source Re-coder

- Text editor
  - Interface to the designer
  - Basic and advanced source-code editing
    - C/C++/SpecC
  - Document object
    - Based on Andrew text editor [8]
    - Basically text organized for faster access
Source Re-coder

- Editor
- Abstract Syntax Tree
  - Captures the structure of the program
  - For transformation tools
  - Completeness
    - C and SLDLs
    - Correspondence with document object
  - Needed to re-generate code in its original form

Design
- Behaviors
- Symbol Table
- Type Table
- Interfaces
- Variables
- Functions

Document Object
- Preproc
- Parser
- Code Generator
- AST
- Transformation Tools

GUI

(c) R. Doemer, P. Chandraiah

Paderborn, Germany, 08/03/2007

Source Re-coder

- Editor
- AST
- Preprocessor and Parser
  - Build AST
  - Keep AST in synch
  - Complement the editor
    - Color coding
    - Syntax high-lighting

Document Object
- Preproc
- Parser
- Code Generator
- AST
- Transformation Tools

GUI

(c) R. Doemer, P. Chandraiah

Paderborn, Germany, 08/03/2007
Source Re-coder

- Editor
- AST
- Parser
- Code Generator
  - Keeps text in sync
  - Generates the SLDL source-code

Source Re-coder

- Editor
- AST
- Parser
- Code Generator
- Transformation tools
  - Code transformations
    - Code partitioning
    - Introducing behaviors, functions
  - Data transformations
    - Variable re-scoping
    - Data structure partitioning
  - Analysis
    - Dependency analysis
    - Pointer analysis
Source Re-coder Implementation

- Interactive environment
  - Scintilla + QT + AST + Transformations
- Basic editing
  - Syntax highlighting
  - Auto-completion
  - ...
- Transformations
  - Splitting code & composite data structures
  - Variable localizing
  - Variable re-scoping
  - Inserting ports
  - Dependency analysis
  - ...

Experiments and Results: Responsiveness

- Why measure Responsiveness?
  - To check feasibility
- Responsiveness
  - Response to designer actions
  - Time to synch AST
    - On editing
  - Time to synch Editor
    - On transformation
  - Depends on the size of the AST
- Design examples
  - JPEG, MP3, GSM
  - ≪ 1 sec (on a 3 GHz Linux PC)
  - File I/O overhead (20%)
Experiments and Results: Partitioning

- MP3 Audio decoder
  - Application-specific Knowledge
    - Left and Right audio channels are independent

Experiments and Results: Partitioning

- C source of MP3
  - Concurrency was hidden
    - Multiple function layers
  - Automatic tools fail
    - Intel C compiler
  - Source re-coder
    - Loops were split
    - Dependents were identified
    - Vectors partitioned
  - Sequential code into parallel spec.
Experiments and Results: Partitioning

- Productivity Gains
  - Manual re-coding takes hours
  - Source re-coder takes minutes
  - Productivity: 100x

<table>
<thead>
<tr>
<th>Design Time</th>
<th>JPEG</th>
<th>Fix-Point MP3</th>
<th>Floating-Point MP3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loops split</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Vectors split</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Other operations</td>
<td>2</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>Estimated Manual time</td>
<td>85 mins</td>
<td>360 mins</td>
<td>340 mins</td>
</tr>
<tr>
<td>Re-coding time</td>
<td>≈2 mins</td>
<td>≈ 4 mins</td>
<td>≈ 3 mins</td>
</tr>
<tr>
<td>Productivity factor</td>
<td>42</td>
<td>90</td>
<td>113</td>
</tr>
</tbody>
</table>

Experiments and Results: Communication

- Design examples
  - JPEG, MP3, GSM
- Created specification models
  - Manual
  - Source Re-Coder
- Estimated manual time
  - Time to implement the transformations manually for 10 variables, extrapolated for all the variables
  - In the order of minutes
- Re-coding time
  - Time to implement the transformations using Source Re-coder
  - In the order of seconds
- Productivity gain
  - 100x
Summary and Conclusion

- MPSoc specification
  - Good models are necessary for effective exploration and synthesis
  - C reference models act as starting point
- Gap between reference and specification models
  - 90% of the overall design time (MP3 example)
  - Modeling and coding need automation
  - Complete automation is difficult (today’s compilers are ineffective)
- Interactive re-coding using Source Re-Coder
  - Designer-in-the-loop
  - Programming is no longer tedious typing
  - Programming becomes guided automatic re-coding
- Large productivity gains
  - 100x over manual re-coding
- Future work
  - Research and develop more transformations
  - Couple re-coder with system profiling and estimation tools