BeiHang Short Course, Part 4: Spiral: Domain-Specific Synthesis

James C. Hoe
Department of ECE
Carnegie Mellon University

Collaborators: Peter Milder (SUNY Stonybrook), Markus Pueschel (ETH), Franz Franchetti

Conflict btw High-Level vs Generality

- high-level: tools know better than you
- RTL synthesis: general-purpose but special handling of structures like FSM, arith, etc.
- place-and-route: works the same no matter what design


J. C. Hoe, CMU/ECE/CALCM, © 2014, BHSC-L4-s1

J. C. Hoe, CMU/ECE/CALCM, © 2014, BHSC-L4-s2
The SPIRAL Project

- High performance implementations of linear DSP transforms (DFT, DCT, DWT, filters, etc.) are an important class of design problems
- Hand design and tuning is tricky and expensive
  - needs both math and implementation knowledge
  - time-consuming and tedious
  - needs to repeat effort for every new context
- SPIRAL research goal: A flexible push-button design generator that produces SW & HW implementations comparable with expert hand design

Why we can do better than hand design

- SPIRAL is only focused on linear DSP transforms
- These transforms are highly structured, highly regular and very well understood mathematically
- Algorithmic implementations of a transform can be enumerated following a known set of rules
- For a given objective function and mapping target, a computer generates a solution at least as good as the best human effort—by trying enough implementations
I want a DFT of size 1024 on a \{Xilinx, P4, Cell\...\}.

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**

---

**www.spiral.net/hardware/dftgen.html**

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
<th>range</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem specification</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>transform size</td>
<td>64</td>
<td>4–32760</td>
<td>Number of samples [2]</td>
</tr>
<tr>
<td>direction</td>
<td>forward</td>
<td>forward or inverse DFT [7]</td>
<td></td>
</tr>
<tr>
<td>data type</td>
<td>fixed point</td>
<td>fixed point precision [2]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10 bits</td>
<td>scaling mode [2]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>unscaled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameters controlling implementation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>architecture</td>
<td>fully streaming</td>
<td>iterative or fully streaming [3]</td>
<td></td>
</tr>
<tr>
<td>index</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 [3]</td>
</tr>
<tr>
<td>data ordering</td>
<td>natural/normal-end</td>
<td>natural or digit-reversed data order [7]</td>
<td></td>
</tr>
<tr>
<td>DRAM budget</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>permutation method</td>
<td>JACOBI [5]</td>
<td>maximum # of BRAMs to utilize (1 for no limit) [3]</td>
<td></td>
</tr>
</tbody>
</table>

Please click for more information.
Outline

• SPIRAL Formula Framework

• SPIRAL for HW FFT cores

• SPIRAL for HW FFT “un”-core

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: $y = DFT_N \cdot x$

\[
\begin{bmatrix}
  y_0 \\
  y_1 \\
  \vdots \\
  y_j \\
  \vdots \\
  y_{N-1}
\end{bmatrix} =
\begin{bmatrix}
  e^{-i2\pi jk/N} & \cdots & e^{-i2\pi (N-1)k/N}
\end{bmatrix}
\begin{bmatrix}
  x_0 \\
  x_1 \\
  \vdots \\
  x_j \\
  \vdots \\
  x_{N-1}
\end{bmatrix}
\]
“Fast” Algorithms

- A “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 DFT₄

\[
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & i & -1 & -i \\
1 & -1 & 1 & -1 \\
1 & -i & -1 & i
\end{bmatrix} = \begin{bmatrix}
1 & 1 & . & . \\
. & 1 & 1 & . \\
1 & . & 1 & -1 \\
. & 1 & . & 1
\end{bmatrix} \begin{bmatrix}
1 & 1 & . & . \\
. & 1 & 1 & . \\
1 & . & 1 & -1 \\
. & 1 & . & 1
\end{bmatrix}
\]

