Computer-Aided Recoding for Multi-Core Systems

Rainer Dömer
doemer@uci.edu

With contributions by P. Chandraiah

Center for Embedded Computer Systems
University of California, Irvine

UCIrvine
University of California, Irvine
Outline

• Embedded System Design
• Computer-Aided Recoding
• Recoding Transformations
  – Creating structural hierarchy
  – Exposing potential parallelism
  – Creating explicit communication
• Interactive Source Recoder
• Experiments and Results
• Conclusions
Embedded System Design

• How can we overcome the productivity gap?

International Technology Roadmap for Semiconductors (ITRS) 2004: *higher-level abstraction and specification* is the first promising solution

• System Level Design
  – Unified HW and SW design
  – Higher level of abstraction
    • Fewer, more complex components
    • Maintain system overview
      – Without overwhelming details
  – System Level Design Languages
    • SpecC [Gajski et. al, 2000]
    • SystemC [Groetker et. al, 2002]

Embedded System Design

• System Level Modeling
  – Abstract description of a complete system
  – Hardware + Software

• Key Concepts in System Modeling
  – Explicit Structure
    • Block diagram structure
    • Connectivity through ports
  – Explicit Hierarchy
    • System composed of components
  – Explicit Concurrency
    • Potential for parallel execution
    • Potential for pipelined execution
  – Explicit Communication and Computation
    • Channels and Interfaces
    • Behaviors / Modules
Computer-Aided Recoding

- Embedded System Design Flow
  - Input: System model
  - Output: MPSoC platform

- Actual Starting Point
  - C reference code
    - Flat, unstructured, sequential
    - Insufficient for system exploration

- Need: System Model
  - System-Level Description Language (SLDL)
  - Well-structured
    - Explicit computation, explicit communication
    - Potential parallelism explicitly exposed
  - Analyzable, synthesizable, verifiable

- Research: Automatic Re-Coding
  - How to get from flat and sequential C code to a flexible and parallel system model?
Motivation

• **Extend of Automation**
  - Refinement-based design flow
  - **Automatic**
    • Specification model down to implementation
    • Example: SCE (mostly automatic)
    • MP3 decoder: less than 1 week
  - **Manual**
    • C reference code to SpecC specification model
    • Source code transformations
    • MP3 decoder: 12-14 weeks!

• **Automation Gap**
  - 90% of overall design time is spent on re-coding!

• **Proposal: Automatic Recoding**
Computer-Aided Recoding

• Complete Automation is Infeasible!
  – Today’s parallelizing compilers are largely ineffective
    • Heterogeneous architectures
    • Complexity of embedded applications
    • Hard problems (eliminating pointers, exposing parallelism, etc.)
  – Modeling requires understanding of the application
  – Recoding is not a monolithic transformation
    • Multiple transformations in application-specific order

Interactive Approach
  – “Designer-in-the-loop”
  – Designer can utilize application knowledge

• Designer-controlled Transformations
  – Designer makes decisions
  – Tool automatically transforms the source code
Overcoming the Specification Gap

- Recoding Transformations
  - Creating structural hierarchy [ASPDAC’08]
  - Code and data partitioning [DAC’07]
  - Creating explicit communication [ASPDAC’07]
  - Recode pointers [ISSS/CODES’07]
Creating Structural Hierarchy

- **Recoding**
  - Convert functional hierarchy into structural hierarchy
  - Step-wise model transformation
  - Hierarchical encapsulation
    - Utilize given function call tree
    - Convert each function into a behavior
    - Start with root (i.e. `main()` function)
    - Continue step by step down to leaves

![Functional Hierarchy vs. Structural Hierarchy](image-url)
Exposing Potential Parallelism

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

- Recoding tasks
  - Partition code
  - Partition data
  - Synchronize dependents

- Recoding transformations
  1. Loop splitting
  2. Cumulative Access Type analysis
  3. Partitioning of vector dependents
  4. Synchronizing dependent variables
    - [DAC’07, TCAD’08]
Exposing Communication

• Why create explicit 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

Computer-aided Recoding, ASPDAC 2010, Taipei, Taiwan
Exposing 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
Exposing 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
Exposing 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()
Interactive Source Recoder

- Implementation
  - Integrated Development Environment (IDE)

- *Cute* tool is a union of
  - Text editor
  - Abstract Syntax Tree (AST)
  - Parser
  - Transformations
  - Code generator
Interactive Source Recoder

• Text editor
  – Interface to the designer
  – Basic and advanced source-code editing
    • C/C++/SpecC
  – Document object
    • Based on Andrew text editor [8]
Interactive Source Recoder

