Module 3: Behavioral Synthesis
Lecture 3.15: Behavioral Transformations

Anand Raghunathan
raghunathan@purdue.edu

Fall 2014, ME 1052, T Th 12:00PM-1:15PM
Behavioral Synthesis: Why Advanced Techniques?

- One of the reasons for poor adoption of 1\textsuperscript{st} generation behavioral synthesis tools
  - Automatically synthesized RTL did not measure up to manual designs!
- Focus turned to improving the quality of synthesis results
  - Borrow advanced optimization techniques from microprocessors, compilers
Advanced Techniques in Behavioral Synthesis: Overview

• Behavioral transformations
  – Arithmetic transformations
  – Control-flow transformations
  – Loop transformations (unrolling, re-ordering)

• Scheduling with speculative execution

• Pipelining
  – Synthesis with pipelined functional units
  – Synthesis of pipelined data paths
  – Loop pipelining

• Other advanced techniques
  – SIMD units and Variable-latency units
  – Common-case optimized circuits
Behavioral Transformations

• Optimize the behavioral description before scheduling and resource sharing
  – Arithmetic transformations
  – Control-flow transformations
  – Loop transformations

• Largely inspired by high-performance compilers
Simple Transformations

- **Constant propagation (a.k.a. constant folding)**

  ```c
  ... u = 1 << 8; y = u + x; ... 
  ...
  ...
  ```

- **Strength reduction**

<table>
<thead>
<tr>
<th>Original</th>
<th>Strength-reduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>y = x / 8</td>
<td>y = x &gt;&gt; 3</td>
</tr>
<tr>
<td>y = x * 64</td>
<td>y = x &lt;&lt; 6</td>
</tr>
<tr>
<td>y = x * 2</td>
<td>y = x + x OR y = x &lt;&lt; 1</td>
</tr>
<tr>
<td>y = x * 15</td>
<td>y = (x &lt;&lt; 4) - x</td>
</tr>
</tbody>
</table>
Arithmetic Transformations

- Commutativity, Associativity, Distributivity

- Common sub-expression elimination
**Arithmetic Transformations**

- **Tree height reduction**

  - $O(n)$ critical path
  - $O(\log n)$ critical path

  - $a + b \times c + d \rightarrow (a + d) + b \times c$
  - $a \times (b \times c \times d + e) \rightarrow a \times b \times c \times d + a \times e$
Control-Flow Transformations

• Flattening nested conditionals

```c
if(c1) {
    if(c2) {
        if(c3) {
            x = a+b;
        }
    }
}
```

```c
if(c1 && c2 && c3) {
    x = a+ b;
}
```

• Conditional expansion / predication

```c
if (a) {
    x = b + d;
} else {
    x = bd;
}
```

```c
x = !(a == 0) * (b + d) + (a == 0) * bd
```
Control-Flow Transformations

- Code motion across conditionals

```c
if (c1) {
    x = a + b;
    y = x * d;
}
```

```c
x1 = a + b;
if (c1) {
    x = x1;
    y = x * d;
}
```

Code motion introduces speculative execution into the behavior!
Loop Transformations

- **Loop Unrolling**

```plaintext
FOR i IN 0 to 3 LOOP
    result := result + S(i);
END LOOP;
```

\[ \text{result} + S(0) + S(1) + S(2) + S(3) \Rightarrow \text{result} \]

- **Loop Re-ordering**

```plaintext
for j from 0 to 10
    for i from 0 to 20
        a[i,j] = i + j
```  

```plaintext
for i from 0 to 20
    for j from 0 to 10
        a[i,j] = i + j
```  

Many other loop transformations exist, such as loop fusion, iteration space tiling, ...