- Matrix formula representation

\[
DFT₄ = (DFT₂ \otimes I₂)D₂⁴(I₂ \otimes DFT₂)L₂⁴
\]

Factorization Rules

E.g. Cooley-Tukey

\[
DFT_{n,m} = (DFTₙ \otimes Iₘ)Dₙ^{m,m}(Iₙ \otimes DFTₘ)Lₙ^{n,m}
\]

- \(DFT₂\) 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

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

\[
A \otimes Iₙ \Rightarrow \begin{bmatrix}
& & & \ldots & \ldots \\
& & & \ldots & \ldots \\
\end{bmatrix}
\]
“Fast” Fourier Transform Algorithms

- Recursively factorize by the 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 rule tree corresponds a different algorithm

All cost \(O(N \log(N))\)

A System of Transforms and Rules

\[
\begin{align*}
DCT_2^{(II)} & \rightarrow \text{diag} \left( 1, 1/\sqrt{2} \right) \cdot F_2 \\
DCT_n^{(II)} & \rightarrow P \cdot (DCT_n^{(II)} \otimes DCT_{n/2}^{(IV)}) \cdot (I_{n/2} \otimes F_2)^0 \\
DCT_n^{(IV)} & \rightarrow S \cdot DCT_n^{(II)} \cdot D \\
DCT_n^{(IV)} & \rightarrow M_1 \cdots M_r \\
DFT_n & \rightarrow B \cdot (DCT_n^{(I)} \otimes DST_n^{(I)}) \cdot C \\
DFT_{nm} & \rightarrow (DFT_n \otimes I_m) \cdot D \cdot (I_n \otimes DFT_m) \cdot P \\
F_n(h) & \rightarrow (I_{n/d} \otimes I_{d+k}) \cdot (I_{n/d} \otimes F_d(h)) \\
F_n(h) & \rightarrow \text{Circ} \left( \frac{h}{n} \right) \cdot E \\
DWT_n(W) & \rightarrow (\text{DWT}_{n/2}(W) \otimes I_{n/2}) \cdot P \cdot (I_{n/2} \otimes W) \cdot E \\
WHT_n & \rightarrow \prod_{i=1}^n (I_{2n^{i-1} \cdots n-1} \otimes \text{WHT}_{2^{i}} \otimes I_{2n^{i-1} \cdots n})
\end{align*}
\]

50+ transforms

150+ rules
Algorithmic Design Space

<table>
<thead>
<tr>
<th>size</th>
<th># of DFT</th>
<th># of DCT-IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>126</td>
</tr>
<tr>
<td>16</td>
<td>296</td>
<td>31242</td>
</tr>
<tr>
<td>32</td>
<td>27744</td>
<td>1924443362</td>
</tr>
<tr>
<td>64</td>
<td>~1.01 × 10^{27}</td>
<td>7343815121631354242</td>
</tr>
<tr>
<td>128</td>
<td>~2.31 × 10^{61}</td>
<td>~1.07 × 10^{38}</td>
</tr>
<tr>
<td>256</td>
<td>~2.86 × 10^{133}</td>
<td>~2.30 × 10^{76}</td>
</tr>
<tr>
<td>512</td>
<td>~2.86 × 10^{133}</td>
<td>~1.06 × 10^{153}</td>
</tr>
</tbody>
</table>

Different characteristics: data flow, numerical stability, operation ordering, working set size, datapath regularity

Design Space: SW DCT 32 on P4

Histogram of 10,000 randomly selected algorithms

- Histogram by runtime (P4, 3.2 GHz)
- Histogram by num. accuracy

J. C. Hoe, CMU/ECE/CALCM, © 2014, BHSC-L4-s13
Outline

- SPIRAL Formula Framework
- SPIRAL for HW FFT cores
- SPIRAL for HW FFT “un”-core

Formula to HW (Combinational)

