SPIRAL DSP Transform Compiler: Application Specific Hardware Synthesis

Peter A. Milder\(^1\) (peter.milder@stonybrook.edu)
Franz Franchetti, James C. Hoe, and Markus Pueschel\(^2\)

Department of ECE
Carnegie Mellon University

now with \(^1\)SUNY Stonybrook and \(^2\)ETH
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
SPIRAL Framework

Algorithm Level

Algorithm Generation
Algorithm Optimization

Implementation Level

Implementation
Code Optimization

Evaluation Level

Compilation
Performance Evaluation

DSP transform (user specified)
algorithm in SPL language
C/Fortran implementation
optimalized/adapted implementation

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
### Problem specification

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
<th>range</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>transform size</td>
<td>04</td>
<td>1-32768</td>
<td>Number of samples (2)</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>
</tbody>
</table>

### Parameters controlling implementation

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
<th>range</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<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 (2)</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>

[Generate Verilog] [Reset]
High-Level, Quality, and Specialization

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
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:
  \[
  y = DFT_N \cdot x
  \]
  \[
  \begin{bmatrix}
  y_0 \\
  y_1 \\
  \vdots \\
  y_j \\
  \vdots \\
  y_{N-1}
  \end{bmatrix} =
  \begin{bmatrix}
  x_0 \\
  x_1 \\
  \vdots \\
  x_k \\
  \vdots \\
  x_{N-1}
  \end{bmatrix} \cdot
  e^{-i2\pi jk/N}
  \]
  \[
  \begin{bmatrix}
  1 & \cdots & 0 \\
  0 & \cdots & 1
  \end{bmatrix}
  \]
  \[
  k \rightarrow 0 \ldots N-1
  \]
“Fast” Algorithms

- A “fast” algorithm factors the matrix into a sequence of structured, sparse matrices

  - Cheaper sparse multiplies \( \Rightarrow O(N \log(N)) \) operations

- E.g. Cooley-Tukey Factorization of \( \text{DFT}_4 \)

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

- Matrix formula representation

\[
\text{DFT}_4 = (\text{DFT}_2 \otimes I_2)D_2^4(I_2 \otimes \text{DFT}_2)L_2^4
\]
Factorization Rules

E.g. Cooley-Tukey

\[ \text{DFT}_{n \cdot m} = (\text{DFT}_n \otimes I_m) \text{DFT}^{n \cdot m}_n (I_n \otimes \text{DFT}_m) L^{n \cdot m}_n \]

- \(\text{DFT}_2\) 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

\[ \text{e.g., } I_n \otimes B \Rightarrow \begin{bmatrix} B & B & 0 \\ 0 & \cdots & B \end{bmatrix} \quad A \otimes I_n \Rightarrow \begin{bmatrix} a_{0,0} & 0 & a_{0,1} & 0 & \cdots \\ 0 & a_{0,0} & 0 & a_{0,1} & \cdots \\ a_{1,0} & 0 & a_{1,1} & 0 & \cdots \\ 0 & a_{1,0} & 0 & a_{1,1} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \]
“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 ruletree corresponds a different algorithm

- All cost $O(N \log(N))$
A System of Transforms and Rules

\[ DCT_{2}^{(II)} \rightarrow \text{diag} \left(1, 1 / \sqrt{2} \right) \cdot F_{2} \]
\[ DCT_{n}^{(II)} \rightarrow P \cdot \left( DCT_{n/2}^{(II)} \oplus DCT_{n/2}^{(IV)} \right) \cdot \left( I_{n/2} \otimes F_{2} \right) \circ \]
\[ DCT_{n}^{(IV)} \rightarrow S \cdot DCT_{n}^{(II)} \cdot D \]
\[ DCT_{n}^{(IV)} \rightarrow M_{1} \cdots M_{r} \]
\[ DFT_{n} \rightarrow B \cdot \left( DCT_{n/2}^{(I)} \oplus DST_{n/2}^{(I)} \right) \cdot C \]
\[ DFT_{nm} \rightarrow \left( DFT_{n} \otimes I_{m} \right) \cdot D \cdot \left( I_{n} \otimes DFT_{m} \right) \cdot P \]
\[ F_{n}(h) \rightarrow \left( I_{n/d} \otimes^{k} I_{d+k} \right) \cdot \left( I_{n/d} \otimes F_{d}(h) \right) \]
\[ F_{n}(h) \rightarrow \text{Circ}(\bar{h}) \cdot E \]
\[ DWT_{n}(W) \rightarrow \left( DWT_{n/2}(W) \oplus I_{n/2} \right) \cdot P \cdot \left( I_{n/2} \otimes_{k} W \right) \cdot E \]
\[ WHT_{2^{n}} \rightarrow \prod_{i=1}^{n} \left( I_{2^{m_{+\ldots+n_{i-1}}} \otimes WHT_{2^{n_{i}}} \otimes I_{2^{n_{i+1}+\ldots+n_{t}}}} \right) \]

\[ \cdots \cdots \cdots \cdots \cdots \]

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>162570361280</td>
<td>7343815121631354242</td>
</tr>
<tr>
<td>128</td>
<td>~1.01 × 10^{27}</td>
<td>~1.07 × 10^{38}</td>
</tr>
<tr>
<td>256</td>
<td>~2.31 × 10^{61}</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
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 \( x \)

\[
\begin{align*}
y &= (A \cdot B) \cdot x \\
y &= (I_2 \otimes A) \cdot x
\end{align*}
\]
$DFT_8 = (DFT_2 \otimes I_4) D_2^8 (I_2 \otimes (DFT_2 \otimes I_2) D_2^4 (I_2 \otimes DFT_2) L_2^4)) L_2^8$

