High-Performance Embedded Systems-on-a-Chip

Sanjay Rajopadhye
Computer Science, Colorado State University

Lecture 4: Systolic Synthesis (contd.)
Systolic Synthesis

- Project Administrivia
- Convolution example (concl.)
- Loop-level vs Instruction level parallelism
Recap Schedule

- Sept 4: Decide projects
- Sept 11: Proposal
- Sept 27: Related Work
- Nov 1: Status Report
- Dec 7: Final Report
- Exams Week: Presentations
Recap Schedule

- Sept 4: Decide projects
- Sept 11: Proposal
- Sept 27: Related Work
- Nov 1: Status Report
- Dec 7: Final Report
- Exams Week: Presentations

DISCUSS FREQUENTLY WITH ME
Convolution

Initial specification:

\[ y_i = \sum_{j=0}^{n-1} w_j \ast x_{i-j} \]
Serialization & Alignment

Replace (unbounded fan-in) \( \sum \) by sequence of binary additions. Align input and output vars.

\[ y_i = Y[i, 0] \]

\[ Y[i, j] = \begin{cases} 
  j = n - 1 & : w_j \times x_{i-j} \\
  j < n - 1 & : Y[i, j+1] + w_j \times x_{i-j} 
\end{cases} \]
Localization/Uniformization

Remove unbounded fan-out (i.e., “long”) dependences.

\[
y_i = Y[i, 0]
\]

\[
Y[i, j] = \begin{cases} 
  j = n - 1 & : W[i, j] * X[i, j] \\
  j < n - 1 & : Y[i, j + 1] + W[i, j] * X[i, j]
\end{cases}
\]

\[
X[i, j] = \begin{cases} 
  j = 0 & : x_i \\
  j > 0 & : X[i - 1, j - 1]
\end{cases}
\]

\[
W[i, j] = \begin{cases} 
  i = 0 & : w_j \\
  i > 0 & : W[i - 1, j]
\end{cases}
\]
Scheduling & Allocation
Scheduling & Allocation

\[ t(i, j) = 2i - j + n - 1 \]
Scheduling & Allocation

\[ t(i, j) = 2i - j + n - 1 \]

\[ a(i, j) = j \]
CoB Transformation

\[ T = (i, j \rightarrow 2i - j + n - 1, j) \]

\[
\begin{pmatrix}
  t \\
p
\end{pmatrix}
= T \begin{pmatrix}
  i \\
j
\end{pmatrix}
= \begin{pmatrix}
  2i - j + n - 1 \\
j
\end{pmatrix}
\]

\[
= \begin{bmatrix}
  2 & -1 \\
0 & 1
\end{bmatrix}
\begin{pmatrix}
i \\
j
\end{pmatrix}
+ \begin{bmatrix}
n - 1 \\
0
\end{bmatrix}
\]
CoB Transformation

\[ T = (i, j \rightarrow 2i - j + n - 1, j) \]

\[
\begin{pmatrix}
  t \\
  p
\end{pmatrix}
= T
\begin{pmatrix}
  i \\
  j
\end{pmatrix}
= \begin{pmatrix}
  2i - j + n - 1 \\
  j
\end{pmatrix}
\]

\[
= \begin{bmatrix}
  2 & -1 \\
  0 & 1
\end{bmatrix}
\begin{pmatrix}
  i \\
  j
\end{pmatrix}
+ \begin{bmatrix}
  n - 1 \\
  0
\end{bmatrix}
\]

\[
\begin{pmatrix}
  t \\
  p \\
  n
\end{pmatrix}
= \begin{bmatrix}
  2 & -1 & 1 \\
  0 & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix}
\begin{pmatrix}
  i \\
  j \\
  n
\end{pmatrix}
+ \begin{bmatrix}
  -1 \\
  0 \\
  1
\end{bmatrix}
\]
Graph of Transformed SURE
Transformed SURE

\[ Y[i, j] = \begin{cases} 
    j = n - 1 & : W[i, j] \times X[i, j] \\
    j < n - 1 & : Y[i - 1, j + 1] + W[i, j] \times X[i, j]
\end{cases} \]

\[ X[i, j] = \begin{cases} 
    j = 0 & : \ldots \\
    j > 0 & : X[i - 1, j - 1]
\end{cases} \]

