Software Synthesis and Co-Design based on an Asynchronous Concurrency Model

Bill Lin
Electrical and Computer Engineering
University of California, San Diego
billlin@ece.ucsd.edu
Acknowledgments

• Industrial interactions
  – Qualcomm Inc.
  – Hughes Network Systems
  – Nokia Design Center

• Group members:
  – Ranjita Bhagwan
  – James Hurt
  – Andrew May
  – Xin Wang
  – Xiaohan Zhu

Picasso Project
Billion Transistors Era

TI’s roadmap for 2001 [EB News, 8/31/98]
• 0.07\textmu m effective channel length
• Copper interconnect
• 1 GHz clock frequency

\Rightarrow 400 million transistors on a single chip!

ARM9TDMI, latest 32-bit ARM RISC CPU
• 111K transistors

\Rightarrow 400 million transistors = 3,603 ARM9 CPUs
System-on-a-Chip

Single-Chip Digital Baseband Processor
Outline

• Model of computation
• Software synthesis
• Hardware synthesis
• Hardware/software co-design
# Models of Computation

## User’s Perspective

### Asynchronous Models
- Java, Modula-3, Ada, Occam (CSP) …
- used for concurrent software
- supported by multi-tasking OS

### Synchronous Models
- VHDL, Verilog: hardware
- Esterel: reactive programming
- software synthesis, logic synthesis

### Data-Flow Models
- DSP designs
- hardware: Cathedral, Hyper, …
- system synthesis, software: Ptolemy, Cossap, SPW, …
Programming Model

- Model = “C” + CSP
  - C-like basic constructs: easy to learn.
  - CSP (Concurrent Sequential Processes) [Hoare’85] model of concurrency and communication. Formally and rigorously defined.

Communication: send and receive along channels
P1 (input chan(int) a, output chan(int) b)
{
    int x, t;
    for (;;) {
        x = <-a; // receive
        if (x < 0) {
            x = 10 - x;
        } else {
            x = 10 + x;
        }
        b<- = x; // send
    }
}

P2 (input chan(int) b, output chan(int) a)
{
    int y, z = 0;
    for (;;) {
        a<- = 10; // send
        y = <-b; // receive
        z = (z + y) % 345;
    }
}
Programming Model

- True concurrent threads of control with conditional execution and data-dependent loops
- Can model both control and data computations
Outline

• Model of computation
• Software synthesis
• Hardware synthesis
• Hardware/software co-design
Traditional Approach

• Real-Time Operating Systems (RTOS) widely used to support multi-tasking, but …

- On-chip RAM = $$$
- Portability depends on RTOS

- RTOS
- CPU

- Task Task ... Task

• Alternative approach: statically schedule the computation at compile-time

Context-switching and communication = performance overhead
Main Contribution

• Static scheduling
  – Input: CSP-based model
  – Output: Ordinary sequential C

• Advantages
  – Portable to different processors with conventional C compilers
  – Avoids memory and performance overhead of RTOS
  – Compiler optimizations across processes
Intermediate Representation

- Hierarchical construction using Petri net algebra
  [DAC’94, ISSS’97]
Synthesis Procedure

• Systematically generate acyclic Petri net segments
• Statically schedule operations (transitions) in each segment
• Generate control-flow graph induced by static schedule
• Generate C code from the control-flow graph
Maximal Expansions

• Cut-off places correspond to set of places encountered when a cycle has been reached

• Corresponding acyclic Petri net segment is called a Maximal Expansion with respect to m
Cut-Off Markings

- Reachable markings from $m$ with no enabled transitions: $\text{CM}(E)$
- e.g. $m = (p_1,p_2)$, $\text{CM}(E) = \{ (p_1,p_2), (p_3,p_4) \}$
Expansion on New Markings

- Until no new cut-off markings. Convergence guaranteed. e.g. $m = (p3,p4)$, $CM(E) = \{(p3,p4)\}$. Stop.
Static Schedule

• Definition: $\pi : T \rightarrow N$ such that if $t_1$ precedes $t_2$, then $\pi(t_1) < \pi(t_2)$

• Given $\pi$, use modified Petri net firing rules to generate reachability graph
Optimized Code Generation

• Code optimization across processes possible

```
generated-program ( )
{
    int x, y, z = 0;
p1p2:
    x = 10;
    if (x < 10) x = 10 - x;
    else x = 10 + x;
y = x;
z = (z + y) % 345;
goto p1p2;
}
```

```
generated-program ( )
{
    int z = 0;
p1p2:
    z = (z + 20) % 345;
goto p1p2;
}
```

e.g. after constant propagation

• Sophisticated state-of-the-art optimizing C compilers can exploit instruction-level parallelism (e.g. super-scalar, pipelined, VLIW, EPIC CPUs)
Experimental Results

