Designer-in-the-Loop Recoding to Create Safe Parallel ESL Models

Tutorial SD1: High-Level Specifications to Cope with Design Complexity

Outline

• Embedded System Design Challenge
  – Productivity Gap
  – System Level Modeling Concepts

• Computer-Aided Recoding
  – Introduction and Motivation
  – Recoding Transformations
  – Recoding Analysis

• Prototype Implementations
  – Interactive Source Recoder
  – Eclipse-based Recoding Platform

• Experiments and Results
  – Classroom Case Studies

• Conclusions
Embedded System Design

- Productivity Gap
  
  Hardware design gap
  + Software design gap
  = System design gap

![Graph showing productivity gap](source: "Hardware-dependent Software", Ecker et al., 2009)

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
    - Compose a system of algorithms
  - 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
    - Source code transformations
    - C reference code to SpecC specification model
    - MP3 decoder: 12-14 weeks!
- Automation Gap
  - 90% of overall design time is spent on re-coding!
- Proposal: Automatic Recoding

Problem Definition

- How to get from flat, sequential C code to a flexible, parallel system model?
- Recoding
  - Create structural hierarchy
  - Partition code and data
    - Expose concurrency (parallelize/pipeline)
  - Expose communication
  - Eliminate pointers
  - Make the code compliant to the design tools, …
- Our approach
  - Computer-Aided Recoding
    - Interactive source code transformations

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

![Diagram of C Reference Model and Specification Model](image-url)
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]

Prototype 1: Interactive Source Recoder

- Prototype Implementation (by P. Chandraiah)
  - Integrated Development Environment (IDE)

- *Cute* tool is a union of
  - Text editor
  - Abstract Syntax Tree (AST)
  - Parser
  - Transformations
  - Code generator
Prototype 1: 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
  – …

Prototype 1: 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)
  – …
Prototype 1: Responsiveness

- Why measure Responsiveness?
  - To check feasibility
- Responsiveness
  - Response to designer actions
    - Time to synch AST
      - On editing
    - Time to synch Editor
  - 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%)

Prototype 1: Experimental Results

- Productivity Gain
  - Creating structural hierarchy
    - Manually
      - estimation
    - Automatically
      - measured
- Results
  - Manual time
    - weeks
  - Recoding time
    - minutes
  - Re-Coding time
    - ≈ 30 mins
    - ≈ 35 mins
    - ≈ 40 mins
    - ≈ 50 mins
  - Manual time
    - 1.5 weeks
    - 3 weeks
    - 2 weeks
    - 4 weeks
  - Productivity gain
    - 120
    - 205
    - 120
    - 192

» Significant productivity gains!
Prototype 1: Productivity Gains

• 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!

Recoding for Safe Parallel ESL Models

Recoding Project 1: Creation of Parallel Models

Recoding Project 2: Safe Parallel Models
Recoding for Safe Parallel ESL Models

• Recoding Project 1: Creation of Parallel Models
  – Prototype 1: Interactive Source Recoder
    • by Pramod Chandraiah
  – Focus on designer-controlled source code transformations

➢ Recoding Project 2: Recoding for Safe Parallelism
  – Prototype 2: Eclipse-based Recoding Platform
    • by Xu Han, Weiwei Chen
  – Focus on Advanced Model Analysis
    ➢ Case Study on a Canny Edge Decoder
    • Variable Dependency Analysis
    • Static Parallel Access Conflict Analysis
    • Race Condition Analysis

Eclipse-Based Recoding Platform

• Eclipse Framework
  is an extensible platform to build IDEs

Eclipse SDK: source: Carlson, Eclipse Distilled, 2005
Eclipse-Based Recoding Platform

- Eclipse Framework:
  Integrated Recoding Operations
  - Automatic Compiling
    - Compilation in the background
    - Static design analysis
    - Variable dependency analysis
  - Hierarchy View
    - Behavior hierarchy display
    - Behavior hierarchy navigator
    - Context menu for advanced analysis
  - Non-local Variable View
    - Dependent variables display
    - Conflicting variable access display