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

\[
\begin{align*}
  y &= (A \cdot B) \cdot x \\
  y &= (I_2 \otimes A) \cdot x
\end{align*}
\]
**DFT<sub>8</sub> Datapath Example**

\[ DFT_8 = (DFT_2 \otimes I_4) D^8_2 (I_2 \otimes (DFT_2 \otimes I_2)) D^4_2 (I_2 \otimes DFT_2) L^4_2 L^8_4 \]

(formula is applied from right to left)

**Pease DFT<sub>8</sub> Example**

\[ DFT_8 = R_8 \cdot T_2(I_4 \otimes F_2) L^8_4 \cdot T_1(I_4 \otimes F_2) L^8_4 \cdot T_0(I_4 \otimes F_2) L^8_4 \]
How about good HW?

- Matrix formulas have a natural mapping to dataflow and hence combinational datapath
- However, real hardware designs must fit a given resource constraint
  \[ \Rightarrow \text{sequential datapath that reuse available HW} \]
  - identify repeated kernels
  - instantiate kernels under resource constraints
  - schedule computation to reuse instantiated kernels

We want to do the analysis and mapping at formula level, with high-level algorithm knowledge

Tensor as Streaming Parallelism

\[
I_m \otimes A_n \quad I_m \otimes^{sr} A_n \quad I_{mn/w} \otimes^{sr} \left( I_{w/n} \otimes A_n \right)
\]

- fully parallel
- fully streamed
- partially streamed
Pease DFT Example: DFT$_8$

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

Pease DFT Example: DFT$_8$ Streaming

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

- Simple regular structure embodied in Pease FFT

\[
\text{DFT}_{2^k} = R_{2^k} \left( \prod_{i=0}^{k-1} T_i (I_{2^{k-1}} \otimes F_2) L_{2^{k-1}}^{2^k} \right)
\]

- Example:

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

Formally representing horizontal reuse

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

- not horizontally reused
- horizontally reused
- partially horizontally reused

J. C. Hoe, CMU/ECE/CALCM, © 2014, BHSIC-14-424
Iterative Reuse of Logic

\[ p_{\text{max}} = \frac{n}{2} \]

\[ p = 1, 2, 4, \ldots, \frac{n}{2} \]

\[ p_{\text{min}} = 1 \]

Cost \( \propto p \)
Latency \( \propto \frac{1}{p} \)

Fine-grained control over cost/latency tradeoff

---

### Example:
Rewriting rules for streaming reuse

<table>
<thead>
<tr>
<th>Name</th>
<th>Rule</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>base-IR</td>
<td>( A_0 \leftarrow \text{stream}(w) )</td>
<td></td>
</tr>
<tr>
<td>product-IR</td>
<td>( A_0 \cdot A_1 \cdots A_n \leftarrow \text{stream}(w) \cdot \text{stream}(w) \cdots \text{stream}(w) )</td>
<td></td>
</tr>
<tr>
<td>stream-IR</td>
<td>( \prod A_i \leftarrow \text{stream}(w) )</td>
<td></td>
</tr>
<tr>
<td>stream1</td>
<td>( I_m \leftarrow A_k \leftarrow I_m \leftarrow A_k \leftarrow I_m \leftarrow A_k \leftarrow I_m )</td>
<td>( m &gt; w ) and ( k \leq w )</td>
</tr>
<tr>
<td>stream1-dep</td>
<td>( I_m \leftarrow A_k \leftarrow I_m \leftarrow A_k \leftarrow I_m \leftarrow A_k \leftarrow I_m )</td>
<td>( m &gt; w ) and ( k \leq w )</td>
</tr>
<tr>
<td>stream2</td>
<td>( I_m \leftarrow A_k \leftarrow I_m \leftarrow A_k \leftarrow I_m \leftarrow A_k \leftarrow I_m )</td>
<td>( k &gt; w )</td>
</tr>
<tr>
<td>stream2-dep</td>
<td>( I_m \leftarrow A_k \leftarrow I_m \leftarrow A_k \leftarrow I_m \leftarrow A_k \leftarrow I_m )</td>
<td>( k &gt; w )</td>
</tr>
<tr>
<td>stream-dist</td>
<td>( D_n \leftarrow \text{StreamDist}(D_n, w) )</td>
<td>( w \mid n )</td>
</tr>
<tr>
<td>stream-pen</td>
<td>( F_n \leftarrow \text{StreamPen}(F_n, w) )</td>
<td>( w \mid n )</td>
</tr>
</tbody>
</table>
Applicability to other transforms?