• RC5 encryption chain example

\[
\begin{align*}
\text{RC5 Encrypt} & : \\
\text{In} & \quad \rightarrow \quad \text{Encrypt} \\
\text{Out} & \quad \quad \rightarrow \quad \text{RC5 Decrypt}
\end{align*}
\]

\[
\ldots \\
// r = \# \text{ of rounds} \\
\text{while (i <= r) \{} \\
\quad A = \text{ROTL}(A^B, A) + S[2*i]; \\
\quad B = \text{ROTL}(B^A, B) + S[2*i+1]; \\
\quad i++; \\
\}\ \\
\ldots
\]

• Contains data-dependent loops and mixed control-data computations
Implementation and Results

• Implementation: C generator
  – Synthesis = pre-processor to C
  – Threads = multi-tasking using C + Solaris Threads (can port to other thread packages or RTOS)

• Results

<table>
<thead>
<tr>
<th>Size</th>
<th>Synthesis</th>
<th>Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>2MB</td>
<td>2.0</td>
<td>34.7</td>
</tr>
<tr>
<td>8MB</td>
<td>6.1</td>
<td>103.5</td>
</tr>
<tr>
<td>32MB</td>
<td>21.7</td>
<td>554.5</td>
</tr>
</tbody>
</table>

| Rate   | 1.51MB/s  | 58KB/s  |
Java Generator

• Synthesis
  – Generate Java instead of C
  – No usage of Java Threads and Monitors
  – Only need (Embedded) Java VM “minus” Java Threads and Monitors

• Threads
  – Processes and channels mapped to Java Threads and Monitors
  – Need Java VM that supports Java Threads and Monitors
Outline

- Model of computation
- Software synthesis
- Hardware synthesis
- Hardware/software co-design
**Hardware Synthesis**

- **Procedure**
  - Group processes together into Petri nets (degenerate case: one process, one Petri net)
  - Apply handshake expansion to each Petri net for external communications
  - Apply static scheduling of each Petri net to synthesize state machine
  - Convert state machine to behavioral VHDL (Verilog)
  - Apply VHDL (Verilog) synthesis
Handshake Expansion

• Examples:
  – request / acknowledge protocol
  – sender_ready / receiver_ready protocol
  – on-chip bus protocols
Hardware Synthesis

• Commercial tools based on earlier high-level synthesis research
  – IMEC, Berkeley, IBM, Irvine, CMU, USC, …

• Cycle-true synthesis (high-level RT synthesis)
  – retains one-to-one correspondence between states and clock cycles

• Behavioral synthesis
  – introduces “micro-cycles” as extra degree of freedom
Embedding VHDL / Verilog

- Embedded VHDL / Verilog component is encapsulated using handshake protocol

```vhdl
native P1(input chan(int) a, b,  
          output chan(int) c);

native P4(input chan(int) a);

P2(input chan(int) a,  
    output chan(int) b, c) {  
    ...  
}

P3(input chan(int) a,  
    output chan(int) b, c) {  
    ...  
}

system (input chan(int) cin,  
        output chan(int) cout)  
{  
    chan(int) c1, c2, c3;  
    par {  
        P1 (cin, c1, c2);  
        P2 (c2, c1, c3);  
        P3 (c3, c4, cout);  
    }  
}
```
Outline

• Model of computation
• Software synthesis
• Hardware synthesis
• Hardware/software co-design
Problem
CoWare Approach

CoWare = C compilation + HDL synthesis
+ Interface Synthesis
[IEEE’97, DAES’97, DAC’96, EuroDAC’96]
New Approach

• Permits direct programming of software and hardware components when appropriate
• Permits the embedding of C / Java and VHDL / Verilog for the development of software and hardware components, respectively, when appropriate
• Permits the use of model as a scripting language to glue together components, including non-trivial glue logic behavior
• Builds upon interface synthesis and co-simulation solutions from CoWare project
### Putting it all Together

<table>
<thead>
<tr>
<th>Concurrent Specification, including embedding of C / Java code and VHDL / Verilog components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Synthesis</td>
</tr>
<tr>
<td>C / Java Compilation</td>
</tr>
<tr>
<td>Interface Synthesis</td>
</tr>
</tbody>
</table>

![Diagram of integrated circuits and system level design capabilties]
Thank You

billlin@ece.ucsd.edu