18-643 Lecture 12: Spiral: Domain-Specific HLS

James C. Hoe
Department of ECE
Carnegie Mellon University
Housekeeping

• Your goal today: see an example of really high-level synthesis (this lecture not on Midterm)

• Notices
  – Handout #4: lab 2, due noon, 10/6
  – Handout #5: lab 3, due noon, 10/20
  – 2.5 weeks to project proposal
  – 1.5 week to midterm

• Readings
Conflict btwn High-Level and Generality

high-level: tool knows better than you

HLS: tool decides what you can say and what you mean

RTL synthesis: general-purpose but special handling of structures like FSM, arith, etc.

place-and-route: works the same no matter what design
### Spiral DFTgen: how high can you go?

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
<th>range</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Problem specification</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>transform size</td>
<td>64</td>
<td>4–32768</td>
<td>Number of samples (?)</td>
</tr>
<tr>
<td>direction</td>
<td>forward</td>
<td></td>
<td>forward or inverse DFT (?)</td>
</tr>
<tr>
<td>data type</td>
<td>fixed point</td>
<td>4–32 bits</td>
<td>fixed point precision (?)</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td></td>
<td>scaling mode (?)</td>
</tr>
<tr>
<td></td>
<td>unscaled</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Parameters controlling implementation</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>architecture</td>
<td>fully streaming</td>
<td></td>
<td>iterative or fully streaming (?)</td>
</tr>
<tr>
<td>radix</td>
<td>2</td>
<td>2, 4, 8, 16, 32, 64</td>
<td>size of DFT basic block (?)</td>
</tr>
<tr>
<td>streaming width</td>
<td>2</td>
<td>2–64</td>
<td>number of complex words per cycle (?)</td>
</tr>
<tr>
<td>data ordering</td>
<td>natural in / natural out</td>
<td></td>
<td>natural or digit-reversed data order (?)</td>
</tr>
<tr>
<td>BRAM budget</td>
<td>1000</td>
<td></td>
<td>maximum # of BRAMs to utilize (-1 for no limit) (?)</td>
</tr>
<tr>
<td>Permutation method</td>
<td>JACM'09 [3] (patented)</td>
<td></td>
<td>Please click for more information</td>
</tr>
</tbody>
</table>

[Generate Verilog][Reset]  

http://www.spiral.net/hardware/dftgen.html
Design Space and Quality of Result

DFT 1024 (16 bit fixed point) on Xilinx Virtex-6 FPGA

throughput [billion samples per second]

[r=2, d=1] Full iterative reuse
[r=4, d=1]
[r=32, d=1]

[r=2, d=2] Partial iterative reuse
[r=2, d=5]
[r=2, d=10]
[r=4, d=5]
[r=8, d=3]
[r=16, d=2]
[r=32, d=2] No iterative reuse

49x slices
132x throughput

[Milder, et al., 2012]
SPIRAL Framework

DSP transform (user specified)

Algorithm Level
- Algorithm Generation
- Algorithm Optimization

Implementation Level
- Implementation
- Code Optimization

Evaluation Level
- Compilation
- Performance Evaluation

Search/Learning

I want a DFT of size 1024

SPIRAL automation starts here

where most tools begin automating the problem

Principle 1: Domain knowledge in the system
Principle 2: Optimization at a high level of abstraction
Very-High-Level Description
Linear Transforms

• Linear transform is a matrix-vector multiplication
  – computing by definition takes \( O(N^2) \) operations
  – the matrix has structure

• E.g. discrete Fourier transform: \( y = DFT_N \cdot x \)

\[
\begin{bmatrix}
  y_0 \\
  y_1 \\
  \vdots \\
  y_j \\
  \vdots \\
  y_{N-1}
\end{bmatrix} = \begin{bmatrix}
  k \rightarrow 0 \ldots N-1 \\
  i \leftarrow 0 \ldots N-1
\end{bmatrix} \begin{bmatrix}
  x_0 \\
  x_1 \\
  \vdots \\
  x_k \\
  \vdots \\
  x_{N-1}
\end{bmatrix} \begin{bmatrix}
  e^{-i2\pi jk/N}
\end{bmatrix}
\]
“Fast” Algorithms

- “Fast” algorithm factors the matrix into a sequence of structured, sparse matrices
  
  cheaper sparse multiplies ⇒ $O(N \log(N))$ operations

- E.g. Cooley-Tukey Factorization of $DFT_4$
  
  $$
  \begin{bmatrix}
  1 & 1 & 1 & 1 \\
  1 & i & -1 & -i \\
  1 & -1 & 1 & -1 \\
  1 & -i & -1 & i \\
  \end{bmatrix} =
  \begin{bmatrix}
  1 & 1 & 1 & 1 \\
  1 & -1 & 1 & -1 \\
  1 & -i & 1 & i \\
  1 & -i & 1 & i \\
  \end{bmatrix}
  $$

