SPIRAL: Tuning DSP Transforms to Computing Platforms

José M. F. Moura

Faculty
- Jeremy Johnson (Drexel)
- Robert Johnson (MathStar Inc.)
- David Padua (UIUC)
- Viktor Prasanna (USC)
- Markus Püschel (CMU)
- Manuela Veloso (CMU)

Students
- Franz Franchetti (TU Vienna)
- Gavin Haentjens (CMU)
- Pinit Kumhom (Drexel)
- Neungsoo Park (USC)
- David Sepiashvili (CMU)
- Bryan Singer (CMU)
- Yevgen Voronenko (Drexel)
- Jianxin Xiong (UIUC)

http://www.ece.cmu.edu/~spiral
Work supported by DARPA (DSO), Applied & Computational Mathematics Program, OPAL, through grant managed by research grant DABT63-98-1-0004 administered by the Army Directorate of Contracting.
Moore’s Law and High(est) Performance
Scientific Computing

(single processor, off-the-shelf)

Moore’s Law:
- processor-memory bottleneck
- short life cycles of computers
- very complex architectures
  - vendor specific
  - special instructions (MMX, SSE, FMA, …)
  - undocumented features

Effects on software/algorithms:
- arithmetic cost model not accurate for predicting runtime
  (one cache miss = 10 floating point ops)
- better performance models hard to get
- best code is machine dependent (registers/caches size, structure)
- hand-tuned code becomes obsolete as fast as it is written
- compiler limitations
- full performance requires (in part) assembly coding

Portable performance requires automation
Automatic Performance Tuning: Research

Linear Algebra:
- ATLAS (J. Dongarra et al.)
- LAPACK
- PhiPACK (J. Demmel et al.)

Signal Processing:
- FFTW (M. Frigo and S. Johnson)
- SPIRAL
SPIRAL

Automates

Implementation
- cuts development costs
- code less error-prone

Optimization
- systematic exploration of alternatives both at algorithmic and code level

Platform-Adaptation
- takes advantage of architecture specific features
- porting without loss of performance

of DSP algorithms
- are performance critical

A library generator for highly optimized signal processing algorithms
SPIRAL Approach

given → DSP Transform
(DFT, DCT, Wavelets etc.)

SPIRAL Search Space

given → Computing Platform
(Pentium III, Pentium 4, Athlon, SUN, PowerPC, Alpha, …)

Possible Algorithms

Possible Implementations

Performance Evaluation

Intelligent Search

adapted implementation
Organization

- Mathematical Framework
  - Transforms, Rules, and Formulas
- Formula Generator
  - Transform $\rightarrow$ Algorithm
- SPL and SPL Compiler
  - Algorithm $\rightarrow$ Implementation
- Search Engine
  - How to find the best implementation
- SPIRAL system
  - Everything taken together
- Conclusions
DSP Algorithms: Example 4-point DFT

Cooley/Tukey FFT (size 4):  
\[
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & i & -1 & -i \\
1 & -1 & 1 & -1 \\
1 & -i & -1 & i \\
\end{bmatrix} \cdot 
\begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & -1 & 0 \\
0 & 1 & 0 & -1 \\
\end{bmatrix} = 
\begin{bmatrix}
1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 \\
1 & -1 & 0 & 0 \\
0 & 0 & 1 & 1 \\
\end{bmatrix}
\]

Fourier transform  
Diagonal matrix (twiddles)

\[
DFT_4 = (DFT_2 \otimes I_2) \cdot T_2^4 \cdot (I_2 \otimes DFT_2) \cdot L_2^4
\]

- product of structured sparse matrices  
- mathematical notation
DSP Algorithms: Terminology

Transform

\[ DFT_n \] parameterized matrix

Rule

\[ DFT_{nm} \rightarrow (DFT_n \otimes I_m) \cdot D \cdot (I_n \otimes DFT_m) \cdot P \]

