Derivation of Efficient FSM from Polyhedral Loop Nests

Tomofumi Yuki, Antoine Morvan, Steven Derrien
INRIA/Université de Rennes 1
High-Level Synthesis

- Writing HDL is too costly
  - Led to emergence of HLS tools
- HLS is sensitive to input source
  - Must be written in “HW-aware” manner
- Source-to-Source transformations
  - Common in optimizing compilers
  - (semi-)automated exploration at HLS stage
  - Further enhance productivity/performance
HLS Specific Transformations

- Not all optimizing compiler transformations make sense in embedded context
  - Its converse is also true
- Finite State Machines is an example

for loops are preferred in general purpose context

```c
for i
for j
    S0
for k
    S1
```

```c
while (...)
    if (...)
        S0;
    if (...)
        S1;
    if (...)
        k = k+1;
    if (...)
        i=i+1; j=0;
```
Contributions

- Analytical model of Loop Pipelining
  - Understanding when to use Nested LP
  - w.r.t. Single Loop Pipelining
- Derivation of Finite State Machines
  - Handles imperfectly nested loops
  - Based on polyhedral techniques
- Pipelining of the control-path
  - Computing $n$-states ahead
  - Improves performance of the control-path
Outline

- Modeling Loop Pipelining
  - Single Loop Pipelining
  - Nested Loop Pipelining
  - NLP vs SLP
- FSM Derivation
- Evaluation
- Conclusion
Single Loop Pipelining

- Overlapped execution of innermost loop

```
for i=1:M
  for j=1:N
    stage0(i,j);
    stage1(i,j);
    stage2(i,j);
    stage3(i,j);

for i=1:M
  s0(i,1);
  s1(i,1); s0(i,2);
  s2(i,1); s1(i,2); ...
  s3(i,1); s2(i,2); ... s0(i,N);
    s3(i,2); ... s1(i,N);
      ... s2(i,N);
        s3(i,N);
```
Pipeline flush/fill Overhead

- Overhead for each iteration of the outer loop

```plaintext
for i=1:M  
  for j=1:N  
    s0(i,j);  
    s1(i,j);  
    s2(i,j);  
    s3(i,j);
```
Nested Loop Pipelining

“Compress” by pipelining alltogether

\[
\begin{align*}
&\text{for } i=1:M \\
&\quad \text{for } j=1:N \\
&\quad \quad s0(i,j); \\
&\quad \quad s1(i,j); \\
&\quad \quad s2(i,j); \\
&\quad \quad s3(i,j); \\
&\end{align*}
\]

\[
\begin{align*}
&\text{for } i=1:M \\
&\quad j=j+1; \ j<N \\
&\quad s0(i,j); \\
&\quad s1(i,j); \\
&\quad s2(i,j); \\
&\quad s3(i,j); \\
&\end{align*}
\]

while(has_next)

\[
\begin{align*}
&\text{i,j}=\text{next}(i,j) \\
&\quad s0(i,j); \\
&\quad s1(i,j); \\
&\quad s2(i,j); \\
&\quad s3(i,j); \\
&\end{align*}
\]
NLP Overhead

- Larger control-path
  - FSM for loop nest, instead of a single loop
  - FSM for SLP is a simple check on loop bound
- Hinders maximum frequency
  - Complex control-path may take longer than one data-path stage
  - Savings in flush/fill overhead must be greater than the loss in frequency
Modeling Trade-offs

- Important parameters:
  - Frequency Degradation due to NLP
  - Innermost trip count
  - Number of pipeline stages
- $f^*$: NLP frequency normalized to SLP
  - $f^* = 0.9$ means 10% degradation in frequency
- $\alpha = \#\text{stages} / \text{trip count}$
  - larger $\alpha$ means large flush/fill overhead
When is NLP Beneficial?

