Compiling for Fine-Grain Concurrency: Planning and Performing Software Thread Integration

RTSS 2002 -- December 3-5, Austin, Texas
Alex Dean
Center for Embedded Systems Research

Dept. of ECE, NC State University
Alex_Dean@ncsu.edu  http://www.cesr.ncsu.edu/agdean/
Overview

• STI Background
  – Hardware to software migration
  – Code Transformations

• STI Methods
  – Preparation
  – Planning Integration
  – Performing Integration

• Experiments
  – Target System
  – Analysis
  – Integration
  – Results
    • Code Expansion
    • Performance Gains

• Current and Future Work
Hardware to Software Migration

• What is it?
  – Replacing hardware components with software functions on a conventional CPU (uniprocessor)

• Why do it?
  – Custom HW is too expensive unless production volume is high
  – Cost, size, reliability, weight, power
  – Function availability, time to market, field upgrades

• Who does it?
  – Everyone

• Why do they hate it so passionately?
  – Code in assembler and it takes forever to develop
    • Tools are real-time-ambivalent
  – Code in C and you can’t get decent performance, but you finish coding sooner
    • Slow context switches and scheduling
    • Tools are real-time-oblivious
Software Thread Integration for HSM

- **Thread-Level - Basic STI**
  - Set of transforms which interleave code freely and efficiently based on *sub-thread timing requirements*
  - Can trade off code size for execution efficiency
  - Txs implemented in compiler

- **System-Level - STI for HSM**
  - Choosing guests - finding idle time
  - Choosing hosts - finding temporal determinism
  - Matching threads to integrate - predicting system efficiency and performance
  - Triggering guests to meet *thread timing requirements* - latency and resources

- **Result is greater efficiency**
  - Integrated threads more efficient
  - Integration process automated
Thrint

foo.s → Control-flow Analysis → Data-flow Analysis → Static Timing Analysis → Integration Analysis → Integration

GProf

foo.id

XVCG

GnuPlot

THIS TRANSMogrIFIER WILL TURN YOU INTO ANYTHING AT ALL.

Eel Baboon Bug Dinosaur
Detailed View

Control-Flow and Control-Dependence Analysis

Data-Flow Analysis (w/loop iteration analysis)

Static Timing Analysis

Planning Integration
- Single Events
- Looping Events

Performing Integration
- Single Events
- Looping Events

Code Regeneration
Single Event Integration - Loop Node

- **Guard Guest**
  - (Optimize for Space)

- **Split Loop**
  - (Optimize for Speed)
Loop Integration - Guest Loop Slower

Original Loops

Host

Guest

Guest Iteration (Real-Time)

Idle Time

Busy-wait Padding

Host Iteration

Unroll Host
1. Identify Guest Nodes

PROCEDURE VidRef_LineDraw__Fv_en INTO VidRef_LineDraw__Fv_en
   TOLERANCE 0
   BLOCK Video_Reset_Ptr AT 20
   LOOP Video_Loop PERIOD 40 ITERATION_COUNT 128
      BLOCK Video_Pix0 INTO_LOOP Video_Loop FIRST_AT 90
      BLOCK Video_Pix1 INTO_LOOP Video_Loop FIRST_AT 100
      BLOCK Video_Pix2 INTO_LOOP Video_Loop FIRST_AT 110
      BLOCK Video_Pix3 INTO_LOOP Video_Loop FIRST_AT 120
   BLOCK Video_End IMPLICIT
END
2. Analyze Host and Guest Functions

- Data Flow Analysis
  - Register reallocation partitions register file
  - Loop iteration counts extracted

- Static Timing Analysis
  - Predict best and worst case execution times
  - Find temporally deterministic host segments
  - Find idle time in guests
3.1 Plan Integration (w/o Loop Fusion)

For each explicit single event guest in integration directives list
- Find host node which will be executing (in current segment)

Node::Find Hosts(target_time, int_plan)