- a breakdown strategy
- product of sparse matrices

Ruletree

\[
\begin{align*}
DFT_8 & \quad \text{\textbullet recursive application of rules} \\
DFT_2 & \quad \text{\textbullet uniquely defines an algorithm} \\
DFT_4 & \quad \text{\textbullet efficient representation} \\
DFT_2 & \quad \text{\textbullet easy manipulation} \\
DFT_2 & \\
\end{align*}
\]

Formula

\[ DFT_8 = (F_2 \otimes I_4) \cdot D \cdot (I_2 \otimes (I_2 \otimes F_2 \cdots)) \cdot P \]

- few constructs and primitives
- uniquely defines an algorithm
- can be translated into code
More Cooley-Tukey Rules

- DFT is symmetric ⇒ transpose the rule:
  \[
  F_{RS} = L_S^{RS} \left( I_R \otimes F_S \right) T_S^{RS} \left( F_R \otimes I_S \right)
  \]
  CT rule transposed

- Commuting tensor product factors
  \[
  B \otimes A = L_n^{mn} \left( A \otimes B \right) L_m^{nn}
  \]
  A and B square size m and n

- Commutation property ⇒ further variations

  \[
  F_N = L_S^{RS} \left( I_S \otimes F_R \right) L_R^{RS} T_S^{RS} \left( I_R \otimes F_S \right) L_R^{RS}
  \]

  \[
  F_N = \left( F_R \otimes I_S \right) T_S^{RS} L_S^{RS} \left( F_S \otimes I_R \right)
  \]

  \[
  (F_2 \otimes I_2)x = \begin{bmatrix}
  1 & 0 & 1 & 0 \\
  0 & 1 & 0 & 1 \\
  1 & 0 & -1 & 0 \\
  0 & 1 & 0 & -1
\end{bmatrix} \begin{bmatrix}
  x_0 \\
  x_1 \\
  x_2 \\
  x_3
\end{bmatrix}
  \]

  \[
  (I_2 \otimes F_2) = \begin{bmatrix}
  1 & 1 & 0 & 0 \\
  1 & -1 & 0 & 0 \\
  0 & 0 & 1 & 1 \\
  0 & 0 & 1 & -1
\end{bmatrix} \begin{bmatrix}
  x_0 \\
  x_1 \\
  x_2 \\
  x_3
\end{bmatrix}
  \]

- Different patterns for access, storage and flow of data
Haar Wavelets – Example

- Haar wavelets = square waves
  \[ h = \frac{1}{\sqrt{2}} [1, 1], \quad h^1 = \frac{1}{\sqrt{2}} [1, -1] \]

- First stage: \( V_2 \Rightarrow V_1 \oplus W_1 \)

\[
H = \begin{bmatrix}
1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 \\
1 & -1 & 0 & 0 \\
0 & 0 & 1 & -1
\end{bmatrix}
= L_2^4 \begin{bmatrix}
1 & 1 & 0 & 0 \\
1 & -1 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 0 & 1 & -1
\end{bmatrix}
= L_2^4 (I_2 \otimes F_2) \quad \begin{bmatrix}
c_1 \\
d_1
\end{bmatrix} = H c_2
\]

- The process is repeated for the upper half of the output

\[
HT_{2^n} = \left( HT_{2^{n-1}} \oplus I_{2^{n-1}} \right) L_{2^{n-1}}^2 (I_{2^{n-1}} \otimes F_2), \quad HT_2 = F_2
\]
Discrete-Time Wavelet Transform

- Discrete-Time Wavelet Transform (DTWT) rule

\[
DTWT_{2^n} = (DTWT_{2^{n-1}} \oplus I_{2^{n-1}}) \underbrace{L_{2^{n-1}} (I_{2^{n-1}} \otimes_l W)}_H
\]

- Scaling (lowpass) and wavelet (highpass) filter coefficients

\[
W = \begin{bmatrix} h_0 & h_1 & \cdots & h_{l-1} \\ h'_0 & h'_1 & \cdots & h'_{l-1} \end{bmatrix}
\]