\[ W[i, j] = \begin{cases} 
    i = n - 1 - j & : \ldots \\
    i > n - 1 - j & : W[i - 2, j]
\end{cases} \]
Gory Details

\[ y_i = Y[2i, 0] \]

\[ Y[t, p] = \begin{cases} 
  p = n - 1; \text{even}(t) & : W[t, p] * X[t, p] \\
  p < n - 1; & \\
  \text{even}(t + p - n + 1) & : W[t - 1, p] + W[t, p] * X[t, p] 
\end{cases} \]

\[ X[t, p] = \begin{cases} 
  p = 0; \text{even}(t - n + 1) & : x_{\frac{t-n+1}{2}} \\
  p > 0 & : X[t - 1, p - 1] 
\end{cases} \]

\[ W[t, p] = \begin{cases} 
  t = n - 1 - p & : w_p \\
  t > n - 1 - p; & \\
  \text{even}(t + p - n + 1) & : W[t - 2, p] 
\end{cases} \]
Loop vs Instruction Level Parallelism

Custom Computing Machines (CCM)
Loop vs Instruction Level Parallelism

Custom Computing Machines (CCM)

CCMs vs ASICs

- 3 to 10 times slower
- 20 to 40 times area penalty
- 10 times power consumption
Loop vs Instruction Level Parallelism

Custom Computing Machines (CCM)

CCMs vs ASICs

- 3 to 10 times slower
- 20 to 40 times area penalty
- 10 times power consumption
- NRE costs significantly smaller
- Reusable (reprogrammable)
CCMs vs microprocessors

- Performance in some cases (regular, non floating-point): 5 to 100 times faster
- 1 to 2 orders of magnitude lower power consumption
CCMs vs microprocessors

- Performance in some cases (regular, non floating-point): 5 to 100 times faster
- 1 to 2 orders of magnitude lower power consumption
- **Significantly harder** to program
A Niche for Custom Computing

A: Microprocessors
B: ASICs
C: CCMs

Diagram:
- A: Microprocessors
- B: ASICs
- C: CCMs

Axes:
- Performance
- Programmability
A Niche for Custom Computing

A: Microprocessors
B: ASICs
C: CCMs, the ideal
A Niche for Custom Computing

A: Microprocessors
B: ASICs
C: CCMs, the ideal
D: CCMs, the reality :-(

Performance

Programmability
A Niche for Custom Computing

A: Microprocessors
B: ASICs
C: CCMs, the ideal
D: CCMs, the reality :-(

CCM sweet spot
PE = loop body
\text{PE} = \text{loop body} \implies \text{poor performance}
**PE = loop body \[\Rightarrow\] poor performance**

<table>
<thead>
<tr>
<th>Cell type</th>
<th>FPGA Area</th>
<th>FPGA Speed</th>
<th>CPU Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix mult (int16)</td>
<td>350</td>
<td>40</td>
<td>180</td>
</tr>
<tr>
<td>Matrix mult (float)</td>
<td>760</td>
<td>11</td>
<td>180</td>
</tr>
<tr>
<td>Seq comp (amino acid)</td>
<td>200</td>
<td>34</td>
<td>12</td>
</tr>
<tr>
<td>Seq comp (DNA)</td>
<td>57</td>
<td>70</td>
<td>12</td>
</tr>
</tbody>
</table>
Solution: pipeline the processor

- Gain can be significant
- Low area penalty expected
- Possibility of aggressive pipelining
Solution: pipeline the processor

- Gain can be significant
- Low area penalty expected
- Possibility of aggressive pipelining

How?
Solution: pipeline the processor

- Gain can be significant
- Low area penalty expected
- Possibility of aggressive pipelining

How?

- Post-process the PE array through
  - **Skewing**: Standard affine transformation
  - **Clustering/Serialization**: Nonlinear transformation
**Solution: pipeline the processor**

- Gain can be significant
- Low area penalty expected
- Possibility of aggressive pipelining

**How?**

- Post-process the PE array through
  - **Skewing**: Standard affine transformation
  - **Clustering/Serialization**: Nonlinear transformation
- Retime critical paths with newly introduced registers
Solution: pipeline the processor

- Gain can be significant
- Low area penalty expected
- Possibility of aggressive pipelining

How?

- Post-process the PE array through
  - **Skewing**: Standard affine transformation
  - **Clustering/Serialization**: Nonlinear transformation
- Retime critical paths with newly introduced registers
- Transform the I/O control and array interface