- Matrix formula representation

  $$
  DFT_4 = (DFT_2 \otimes I_2)D_2^4(I_2 \otimes DFT_2)L_2^4
  $$
Factorization Rules

E.g. Cooley-Tukey

\[
DFT_{n \cdot m} = (DFT_n \otimes I_m)D_{n \cdot m}^{n \cdot m}(I_n \otimes DFT_m)L_{n \cdot m}^{n \cdot m}
\]

- \(DFT_2\) is \[
\begin{bmatrix}
1 & 1 \\
1 & -1
\end{bmatrix}
\]
- \(D\) is a diagonal matrix of twiddle factors
- \(L\) is a stride permutation matrix
- \(A \otimes B = [a_{j,k}B]\) is the tensor (or kronecker) product

\[
e.g., \quad I_n \otimes B \Rightarrow \begin{bmatrix}
B & B & \cdot & \cdot & \cdot & 0 \\
B & 0 & \cdot & \cdot & \cdot & 0 \\
\cdot & \cdot & \cdot & \cdot & \cdot & 0 \\
\cdot & \cdot & \cdot & \cdot & \cdot & 0 \\
\cdot & \cdot & \cdot & \cdot & \cdot & 0 \\
\cdot & \cdot & \cdot & \cdot & \cdot & 0
\end{bmatrix}
\]

\[
A \otimes I_n \Rightarrow \begin{bmatrix}
a_{0,0} & 0 & a_{0,1} & 0 & \cdot & \cdot \\
0 & a_{0,0} & 0 & a_{0,1} & 0 & \cdot \\
a_{1,0} & 0 & a_{1,1} & 0 & \cdot & \cdot \\
a_{0,0} & 0 & a_{0,1} & 0 & \cdot & \cdot \\
0 & a_{1,0} & 0 & a_{1,1} & 0 & \cdot \\
0 & 0 & a_{1,1} & 0 & \cdot & \cdot 
\end{bmatrix}
\]
Fast Fourier Transform Algorithms

• Recursively factorize by Cooley-Tukey rule until only leaf cases remain (e.g. $DFT_r$ for radix-$r$)

$$DFT_8 = (DFT_2 \otimes I_4)D_2^8(I_2 \otimes DFT_4)L_2^8$$

$$= (DFT_2 \otimes I_4)D_2^8(I_2 \otimes ((DFT_2 \otimes I_2)D_2^4(I_2 \otimes DFT_2)L_2^4))L_2^8$$

• Exponential number of alternatives

• Each ruletree corresponds a different algorithm

• All cost $O(N \log(N))$
Describing a Design Space vs a Point

\[ DCT_2^{(II)} \rightarrow \text{diag} \left( 1,1 / \sqrt{2} \right) \cdot F_2 \]
\[ DCT_n^{(II)} \rightarrow P \cdot \left( DCT_{n/2}^{(II)} \oplus DCT_{n/2}^{(IV)} \right) \cdot (I_{n/2} \otimes F_2)^Q \]
\[ DCT_n^{(IV)} \rightarrow S \cdot DCT_n^{(II)} \cdot D \]
\[ DCT_n^{(IV)} \rightarrow M_1 \land M_r \]
\[ DFT_n \rightarrow B \cdot (DCT_{n/2}^{(I)} \oplus DST_{n/2}^{(I)}) \cdot C \]
\[ DFT_{nm} \rightarrow (DFT_n \otimes I_m) \cdot D \cdot (I_n \otimes DFT_m) \cdot P \]
\[ F_n(h) \rightarrow (I_{n/d} \otimes^k I_{d+k}) \cdot (I_{n/d} \otimes F_d(h)) \]
\[ F_n(h) \rightarrow \text{Circ} (\overline{h}) \cdot E \]
\[ DWT_n(W) \rightarrow (DWT_{n/2}(W) \oplus I_{n/2}) \cdot P \cdot (I_{n/2} \otimes_k W) \cdot E \]
\[ WHT_{2^n} \rightarrow \prod_{i=1}^{n} (I_{2^{m_i+K+n_i-1}} \otimes WHT_{2^{n_i}} \otimes I_{2^{n_i+1+K+n_i}}) \]

\[ \cdots \cdots \cdots \]

Done once per transform by an expert and the tool becomes the expert
Very-High-Level Synthesis
Formula to HW