If this node finishes too early
    continue to next node
Else
    pad node start jitter
    if this node ends in target time
        plan to put guest AFTER this node
    else descend into CDG, switch on this->type
        CODE: plan to put guest WITHIN this node
        PREDICATE: for each condition TV
            get_first_child(TV)->find_hosts

LOOP:
    if guest node is not in loop
        find hosts in loop (will plan to split loop or guard guest as needed)
    else
        find hosts (loop fusion has already been planned, so just find correct location without any transformations)
3.b Plan Loop Fusion

• for each explicit looping guest event in integration directives list
  - while guest loop iterations remain
    • find time covered by guest iters
    • if any host loops overlap, for each host loop
      - if guest loop starts first
        » peel preceding iters from guest loop
        » plan integration as non-looping guests
      - else if host loop starts first
        » plan to unroll guest or host loop
        » mark for fusion
        » if loop iterations remain
        » mark for clean-up loops
  - else no host loops during guest loop
    - peel all guest loop iterations
    - plan their integration as non-looping guests
Target Host Nodes Are Now Identified
4. Integrate

- Det_Segment::Integrate(plan)
  - for each top_level_guest
    - if guest is loop,
      - for each host
        » if has a loop
          transform, do it
        » make guarded copy of
          guest loop after host
          loop if needed
      - cur_guest <=
        top_level_guest’s first child
    - else cur_guest <=
      top_level_guest
  - else cur_guest <=
    top_level_guest

- do
  - pad previous nodes (start jitter)
  - pad host node (end jitter)
  - for each host cur_host
    - do host loop transforms
    - update cur_host based on
      splitting, prev guests, etc.
    - insert implicit guests and
      explicit guest
    - advance cur_host
  - if cur_guest in guest loop
    - advance to next guest
  - for each top_level_guest
    - create fused loop control
tests
Application: NTSC Video Signal Generator

- Predictable 100 MIPS CPU (Alpha ISA)
- Fixed memory access time (on-chip memory)
- 512x494 resolution image
- Generate stream of pixels (100 ns each) and sync signals in SW to feed a DAC
- **Coarse-grain** idle time handled by timer-based ISRs (vertical sync, serration, equalization pulses)
- **Fine-grain** idle time handled by integration (between pixels)
Software Architecture

- Application defers work via queues
  - Line and sprite drawing functions split into enqueing stub, dequeuing service routine
  - Interrupt service routine triggers integrated functions for rendering and display refresh (host work fills idle inter-pixel time)
  - If both queues are empty, ISR calls dedicated video refresh function (nops fill idle inter-pixel time)
Initial Integration

DrawLine

DrawSprite

Video Refresh

Video Refresh
Final Integration

**DrawLine**

**DrawSprite**
Performance

Pixels Rendered per Video Row Refreshed

- Line Drawing: 31 (Discrete), 101 (Integrated)
- Circle Drawing: 57.8 (Discrete), 190 (Integrated)
- Sprite Drawing: 180 (Discrete), 690 (Integrated)

CPU Time Remaining

- Discrete Draw Line
- Integrated Draw Line
- Discrete Draw Sprite
- Integrated Draw Sprite

Pixels Drawn per Video Row Refreshed

0% 5% 10% 15% 20% 25% 30% 35%

0 200 400 600
Code expansion is limited to integrated functions, not entire application.
Conclusions

- Introduced techniques to plan and perform STI
- STI improves HSM
  - Product (Code)
    - Reclaims idle time better than traditional HSM methods
    - Helps designers meet embedded system constraints
  - Process (Development)
    - Automated, built into post-pass compiler
    - Provides a sharp tool, allows development at higher level (i.e. faster)
- Current Work: Extend concepts of STI
  - HSM for bit-banged communication protocols
  - Retargeted Thrint to support AVR 8-bit microcontroller
    - Communication protocols
    - Encryption
    - Video Generation
  - Integration to boost ILP and performance on wide CPUs
- Alex_Dean@ncsu.edu, http://www.cesr.ncsu.edu/agdean/