<table>
<thead>
<tr>
<th>α: larger = small trip count (innermost)</th>
<th>f*: higher = less degradation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>0.12</td>
</tr>
<tr>
<td>0.4</td>
<td>0.14</td>
</tr>
<tr>
<td>0.6</td>
<td>0.16</td>
</tr>
<tr>
<td>0.8</td>
<td>0.18</td>
</tr>
<tr>
<td>1</td>
<td>0.20</td>
</tr>
<tr>
<td>1.2</td>
<td>0.22</td>
</tr>
<tr>
<td>1.4</td>
<td>0.24</td>
</tr>
<tr>
<td>1.6</td>
<td>0.26</td>
</tr>
<tr>
<td>1.8</td>
<td>0.28</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
</tr>
</tbody>
</table>

Program Characteristic (cannot change)

Improving control-path is possible
Outline

- Modeling Loop Pipelining
- FSM Derivation
  - Polyhedral Representation
  - Computing Transitions
  - State Look Ahead
- Evaluation
- Conclusion
Polyhedral Representation

- Represent loops as mathematical objects

\[
\begin{align*}
\text{for } i &= 0 : N \\
\text{for } j &= 0 : M \\
S
\end{align*}
\]

\[
i, j \in \{0 \leq i \leq N \land 0 \leq j \leq M\}
\]

\[
\begin{align*}
\text{for } i &= 0 : N \\
\text{for } j &= 0 : i \\
S0 \\
\text{for } k &= 0 : N - i \\
S1
\end{align*}
\]

\[
i, j \in \{0 \leq i \leq N \land i \leq j \leq N\} \lor \{0 \leq i \leq N \land 0 \leq j \leq N - i\}
\]
FSM Derivation

- **next function**
  - Find a *piece-wise* function that gives the immediate successor in lexicographic order
  - Proposed in 1998 for low-level code generation

- Direct Application to FSM
  - Each piece = condition of transition
  - Function = transition

- Can be composed to obtain $\text{next}^n$
State Look Ahead

- Pipelining the control-flow
  - When data-path is heavily pipelined, control-path becomes the critical path
- Computing $n$-states ahead
  - Allows $n$-stage pipelining of the control-path
Other Optimizations

- **Merging transitions**
  - `next` computed can have many transitions
  - Some can be merged by looking at its context

\[
\begin{align*}
  \text{next}(i,j) &= (i,j+1) \text{ if } i < N \\
  \text{next}(i,j) &= (N,j+1) \text{ if } i = N
\end{align*}
\]

- **Common Sub-expressions**
  - HLS tools sometimes fail to catch

\[
\begin{align*}
  \text{if } (a>b \land c>d) & \text{ A; } \\
  \text{if } (a>b \land e>f) & \text{ B; }
\end{align*}
\]

\[
\begin{align*}
  x &= a>b; \\
  \text{if } (x \land c>d) & \text{ A; } \\
  \text{if } (x \land e>f) & \text{ B; }
\end{align*}
\]
Evaluation Methodology

- Focus on control-path
  - empty data-path (incrementing arrays)
  - independent iterations
  - loops with different shapes
- 3 versions: different pipelining
  - SLP: innermost loop
  - NLP: all loops
  - FSM-LA2: while loop of FSM with next

\[^2\]
Evaluation: HLS Phase

Maximum Target Frequency

- rect 2d
- rect 3d
- triangular 2d
- triangular 3d

- SLP
- NLP
- FSM-LA2
Evaluation: Synthesized Design

**Achieved Frequency**

- **rect 2d**
- **rect 3d**
- **triangular 2d**
- **triangular 3d**

Graph showing achieved frequencies for different designs with categories SLP, NLP, and FSM-LA2.
Conclusion

- Improved FSM generation from for loops
  - Example of HLS specific transformation
  - State look ahead to pipeline control-path
  - HLS tools currently lack compiler optimizations

- Applied to Nested Loop Pipelining
  - Enlarge applicability by reducing its overhead

- Future Directions
  - Other uses of next function
  - Other HLS-specific transformations