(formula is applied from right to left)
Pease DFT$_8$ Example

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

⇒ *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

\[ I_m \otimes A_n \]

\[ I_m \otimes^{sr} A_n \]

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

- Fully parallel
- Fully streamed
- Partially streamed
Pease DFT Example: $\text{DFT}_8$

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

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

• 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

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

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

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

not horizontally reused

horizontally reused

partially horizontally reused
Iterative Reuse of Logic

Fine-grained control over cost/latency tradeoff

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

\[ p = 1, 2, 4, \ldots n/2 \]

\[ \text{cost} \propto p \quad \text{latency} \propto \frac{1}{p} \]
Example: rewriting rules for streaming 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></td>
<td>(\text{stream}(n))</td>
<td></td>
</tr>
<tr>
<td>product-SR</td>
<td>$A_n \cdot B_n \cdots Z_n \rightarrow A_n \cdot B_n \cdots Z_n$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{stream}(w))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{stream}(w))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{stream}(w))</td>
<td></td>
</tr>
<tr>
<td>stream-IR</td>
<td>$\prod^{\text{ir}} A_n \rightarrow \prod^{\text{ir}} A_n$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(\text{stream}(w))</td>
<td></td>
</tr>
<tr>
<td>stream1</td>
<td>$I_m \otimes A_k \rightarrow I_{mk/w} \otimes^{sr} (I_{w/k} \otimes A_k)$</td>
<td>(mk &gt; w) and (k \leq w)</td>
</tr>
<tr>
<td></td>
<td>(\text{stream}(w))</td>
<td></td>
</tr>
<tr>
<td>stream1-dep</td>
<td>$I_m \otimes \ell A^k_{\ell} \rightarrow I_{mk/w} \otimes^{sr}<em>{\ell_0} (I</em>{w/k} \otimes_{\ell_1} A^k_{\ell_0\cdot w/k+\ell_1})$</td>
<td>(mk &gt; w) and (k \leq w)</td>
</tr>
<tr>
<td></td>
<td>(\text{stream}(w))</td>
<td></td>
</tr>
<tr>
<td>stream2</td>
<td>$I_m \otimes A_k \rightarrow I_m \otimes^{sr} A_k$</td>
<td>(k &gt; w)</td>
</tr>
<tr>
<td></td>
<td>(\text{stream}(w))</td>
<td></td>
</tr>
<tr>
<td>stream2-dep</td>
<td>$I_m \otimes \ell A_k \rightarrow I_m \otimes^{sr}<em>\ell A^k</em>{\ell}$</td>
<td>(k &gt; w)</td>
</tr>
<tr>
<td></td>
<td>(\text{stream}(w))</td>
<td></td>
</tr>
<tr>
<td>stream-diag</td>
<td>$D_n \rightarrow \text{StreamDiag}(D_n, w)$</td>
<td>(w</td>
</tr>
<tr>
<td></td>
<td>(\text{stream}(w))</td>
<td></td>
</tr>
<tr>
<td>stream-perm</td>
<td>$P_n \rightarrow \text{StreamPerm}(P_n, w)$</td>
<td>(w</td>
</tr>
</tbody>
</table>
Applicability to other transforms?

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

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

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

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

- DCT (type II)
  \[ DP \prod_{i=0}^{k-1} \left[ A_{k-i} L_{2^i}^2 \right] L_{2^{k-1}}^2 L_{2^{k-1}}^2 P_{2^k}^H \]
FPGA: Area vs. Throughput

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

throughput [billion samples per second]

Pareto optimal

49x slices

132x throughput

Spiral generated
Outline

• SPIRAL Formula Framework

• SPIRAL for HW FFT cores

• SPIRAL for HW FFT “un”-core
A 2D-FFT Algorithm

• Row-column algorithm:

\[ 2D\text{-}DFT_{n \times n} = (DFT_n \otimes I_n)(I_n \otimes DFT_n) \]

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

 DDR2-800 Bandwidth on DE4 (per channel)
Bandwidth [GB/s]

0 1 2 3 4 5 6 7

Packet Size [KB]

0 0.02 0.06 0.13 0.25 0.5 1 2 4 8 16 32

Read
Write
Row buffer size

CMU/ECE/Hoe, February 2013, slide-32
How to Optimize the Access Patterns

row-major “blocked”

in row-buffer sized chunks

[0, k^2]

linear mem space

n^2

n

n

k

k

in row-buffer sized chunks

[Akin, et al., FCCM 2012]
Design Generator w/ Tensor Formalism

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

\[
= \prod_{i=0}^{1} \left( L_n^{n^2} \left( I_n \otimes \text{DFT}_n \right) I_n^{2} \right)
\]

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

Problem Size [Akin, et al., FCCM 2012]

Performance [Gflop/s]

- GTX 480 (GPU)
- Core i7 960 (CPU)
- DE4 (FPGA)
2D-FFT (double) BW Efficiency

Bandwidth normalized performance \([\text{Gflop/s}/\text{GB/s}]\)

Problem Size

- DE4 (FPGA)
- GTX 480 (GPU)
- Core i7 960 (CPU)

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

![Graph showing Power normalized performance vs. Problem Size for DE4 (FPGA), GTX 480 (GPU), and Core i7 960 (CPU).](Akin, et al., FCCM 2012)
Conclusions

• Encapsulating domain knowledge in a domain specific tool for 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

• Thank you