- DTWT - convolution rule

\[
H = ([1 \ 1] \otimes I_{m/2}) \cdot (C_{m/2}^T (lo) \oplus C_{m/2}^T (le)) \cdot L_2^n \oplus \\
([1 \ 1] \otimes I_{m/2}) \cdot (C_{m/2}^T (ho) \oplus C_{m/2}^T (he)) \cdot L_2^n \cdot \left( \begin{array}{c} 1 \\ 1 \end{array} \right) \otimes I_n
\]

- \(lo\)-lowpass odd coeffs., \(le\)-lowpass even coeffs.
- \(ho\)-highpass odd coeffs., \(he\)-highpass even coeffs.
DSP Transforms

- discrete Fourier transform
  \[ DFT_n = \left[ \exp \left( 2kli \pi / n \right) \right] \]

- Walsh-Hadamard transform
  \[ WHT_{2^k} = DFT_2 \otimes \cdots \otimes DFT_2 \]

- discrete cosine and sine Transforms (16 types)
  \[ DCT^{(II)}_n = \left[ \cos \left( k(l + 1/2)\pi / n \right) \right] \]
  \[ DCT^{(IV)}_n = \left[ \cos \left( (k + 1/2)(l + 1/2)\pi / n \right) \right] \]
  \[ DST^{(I)}_n = \left[ \sin \left( kl\pi / n \right) \right] \]
  \[ MDCT_{n \times 2^n} = \left[ \cos \left( (k + (n + 1)/2)(l + 1/2)\pi / n \right) \right] \]

- modified discrete cosine transform

- two-dimensional transform
  \[ T \otimes T \]

- discrete wavelet transform
  \[ DTWT_{2^n} = \left( DTWT_{2^{n-1}} \oplus I_{2^{n-1}} \right) L_{2^{n-1}}^{2^n} \left( I_{2^{n-1}} \otimes I_{l-2} \right) W \]

- Others: filtering, Haar, Hartley, ...
Rules = Breakdown Strategies

\[ DCT_{2}^{(II)} \rightarrow \text{diag} \left(1, 1 / \sqrt{2}\right) \cdot F_{2} \]

\[ DCT_{n}^{(III)} \rightarrow P \cdot (DCT_{n/2}^{(II)} \oplus DCT_{n/2}^{(IV)}) \cdot (I_{n/2} \otimes F_{2})^{Q} \]

\[ DCT_{n}^{(IV)} \rightarrow S \cdot DCT_{n}^{(II)} \cdot D \]

\[ DCT_{n}^{(IV)} \rightarrow M_{1} \cdots M_{r} \]

\[ DFT_{n} \rightarrow \text{CosDFT}_{n} + j \cdot \text{SinDFT}_{n} \]

\[ DFT_{n} \rightarrow B \cdot \left( DCT_{n/2}^{(I)} \oplus DST_{n/2}^{(I)} \right) \cdot C \]

\[ DFT_{nm} \rightarrow \left( DFT_{n} \otimes I_{m} \right) \cdot D \cdot \left( I_{n} \otimes DFT_{m} \right) \cdot P \]

\[ \text{CosDFT}_{n} \rightarrow \cdots \text{CosDFT}_{n/2} \cdots DCT_{n/4}^{(II)} \cdots \]

\[ \text{SinDFT}_{n} \rightarrow \cdots \text{SinDFT}_{n/2} \cdots DCT_{n/4}^{(II)} \cdots \]

\[ WHT_{2^{n}} \rightarrow \prod_{i=1}^{n} \left( I_{2^{n_{1}+\cdots+n_{i-1}}} \otimes WHT_{2^{n_{i}}} \otimes I_{2^{n_{i+1}+\cdots+n_{l}}} \right) \]

\[ \text{MDCT}_{n \times 2^{n}} \rightarrow S \cdot DCT_{n}^{(IV)} \cdot P \]

\[ \text{DTWT}_{2^{n}} = \left( \text{DTWT}_{2^{n-1}} \oplus I_{2^{n-1}} \right) \left( I_{2^{n-1}} \otimes I_{-2} \right) W \]

\[ H \]
Algorithms = Ruletrees = Formulas

\[ DCT_{8}^{(II)} \]