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

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

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

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

- DCT (type II)
  \[ DP \prod_{i=0}^{k-1} \left[ A_{k-i} L_{2^i}^2 \right] L_{2^k}^2 \cdot P_H^H \]

---

FPGA: Area vs. Throughput

![Graph showing Pareto optimal and 49x slices, 132x throughput](image)

---

J. C. Hoe, CMU/ECE/CALCM, © 2014, BHSC-14-427
Outline

- SPIRAL Formula Framework
- SPIRAL for HW FFT cores
- SPIRAL for HW FFT “un”-core

A 2D-FFT Algorithm

- Row-column algorithm:

\[
2D\text{-DFT}_{n \times n} = \underbrace{\text{DFT}_{n} \otimes I_{n}}_{\text{Column Stage}} \underbrace{I_{n} \otimes \text{DFT}_{n}}_{\text{Row Stage}}
\]

Dataset: (Logical abstraction of the 2D dataset)
Off-chip Data Sets

- Need to balance
  - kernel processing bandwidth
  - off-chip memory bandwidth
  - on-chip storage capacity

Inefficient DRAM Access Patterns

- Row-wise traversal -> Sequential accesses
- Column-wise traversal -> Large strided accesses

Row-major 2D array
How to Optimize the Access Patterns

row-major “blocked”

in row-buffer sized chunks

[Akin, et al., FCCM 2012]

Design Generator w/ Tensor Formalism

2D-DFT_{n \times n} = \left( \text{DFT}_n \otimes I_n \right) \left( I_n \otimes \text{DFT}_n \right)

\text{row-column algorithm}

\text{symmetric algorithm}

\text{symmetric algorithm with tiling}

write tiles column-wise
transpose and re-tile on-chip
FFT processing
linearize on-chip
read tiles row-wise

[Akin, et al., FCCM 2012]
2D-FFT (double) Raw Performance

2D-FFT (double) BW Efficiency
Recap the Last Hour

- Encapsulating domain knowledge in a domain specific tool for **truly** high-level design automation
- SPIRAL
  - mathematical approach to DSP transform implementation (cores and “un”-core)
  - generalizable to other linear DSP transforms
  - as good as best expert designer
An Aside on Permutations

- Permutation: a fixed reordering of a given number of data points
- Example: $P_{16}$

Efficient streaming permutation is the secret to efficient high-performance FFT in hardware

Permuting Streaming Data

- Streaming data vector
- 4 words per cycle, 4 cycles
- Streaming permutation: data reordered in space & time

Example: $P_{16}$ with streaming width 4

Requirements:
- scale with permutation size $n$
- scale with streaming width $w$
- support full throughput: $w$ words per cycle
Naive Implementation

- Perform streaming permutation with one many-ported RAM

\[
\begin{array}{c}
\text{RAM} \\
\text{w read ports} \\
\text{w write ports} \\
\end{array}
\]

- 2w-ported RAM is impractical in practice for \( w > 1 \)

Solve the above using \( w \), 2-ported RAMs instead?

Using only 2-ported RAMs

- For 2-power vector and 2-power streaming width
  - solution for “bit” permutations requires 1x storage in \( w \) 2-ported RAMs
  - solution for arbitrary permutations requires 2x storage in 2w 2-ported RAMs

See [Pueschel, JACM’2009][ Milder, DATE’2009]