Synthesis for Multicore Protocols
Breakout Session Report

Milo Martin
University of Pennsylvania
Synthesis of Multicore Protocols
Breakout Session

- Earlier presentation
  - “Transit for synthesis of distributed protocols” by Abhishek Udupa (Penn)

- Full presentations
  - “Synthesis-Based Compiler for Low-Power Multicore Architectures” by Mangpo Phitchaya Phothilimthana (UC-Berkeley)
  - “Device Drivers” by Susmit Jha (Intel)
  - “Automated Synthesis of Protocols using Flows” by Murali Talupur (Intel)

- Shorter presentations
  - “Synthesizing the Alternating-bit Protocol with TRANSIT” by Stavros Tripakis (Berkeley)

- Interspersed discussion
GreenArrays Spatial Processors

Specs
- Stack-based 18-bit architecture
- 32 instructions
- 8 x 18 array of asynchronous computers (cores)
- No shared resources (i.e. clock, cache, memory). Very scalable architecture.
- Limited communication, neighbors only
- < 300 byte memory per core

Example challenges of programming spatial architectures like GA144:
- **Bitwidth slicing:** Represent 32-bit numbers by two 18-bit words
- **Function partitioning:** Break functions into a pipeline with just a few operations per core.

---

Finite Impulse Response Benchmark

GA144 is **11x faster** and simultaneously **9x more energy efficient** than MSP 430.

*Data from Rimas Avizienis*

---

Figure from Per Ljung

~100x

# of Instructions/second vs Power
Synthesis-Based Compiler for Low-Power Multicore Architectures

1. New Programming Model

2. New Approach Using Synthesis

High-level Program

Partitioner

with partition annotation

Location Mapper

with location annotation & routing info

Communication Code Generator

Per-core program with communication code

Code Generator

Per-core optimized machine code
Optimal Partitions from Our Synthesizer

- Benchmark: simplified MD5 (one iteration)

- Partitions are automatically generated.

### 256-byte mem per core

**Initial data placement specified**

- **high**: 102 103 105 106
- **low**: 2 3 5 6

### 512-byte mem per core

- **same initial data placement**
- **different initial data placement**
Synthesis-Based Compiler for Low-Power Multicore Architectures

- Programming for super-low power processors (GreenArray)
- Synthesis of software for such a target requires partitioning, mapping, and low-level code generation
  - Partitioning problem creates, in essence, a distributed protocol
Study of Opportunities for Synthesis in Device Drivers

- Device drivers large source of operating systems bugs and crashes

- Recapped recent work on understanding device drivers
  - Most of the code is not what you would think of for traditional device drivers
  - Lots of code in configuration & initialization (one driver, many chipsets)
  - But also power management mechanisms & policy (likely to increase)
  - Development cycle shortening (was 2 years; now is 9 months)

- Many bugs are actually concurrency bugs
  - Partly due to simultaneous design of hardware and software using models
  - Functional models don’t always capture all timing/interleaving behaviors

- Device drivers were early success story in software model checking
  - Ripe for early success of synthesis?
Automated Synthesis of Protocols using Flows

- Hardware protocol design flow

- Step 1
  - “Architect” generates human-language specification with example flows
  - Unwilling to write executable model or code of protocol

- Step 2, in parallel:
  - “Designer” translates specification into hardware-level implementation
  - “Validator” generates test cases for design validation
  - “Verifier” uses formal methods to verify protocol

- Idea: use flows from informal spec. to create a model for step 2

- Abhishek Udupa (presented TRANSIT) interning at Intel this summer
TRANSIT Approach to Specification

- Snippets
- Invariants
- Programmer Inputs

Model Check/Verify

Counter-example

Verified Protocol

TRANSIT
Synthesizing the Alternating-bit Protocol with TRANSIT

- Student Antti Halme used TRANSIT to specify this protocol
- Methodology and synthesis successful
- Explored ways to perform counterexample highlighting
- Identified liveness checking as current limitation in TRANSIT tool
Discussion & Take Aways

- Synthesis of software low-level multicore requires partitioning, mapping, and low-level code generation
  - Partitioning problem creates, in essence, a distributed protocol
  - Many synthesis opportunities

- Device drivers potential application for synthesis
  - Potential for hardware/software co-design synthesis, too

- Industrial protocol design splits “architect” from “design”
  - Supports TRANSIT’s focus on protocol specification (rather than architecting)
  - Use of “flows” rather than just snippets for protocol synthesis
    - Synthesize more of the protocol (e.g., skeleton, states, events)
    - What is the best way to identify the source of a bug in such methodologies?

- Someone other than tool author used TRANSIT successfully