\[ DCT_{4}^{(II)} \]

\[ DCT_{2}^{(II)} \]

\[ R3 \]

\[ F_2 \]

\[ DCT_{4}^{(IV)} \]

\[ DCT_{2}^{(IV)} \]

\[ R6 \]

\[ F_2 \]

\[ DST_{2}^{(II)} \]

\[ R4 \]

\[ DCT_{2}^{(II)} \]

\[ R3 \]

\[ F_2 \]

\[ DST_{2}^{(II)} \]

\[ R4 \]

\[ F_2 \]

\[ DCT_{n}^{(II)} \rightarrow P \cdot (DCT_{n/2}^{(II)} \oplus DCT_{n/2}^{(IV)}) \cdot (F_2 \otimes I_{n/2}) \]

\[ DCT_{n}^{(IV)} \rightarrow P \cdot DCT_{n}^{(II)} \cdot S \]

\[ DCT_{2}^{(II)} \rightarrow \frac{1}{\sqrt{2}} \cdot F_2 \]
DTWT Ruletree – Example

DTWT \( n \)

\[ C(lo) \quad C_{m/2}(le) \quad C(ho) \quad C(he) \]

\[ m = n + l - 1 \]

Overlap-add rule

Circulant rule

\[ l_0 = le \] padded with \( b-1 \) zeroes

Circular convolution rule

\[ s = l/2 + b - 1 \]
Formula for a DCT, size 16

\[
\begin{align*}
(1_2 \otimes \text{DFT}_2)(^{[2,4,3,4]}_4) & \oplus \left( \text{diag}\left( \frac{1}{2 \cos(\frac{\pi}{16})}, \frac{1}{2 \cos(\frac{3\pi}{16})}, \frac{1}{2 \cos(\frac{5\pi}{16})}, \frac{1}{2 \cos(\frac{7\pi}{16})} \right) \cdot (1_2 \otimes \text{DFT}_2)(^{[2,4,3,4]}_4) \right)
\end{align*}
\]

\[
((\text{DFT}_2 \cdot \text{diag}(1, \sqrt{\frac{1}{2}})) \oplus (\left( [1, 2], [1, 2] \cdot R_{\frac{13}{8}\pi} \right))^{[1,2),2]} \cdot (2, 3, 4, 4) \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix})^{[1,4)(2,3),4]}.
\]

\[
(1_4 \otimes \text{DFT}_2)(^{[2,8,5,3]}_8) \oplus (\left( [2, 5, 4, 3, 7, 6, 8, 8] \cdot (1_2 \oplus (1_2 \oplus (\left( [1, 2], [1, 2] \cdot R_{\frac{13}{8}\pi} \right)) \right))
\]

\[
(1_2 \otimes \text{DFT}_2 \otimes 1_2) \cdot 1_4 \oplus (\left( [1, 2], [1, 2] \cdot R_{\frac{13}{8}\pi} \right) \oplus (\left( [1, 2], [1, 2] \cdot R_{\frac{1}{8}\pi} \right)) \cdot (1_1 \otimes \text{DFT}_2 \otimes 1_4) \cdot (\left( [1, 2], [1, 2] \cdot R_{\frac{49}{32}\pi} \right) \oplus (\left( [1, 2], [1, 2] \cdot R_{\frac{53}{32}\pi} \right) \oplus (\left( [1, 2], [1, 2] \cdot R_{\frac{61}{32}\pi} \right)) \cdot (2, 8) \cdot (4, 6, 8) \right) \cdot (1, 8) \cdot (2, 7) \cdot (3, 6) \cdot (4, 5) \cdot (8).)
\]

(1_8 \otimes \text{DFT}_2)(^{[2,16,9,5,3]}_8) \cdot (4, 15, 8, 13, 7) \cdot (6, 14, 10, 12, 11, 16)
Helpful Concept

DSP Transforms

- DSP transform (of size) ↔ Rule
- Formula/Algorithm ↔ Element in Language (only terminals)