Hierarchy View

Non-local Variable View
Eclipse-Based Recoding Platform

Study on browser:
- Parallel Access
- Conflict Analysis

Non-local variable View

Race Condition Browser:
- Parallel Access
Conflict Analysis
Classroom Case Study

- Case Study Experimental Setup
  - A class of 68 graduate students
  - Individually assigned recoding task
  - SpecC-extended Eclipse offered as an *optional* tool
- Assigned task:
  - Recode Canny edge detector from C reference to SpecC SLDL

1. Analysis: `gaussian_smooth` contains 50% of the computation
2. Decision for parallelization: parallelize `gaussian_smooth`
3. Structure and variable recoding

```c
#include <math.h>

float *gaussian_smooth(float *in_img, int rows, int cols, float sigma, int int_rows, int int_cols, int window_size, int center) {
    float temp_img[row_size][col_size], k = WINSIZE * dot_sum;
    /* Create a 1-dimensional gaussian smoothing kernel */
    make_gaussian_kernel(sigma, kernel, &window_size);
    center = window_size / 2;

    /* Blur in the x-direction */
    for (int r = 0; r < rows; r++)
        for (int c = 0; c < cols; c++)
            if ((c - center) >= 0 && (c - center) < col_size)
                temp_img[r][c] = dot_sum;
    /* Blur in the y-direction */
    for (int r = 0; r < int_rows; r++)
        for (int c = 0; c < int_cols; c++)
            if ((r - center) >= 0 && (r - center) < int_rows)
                temp_img[r][c] = dot_sum;

    return temp_img;
}
```

Structure Recoding

- **prep**
  - BlurX
    - 0, 0, 0, 0
  - BlurY
    - 0, 0, 0, 0

```c
float *gaussian_smooth(float *in_img, int rows, int cols, float sigma, int int_rows, int int_cols, int window_size, int center) {
    float temp_img[row_size][col_size], k = WINSIZE * dot_sum;
    /* Create a 1-dimensional gaussian smoothing kernel */
    make_gaussian_kernel(sigma, kernel, &window_size);
    center = window_size / 2;

    /* Blur in the x-direction */
    for (int r = 0; r < rows; r++)
        for (int c = 0; c < cols; c++)
            if ((c - center) >= 0 && (c - center) < col_size)
                temp_img[r][c] = dot_sum;
    /* Blur in the y-direction */
    for (int r = 0; r < int_rows; r++)
        for (int c = 0; c < int_cols; c++)
            if ((r - center) >= 0 && (r - center) < int_rows)
                temp_img[r][c] = dot_sum;

    return temp_img;
}
```
Classroom Case Study

• Recode the `gaussian_smooth` function

```c
// Recode the gaussian_smooth function

gaussian_smooth (unsigned char *imgin, int rows,
int cols, float sigma, short int *smoothing)
{
    int x, y, center,
    float temp[SIZE], kernel[WINSIZE], dot, sum;
    /
    Create a 2-dimensional gaussian smoothing kernel
    
    make_gaussian_kernel(sigma, kernel, WINSIZE, center = windowsize / 2);
    
    // Blur in the x-direction
    for(y=0; y<cols; y++)
    {
        dot = 0.0;
        sum = 0.0;
        for(center=0; center<cols; center++)
        {
            if((center == 0) || (center == cols-1))
                dot += temp[center] * kernel[center];
            else
                dot += temp[center] * kernel[center];
        }
        sum += kernel[center];
        smoothing[0] = (short int)(dot*FACTOT/sum + 0.5);  
    }

    // Blur in the y-direction
    for(x=0; x<cols; x++)
    {
        dot = 0.0;
        sum = 0.0;
        for(center=0; center<cols; center++)
        {
            if((center == 0) || (center == cols-1))
                dot += temp[center] * kernel[center];
            else
                dot += temp[center] * kernel[center];
        }
        sum += kernel[center];
        smoothing[1] = (short int)(dot*FACTOT/sum + 0.5);  
    }
}
```