- **Text editor**
- **Abstract Syntax Tree**
  - Captures the structure of the design model
  - Used by transformation tools
  - Complete coverage
    - C and SLDLs
    - Correspondence with document object
  - Can re-generate code in its original form
Interactive Source Recoder

- Text editor
- Abstract Syntax Tree
- Preprocessor and Parser
  - Build AST from text
  - Keep AST in synch
  - Complement the editor
    - Color coding
    - Syntax high-lighting

Diagram:
- Document Object
- AST
- Preproc → Parser
- Code Generator
- Text Editor
- Transformation Tools
- GUI
Interactive Source Recoder

• Text editor
• Abstract Syntax Tree
• Preprocessor and Parser

• Code Generator
  – Generates SLDL source code after transformations
  – Keeps text in synch
Interactive Source Recoder

- Text editor
- Abstract Syntax Tree
- Preprocessor and Parser
- Code Generator

Transformation tools
- Recoding transformations
  - Code partitioning
  - Create structural hierarchy
- Data transformations
  - Variable re-scoping
  - Data structure partitioning
- Analysis
  - Dependency analysis
  - Pointer analysis
Interactive Source Recoder

- Interactive Environment
  - Scintilla + QT + AST + Transformations

- Basic editing
  - Syntax highlighting
  - Auto-completion
  - ...

- Recoding Transformations
  - Dependency analysis
  - Code and data splitting
  - Variable re-scoping
  - Port insertion
  - ...
Experiments and Results

• We have conducted various sets of experiments
• Goals
  – Responsiveness of the “compiler in the editor”
  – Estimated Productivity Gains
    • Extrapolation based on the number of lines of code changed
  – Measured Productivity Gains
    • Class of graduate students
• Design examples
  – GSM Vocoder (voice codec in mobile phones)
  – MP3 Decoder (audio decoder, e.g. iPod)
    • Fixed-point version
    • Floating-point version
  – JPEG Encoder (image encoder, e.g. digital camera)
  – …
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
  - $\ll 1$ sec (on a 3 GHz Linux PC)
  - File I/O overhead (20%)
Experiments and Results

- **Productivity Gain**
  - Creating structural hierarchy
    - Manually
    - estimation
    - Automatically
    - measured

- **Results**
  - Manual time
    - weeks
  - Recoding time
    - minutes

<table>
<thead>
<tr>
<th>Properties</th>
<th>JPEG (1K)</th>
<th>Float-MP3 (3K)</th>
<th>Fix-MP3 (10K)</th>
<th>GSM (10K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of C code</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C Functions</td>
<td>32</td>
<td>30</td>
<td>67</td>
<td>163</td>
</tr>
<tr>
<td>Lines of SpecC code</td>
<td>1.6K</td>
<td>7K</td>
<td>13K</td>
<td>7K</td>
</tr>
<tr>
<td>Behaviors created</td>
<td>28</td>
<td>43</td>
<td>54</td>
<td>70</td>
</tr>
<tr>
<td>Re-Coding time</td>
<td>≈ 30 mins</td>
<td>≈ 35 mins</td>
<td>≈ 40 mins</td>
<td>≈ 50 mins</td>
</tr>
<tr>
<td>Manual time</td>
<td>1.5 weeks</td>
<td>3 weeks</td>
<td>2 weeks</td>
<td>4 weeks</td>
</tr>
<tr>
<td>Productivity gain</td>
<td>120</td>
<td>205</td>
<td>120</td>
<td>192</td>
</tr>
</tbody>
</table>

- Significant productivity gains!
Experiments and Results: Productivity

• Measured Productivity Gains
  – Class of 15 graduate students
  – Recode an MP3 design example
    • Manually (given detailed instructions)
    • Automatically (using the Source Recoder)

• Results

  – Productivity factors vary, but show significant gains!

Creating Structural Hierarchy

Pointer Recoding

Gain

0.0 10.0 20.0 30.0 40.0 50.0 60.0

Gain

0.0 5.0 10.0 15.0 20.0

Student

1 3 5 7 9 11 13 15

Student

1 3 5 7 9 11 13 15
Conclusions

• Embedded System Design
  – Start from higher level of abstraction
  – Need flexible system models in SLDL
• Motivation
  – Automation gap between C reference and SLDL system models
  – 90% of the overall design time spent on “coding” and “re-coding”
  – Need for design automation
• Problem
  – Complete automation is difficult
• Approach
  – Computer-Aided Recoding using Source Recoder
  – Designer-in-the-loop
• Results
  – Significant productivity gains
• Future work
  – Research and develop more transformations
  – Improve interactive graphical environment
References