Formal Languages

- Non-terminal symbol (with attribute) ↔ Rule (production)
- Element in Language (only terminals)
Mathematical Framework: Summary

- fast algorithms represented as ruletrees (easy generation/manipulation) and as formulas (can be translated into code)

- formulas built from few constructs and primitives

- many different algorithms/formulas generated from few rules (combinatorial explosion)

- these algorithms are (essentially) equal in arithmetic cost, but differ in data flow
Organization

- Mathematical Framework
  - Transforms, Rules, and Formulas
- Formula Generator
  - Transform $\rightarrow$ Algorithm
- SPL and SPL Compiler
  - Algorithm $\rightarrow$ Implementation
- Search Engine
  - How to find the best implementation
- SPIRAL system
  - Everything taken together
- Conclusions
Formula Generation

Formula Generator

- written in GAP/AREP (computer algebra system)
- all computation/manipulation is symbolic
- exact arithmetic
- easy extensible rule and transform data base
- verification of rules and formulas

data base (extensible!)  
data type

runtime  
formula translation (spl compiler)

export

cut here for other optimization problems
Organization

- Mathematical Framework
  Transforms, Rules, and Formulas
- Formula Generator
  Transform → Algorithm
- SPL and SPL Compiler
  Algorithm → Implementation
- Search Engine
  How to find the best implementation
- SPIRAL system
  Everything taken together
- Conclusions
Formulas in SPL

( compose
  ( diagonal ( 2*cos(1/16*pi) 2*cos(3/16*pi) 2*cos(5/16*pi) 2*cos(7/16*pi) )
  ( permutation ( 1 3 4 2 )
  ( tensor
    ( I 2 )
    ( F 2 )
  )
  ( permutation ( 1 4 2 3 )
  ( direct_sum
    ( compose
      ( F 2 )
      ( diagonal ( 1 sqrt(1/2) ) )
    )
    ( compose
      ( matrix
        ( 1 1 0 )
        ( 0 (-1) 1 )
      )
      ( diagonal ( cos(13/8*pi)-sin(13/8*pi) sin(13/8*pi) cos(13/8*pi)+sin(13/8*pi) )
      ( matrix
        ( 1 0 )
        ( 1 1 )
        ( 0 1 )
      )
    )
    ( permutation ( 2 1 )
  )

  )
)
SPL Syntax (Subset)

- **matrix operations:**
  - (compose formula formula ...)
  - (tensor formula formula ...)
  - (direct_sum formula formula ...)

- **direct matrix description:**
  - (matrix (a11 a12 ...) (a21 a22 ...) ...)
  - (diagonal (d1 d2 ...))
  - (permutation (p1 p2 ...))

- **parameterized matrices:**
  - (I n)
  - (F n)

- **scalars:**
  - 1.5, 2/7, cos(..), w(3), pi, 1.2e-04

- **definition of new symbols:**
  - (define name formula)
  - (template formula (i-code-list))

- **directives for code generation**
  - #codetype real/complex
  - #unroll on/off
SPL Compiler, 4-point FFT

(compose (tensor (F 2) (I 2)) (T 4 2)
(tensor (I 2) (F 2)) (L 4 2))

f0 = x(1) + x(3)
f1 = x(1) - x(3)
f2 = x(2) + x(4)
f3 = x(2) - x(4)
f4 = (0.00d0,-1.00d0)*f(3)
y(1) = f0 + f2
y(2) = f0 - f2
y(3) = f1 + f4
y(4) = f1 - f4

r0 = x(1) + x(5)
r1 = x(1) - x(5)
r2 = x(2) + x(6)
r3 = x(2) - x(6)
r4 = x(3) + x(7)
r5 = x(3) - x(7)
r6 = x(4) + x(8)
r7 = x(4) - x(8)
y(1) = r0 + r4
y(2) = r1 + r5
y(3) = r0 - r4
y(4) = r1 - r5
y(5) = r2 + r7
y(6) = r3 - r6
y(7) = r2 - r7
y(8) = r3 + r6
SPL Compiler: Summary