- Given \( y = M \cdot x \) where \( M \) is:
  - \( M = A \cdot B \) apply \( B \), then \( A \)
  - \( M = I_n \otimes A \) apply \( A \), \( n \) times in parallel
  - \( M \) is a permutation permute \( x \)
  - \( M \) is a diagonal scale \( x \)

\[
y = (A \cdot B) \cdot x = A \cdot (B \cdot x)
\]

\[
y = (I_2 \otimes A) \cdot x
\]

\[
y = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \cdot x
\]

\[
y = \begin{bmatrix}
7 & 0 & 0 & 0 \\
0 & 8 & 0 & 0 \\
0 & 0 & 2 & 0 \\
0 & 0 & 0 & 4
\end{bmatrix} \cdot x
\]
DFT_8 Example

\[ DFT_8 = (DFT_2 \times I_4)D_2^8(I_2 \times ((DFT_2 \times I_2)D_2^4(I_2 \times DFT_2)L_2^4))L_2^8 \]

(formula is applied from right to left)
Pease DFT₈ Example

\[ \text{DFT}_8 = R_8 \cdot T_2(I_4 \otimes F_2)L_4^8 \cdot T_1(I_4 \otimes F_2)L_4^8 \cdot T_0(I_4 \otimes F_2)L_4^8 \]

stage 1  stage 2  stage 3
How about good HW?

• Formulas map naturally to combinational dataflow, but this is neither good nor realistic

What if I want DFT\(_{16K}\)?

• Sequential datapath to reuse available HW
  – identify repeated kernels
  – instantiate kernels under resource constraints
  – schedule computation to reuse instantiated kernels

Do this at formula level with math-level knowledge
Tensor as Streaming Pipeline

\[ \mathbf{I}_m \otimes \mathbf{A}_n \quad \mathbf{I}_m \ominus^{sr} \mathbf{A}_n \quad \mathbf{I}_{mn/w} \ominus^{sr} (\mathbf{I}_{w/n} \otimes \mathbf{A}_n) \]

- **fully parallel**
- **fully streamed**
- **partially streamed**

Like data-parallel loops we seen in regular HLS
Pease DFT$_8$

\[
\text{DFT}_8 = R_8 \cdot T_2(I_4 \otimes F_2)L_4^8 \cdot T_1(I_4 \otimes F_2)L_4^8 \cdot T_0(I_4 \otimes F_2)L_4^8
\]
Streaming Pease DFT$_8$

\[
\text{DFT}_8 = R_8 \cdot T_2(I_4 \otimes F_2)L_4^8 \cdot T_1(I_4 \otimes F_2)L_4^8 \cdot T_0(I_4 \otimes F_2)L_4^8
\]
Iterative Reuse

\[
\prod_{\ell=0}^{m-1} A_n
\]

no reuse

\[
\prod_{\ell=0}^{m-1} A_n
\]

iteratively
reuse

\[
\prod_{\ell=0}^{p-1} \left( \prod_{k=0}^{(m/p)-1} A_n \right)
\]

partially iterative reuse

Like data-dependent loops
we seen in regular HLS
**Iterative Pease DFT**

- **Fine-grained control over cost/latency tradeoff**

\[ w_{max} = N \]

\[ w = 2, 4, \ldots, N \]

\[ w_{min} = 2 \]

cost \( \propto w \); latency \( \propto \frac{1}{w} \)

\( 18-643-F17-L12-S22, \) James C. Hoe, CMU/ECE/CALCM, ©2017
## Rewrite Rules for Streaming and Reuse

<table>
<thead>
<tr>
<th>name</th>
<th>rule</th>
<th>condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>base-SR</td>
<td>$A_n \rightarrow A_n$</td>
<td></td>
</tr>
<tr>
<td>product-SR</td>
<td>$\frac{A_n \cdot B_n \cdots Z_n}{\text{stream}(w)} \rightarrow \frac{A_n \cdot B_n \cdots Z_n}{\text{stream}(w)}$</td>
<td></td>
</tr>
<tr>
<td>stream-IR</td>
<td>$\prod_{n}^{\text{ir}} A_n \rightarrow \prod_{n}^{\text{ir}} A_n$</td>
<td></td>
</tr>
<tr>
<td>stream1</td>
<td>$I_m \otimes A_k \rightarrow I_{m/k+w} \otimes (I_{w/k} \otimes A_k)$</td>
<td>$mk &gt; w$ and $k \leq w$</td>
</tr>
<tr>
<td>stream1-dep</td>
<td>$I_m \otimes_t A_k^k \rightarrow I_{m/k+w} \otimes (I_{w/k} \otimes_{t_1} A_k^k_{w/k+w+t_1})$</td>
<td>$mk &gt; w$ and $k \leq w$</td>
</tr>
<tr>
<td>stream2</td>
<td>$I_m \otimes A_k \rightarrow I_m \otimes A_k$</td>
<td>$k &gt; w$</td>
</tr>
<tr>
<td>stream2-dep</td>
<td>$I_m \otimes_t A_k^k \rightarrow I_m \otimes_t A_k^k$</td>
<td>$k &gt; w$</td>
</tr>
<tr>
<td>stream-diag</td>
<td>$D_n \rightarrow \text{StreamDiag}(D_n, w)$</td>
<td>$w \mid n$</td>
</tr>
<tr>
<td>stream-perm</td>
<td>$P_n \rightarrow \text{StreamPerm}(P_n, w)$</td>
<td>$w \mid n$</td>
</tr>
</tbody>
</table>
Applicability to other transforms