Variable Recoding

```
int BlurX, BlurY, Prep;

int gaussian_smooth()
{  
    int x, y, center,
    float temp[SIZE], kernel[WINSIZE], dot, sum;
    /
    Create a 2-dimensional gaussian smoothing kernel
    
    make_gaussian_kernel(sigma, kernel, WINSIZE, center = windowsize / 2);
    
    // Blur in the x-direction
    for(y=0; y<cols; y++)
    {
        dot = 0.0;
        sum = 0.0;
        for(center=0; center<cols; center++)
        {
            if((center == 0) || (center == cols-1))
                dot += temp[center] * kernel[center];
            else
                dot += temp[center] * kernel[center];
        }
        sum += kernel[center];
        smoothing[0] = (short int)(dot*FACTOT/sum + 0.5);  
    }

    // Blur in the y-direction
    for(x=0; x<cols; x++)
    {
        dot = 0.0;
        sum = 0.0;
        for(center=0; center<cols; center++)
        {
            if((center == 0) || (center == cols-1))
                dot += temp[center] * kernel[center];
            else
                dot += temp[center] * kernel[center];
        }
        sum += kernel[center];
        smoothing[1] = (short int)(dot*FACTOT/sum + 0.5);  
    }
}
```

Classroom Case Study

• Recode the `gaussian_smooth` function

```c
// Recode the gaussian_smooth function

gaussian_smooth (unsigned char *imgin, int rows,
int cols, float sigma, short int *smoothing)
{
    int x, y, center,
    float temp[SIZE], kernel[WINSIZE], dot, sum;
    /
    Create a 2-dimensional gaussian smoothing kernel
    
    make_gaussian_kernel(sigma, kernel, WINSIZE, center = windowsize / 2);
    
    // Blur in the x-direction
    for(y=0; y<cols; y++)
    {
        dot = 0.0;
        sum = 0.0;
        for(center=0; center<cols; center++)
        {
            if((center == 0) || (center == cols-1))
                dot += temp[center] * kernel[center];
            else
                dot += temp[center] * kernel[center];
        }
        sum += kernel[center];
        smoothing[0] = (short int)(dot*FACTOT/sum + 0.5);  
    }

    // Blur in the y-direction
    for(x=0; x<cols; x++)
    {
        dot = 0.0;
        sum = 0.0;
        for(center=0; center<cols; center++)
        {
            if((center == 0) || (center == cols-1))
                dot += temp[center] * kernel[center];
            else
                dot += temp[center] * kernel[center];
        }
        sum += kernel[center];
        smoothing[1] = (short int)(dot*FACTOT/sum + 0.5);  
    }
}
```

Variable Dependency Analysis

- Designer-controlled options
  - re-locate
  - localize
  - duplicate
  - channels/ports

Tutorial SD1, ASPDAC '14, Singapore
January 20, 2014
Singapore
Classroom Case Study

• Case Study Results

Student Distribution

- Eclipse 27%
- non-Eclipse 34%
- no valid response 32%
- hybrid 7%

Student Ratings

- very useful 52%
- useful 35%
- somewhat useful 9%
- did not use 4%

Hierarchy View

- not useful 0%
- did not use 4%
- somewhat useful 9%
- useful 35%
- very useful 52%

Non-local Variable View

- not useful 4%
- did not use 9%
- somewhat useful 39%
- useful 39%
- very useful 9%
Classroom Case Study

- Case Study Results

Effect on Working Time and Correctness

Eclipse users: needed less time, yet made less mistakes!

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 gains in productivity
  - Significant improvements in correctness

- Future work
  - SystemC!
References