SPL Program

SPL Formula → Symbol Definition → Template Definition

Parsing

Abstract Syntax Tree → Symbol Table → Template Table

Intermediate Code Generation

Intermediate Code Restructuring

Optimization

Target Code Generation

C, FORTRAN function

Built-in optimizations:
- single static assignment code
- no reuse of temporary vars
- only scalar temporary vars
- constants precomputed

Extensible through templates
**SIMD** Short Vector Extensions

- Extension to instruction set architecture
- Available on most current architectures (SSE on Pentium, AltiVec on Motorola G4)
- Originally for multimedia (like MMX for integers)
- Requires fine grain parallelism
- Large potential speed-up

**Problems:**

- SIMD instructions are architecture specific
- No common API (usually assembly hand coding)
- Performance very sensitive to memory access
- Automatic vectorization very limited
Vector Code Generation from SPL Formulas

Naturally vectorizable construct

\[ A \otimes I_4 \]

vector length

(Current) generic construct **completely vectorizable**:

\[
\prod_{i=1}^{k} P_i D_i (A_i \otimes I_\nu) E_i Q_i
\]

- \( P_i, Q_i \): permutations
- \( D_i, E_i \): diagonals
- \( A_i \): arbitrary formulas
- \( \nu \): SIMD vector length

Vectorization in two steps:

1. Formula manipulation using manipulation rules
2. Code generation (vector code + C code)
Organization

- Mathematical Framework
  - Transforms, Rules, and Formulas
- Formula Generator
  - Transform → Algorithm
- SPL and SPL Compiler
  - Algorithm → Implementation
- Search Engine
  - How to find the best implementation
- SPIRAL system
  - Everything taken together
- Conclusions
### Number of Formulas/Algorithms

<table>
<thead>
<tr>
<th>k</th>
<th># DFT, size $2^k$</th>
<th># DCT-IV, size $2^k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td>126</td>
</tr>
<tr>
<td>4</td>
<td>296</td>
<td>31242</td>
</tr>
<tr>
<td>5</td>
<td>27744</td>
<td>1924443362</td>
</tr>
<tr>
<td>6</td>
<td>162570361280</td>
<td>7343815121631354242</td>
</tr>
<tr>
<td>7</td>
<td>$\sim 1.01 \times 10^{27}$</td>
<td>$\sim 1.07 \times 10^{38}$</td>
</tr>
<tr>
<td>8</td>
<td>$\sim 2.31 \times 10^{61}$</td>
<td>$\sim 2.30 \times 10^{76}$</td>
</tr>
<tr>
<td>9</td>
<td>$\sim 2.86 \times 10^{133}$</td>
<td>$\sim 1.06 \times 10^{153}$</td>
</tr>
</tbody>
</table>

- differ in data flow not in arithmetic cost
- exponential search space
Why Search?

- maaaany different formulas
- large spread in runtimes, even for modest size
- not due to arithmetic cost

DCT, type IV, size 16

~31000 formulas
Search Methods Available in SPIRAL

- Exhaustive Search
- Dynamic Programming (DP)
- Random Search
- Hill Climbing
- STEER (similar to a genetic algorithm)

<table>
<thead>
<tr>
<th></th>
<th>Possible Sizes</th>
<th>Formulas Timed</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exhaust</td>
<td>Very small</td>
<td>All</td>
<td>Best</td>
</tr>
<tr>
<td>DP</td>
<td>All</td>
<td>10s-100s</td>
<td>(very) good</td>
</tr>
<tr>
<td>Random</td>
<td>All</td>
<td>User decided</td>
<td>fair/good</td>
</tr>
<tr>
<td>Hill Climbing</td>
<td>All</td>
<td>100s-1000s</td>
<td>Good</td>
</tr>
<tr>
<td>STEER</td>
<td>All</td>
<td>100s-1000s</td>
<td>(very) good</td>
</tr>
</tbody>
</table>

Search over
- algorithm space and
- implementation options (degree of unrolling)
STEER

Population n:

......

Population n+1:

......