- DFT radix 2
  \[ R_{2^k} \prod_{i=0}^{k-1} \left[ T_i \left( I_{2^{k-1}} \otimes DFT_2 \right) L^{2^k}_{2^{k-1}} \right] \]

- DFT radix 2^r
  \[ R_{2^k} \prod_{i=0}^{k/r-1} \left[ T_i \left( I_{2^{k-r}} \otimes DFT_{2^r} \right) L^{2^k}_{2^{k-r}} \right] \]

- 2-D DFT_{nxn}
  \[ \prod_{i=0}^{1} \left[ L^{n^2}_{n} \left( I_n \otimes DFT_n \right) \right] \]

- WHT
  \[ \prod_{i=0}^{k/r-1} \left[ \left( I_{2^{k-r}} \otimes WHT_{2^r} \right) L^{2^k}_{2^{k-r}} \right] \]

- DCT (type II)
  \[ DP \prod_{i=0}^{k-1} \left[ A_{k-i} L^{2^k}_2 \right] L^{2^k}_{2^{k-1}} L^{2^k}_{2^{k-1}} P^H \]
Toward Very-High-Level IPs
## Is DFTgen Easy to Use?

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
<th>range</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Problem specification</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>transform size</td>
<td>64</td>
<td>4–32768</td>
<td>Number of samples (7)</td>
</tr>
<tr>
<td>direction</td>
<td>forward</td>
<td></td>
<td>forward or inverse DFT (2)</td>
</tr>
<tr>
<td>data type</td>
<td>fixed point</td>
<td></td>
<td>fixed or floating point (2)</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>4–32 bits</td>
<td>fixed point precision (2)</td>
</tr>
<tr>
<td></td>
<td>unscaled</td>
<td></td>
<td>scaling mode (2)</td>
</tr>
<tr>
<td><strong>Parameters controlling implementation</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>architecture</td>
<td>fully streaming</td>
<td></td>
<td>iterative or fully streaming (2)</td>
</tr>
<tr>
<td>radix</td>
<td>2</td>
<td>2, 4, 8, 16, 32, 64</td>
<td>size of DFT basic block (2)</td>
</tr>
<tr>
<td>streaming width</td>
<td>2</td>
<td>2–64</td>
<td>number of complex words per cycle (2)</td>
</tr>
<tr>
<td>data ordering</td>
<td>natural in / natural out</td>
<td></td>
<td>natural or digit-reversed data order (7)</td>
</tr>
<tr>
<td>BRAM budget</td>
<td>1000</td>
<td></td>
<td>maximum # of BRAMs to utilize (-1 for no limit) (2)</td>
</tr>
<tr>
<td>Permutation method</td>
<td>JACM'09 [3] (patented)</td>
<td></td>
<td>Please click for more information</td>
</tr>
</tbody>
</table>

[http://www.spiral.net/hardware/dftgen.html](http://www.spiral.net/hardware/dftgen.html)
Easy to Use for Whom?

- Powerful? **Very!**
- Easy to use? **Not Really . . . .**
  - low-level cryptic domain-specific parameters
  - complexity of integrating, using, tuning and validating an instantiated IP within an enclosing context
- If you went to DFTgen right now
  - which configuration would you ask for first?
  - if not good enough, how to get a better one . . . .
  - do you know what good enough is . . . .
Different Kinds of Experts

**IP Users**
- **Application Developers**
  Assemble, configure and integrate multiple IPs to build larger chips

**IP Authors**
- **Domain Experts**
  Know the underlying algorithms and theory specific to the domain
- **Hardware Experts**
  Can build HW based on a set of specs or SW implementation
Make generator the IP

- Why limit to structural view of design
- Why not offer also . . . .
  - pre-knowledge about outcome & tradeoff of parameter combinations
  - IP-specific “meaningful” parameterizations, that is, ask how fast? instead of how many?
  - performance self-monitor, interface protocol checker
  - any X where IP authors can do better than IP users

**Shift burdens from IP users to IP authors**

⇒ make knowledge and expertise reusable
Parting Thoughts

• Encapsulating domain knowledge in a domain specific tool for truly high-level design automation

• Why is Spiral-DSP so good?
  Ans: it only does linear DSP transforms (fortunately FFT is pretty important)
  – very well understood mathematics
  – highly structured, highly regular computation
  – enumerable design space

Underlying approach/framework is generalizable!!