Mutation

Cross-Breeding

expand differently

swap expansions

Survival of Fittest
DCT Type IV Size 16

Fastest Found Formulas

Number of Formulas Timed

Formula runtime in nanoseconds

DP 1-Best | STEER | Exhaustive

Number of formulas timed

DP 1-Best | STEER | Exhaustive
Experimental Results

search methods (applicable to all transforms)

high performance code (compared with FFTW)

different transforms
Vectorized Code

- Speed-ups up to a factor of 2.5
- Beats hand-tuned Intel MKL (< 1024)
- SIMD platforms supported

(Pentium III, SSE)
Learning to Generate Fast Algorithms

- Learns from given dataset (formulas+runtimes) how to design a fast algorithm (breakdown strategy)
- Learns from a transform of one size, generates the best algorithm for many sizes
- Tested for DFT and WHT

<table>
<thead>
<tr>
<th>Size</th>
<th>Number of Formulas Generated</th>
<th>Generated Included the Fastest Known</th>
<th>Top N Fastest Known Formulas in Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{12}$</td>
<td>101</td>
<td>yes</td>
<td>77</td>
</tr>
<tr>
<td>$2^{13}$</td>
<td>86</td>
<td>yes</td>
<td>4</td>
</tr>
<tr>
<td>$2^{14}$</td>
<td>101</td>
<td>yes</td>
<td>70</td>
</tr>
<tr>
<td>$2^{15}$</td>
<td>86</td>
<td>yes</td>
<td>11</td>
</tr>
<tr>
<td>$2^{16}$</td>
<td>101</td>
<td>yes</td>
<td>68</td>
</tr>
<tr>
<td>$2^{17}$</td>
<td>86</td>
<td>yes</td>
<td>15</td>
</tr>
<tr>
<td>$2^{18}$</td>
<td>101</td>
<td>yes</td>
<td>25</td>
</tr>
<tr>
<td>$2^{19}$</td>
<td>86</td>
<td>yes</td>
<td>16</td>
</tr>
<tr>
<td>$2^{20}$</td>
<td>101</td>
<td>yes</td>
<td>16</td>
</tr>
</tbody>
</table>
Organization

- Mathematical Framework
  Transforms, Rules, and Formulas
- Formula Generator
  Transform $\rightarrow$ Algorithm
- SPL and SPL Compiler
  Algorithm $\rightarrow$ Implementation
- Search Engine
  How to find the best implementation
- SPIRAL system
  Everything taken together
- Conclusions
SPIRAL System

DSP transform

Formula Generator

SPL Compiler

Search Engine

Formula Generator

controls
algorithm generation

controls
implementation options

computes runtime (performance)

platform-adapted implementation

user

goes for a coffee

(or an espresso for small transforms)

user

comes back

SPIRAL
Extensibility of SPIRAL

New transforms are readily included on the high level
(easy, due to SPIRAL’s framework)

New constructs and primitives (potentially required by radically different transforms) are readily included in SPL
(moderate effort, due to template mechanism)

New instructions sets available (e.g., SSE) are included by extending the SPL compiler
(doable one time effort)
SPIRAL System: Summary

- Available for download: www.ece.cmu.edu/~spiral
- Easy installation (Unix: configure/make; Windows: install shield)
- Unix/Linux and Windows 98/ME/NT/2000/XP
- Current transforms: DFT, DHT, WHT, DCT/DST type I – IV, MDCT, Filters, Wavelets, Toeplitz, Circulants
- Extensible
Organization

- Mathematical Framework
  - Transforms, Rules, and Formulas
- Formula Generator
  - Transform → Algorithm
- SPL and SPL Compiler
  - Algorithm → Implementation
- Search Engine
  - How to find the best implementation
- SPIRAL system
  - Everything taken together
- Conclusions
Conclusions

Closing the gap between math domain (algorithms) and implementation domain (programs)

- Mathematical computer representation of algorithms
- Automatic translation of algorithms into code

Optimization as intelligent search/learning in the space of alternatives

- High level: Mathematical manipulation of algorithms
- Low level: Coding degrees of freedom