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
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

DSP Transform
(DFT, DCT, Wavelets etc.)

Possible Algorithms

Possible Implementations

Performance Evaluation

Intelligent Search

adapted implementation

given

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

\[
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & i & -1 & -i \\
1 & -1 & 1 & -1 \\
1 & -i & -1 & i \\
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & -1 & 0 \\
0 & 1 & 0 & -1 \\
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 \\
1 & -1 & 0 & 0 \\
0 & 0 & 1 & 1 \\
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
\end{bmatrix}
\]

Fourier transform

Diagonal matrix (twiddles)

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

Kronecker product

Identity

Permutation

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

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
  \[ B \otimes A = L_n^{mn} \left( A \otimes B \right) L_m^{mn} \quad \text{A and B square size } m \text{ and } n \]

- 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} \quad (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
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)**
  \[ DCT^{(II)}_n = \left[ \cos(k(l + 1/2)\pi / n) \right] \]
  \[ DCT^{(IV)}_n = \left[ \cos((k + 1/2)(l + 1/2)\pi / n) \right] \]
  \[ DST^{(I)}_n = \left[ \sin(kl \pi / n) \right] \]

- **modified discrete cosine transform**
  \[ MDCT_{n \times 2^n} = \left[ \cos((k + (n + 1)/2)(l + 1/2)\pi / n) \right] \]

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

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

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

\[ DCT_{2}^{(II)} \rightarrow \text{diag}(1, 1/\sqrt{2}) \cdot F_{2} \]
\[ DCT_{n}^{(II)} \rightarrow P \cdot (DCT_{n/2}^{(II)} \oplus DCT_{n/2}^{(IV)}) \cdot (I_{n/2} \otimes F_{2})^{Q} \]
\[ DCT_{n}^{(IV)} \rightarrow S \cdot DCT_{n}^{(II)} \cdot D \]
\[ DCT_{n}^{(IV)} \rightarrow M_{1} \cdots M_{r} \]
\[ DFT_{n} \rightarrow \text{CosDFT}_{n} + j \cdot \text{SinDFT}_{n} \]
\[ DFT_{n} \rightarrow B \cdot (DCT_{n/2}^{(I)} \oplus DST_{n/2}^{(I)}) \cdot C \]
\[ DFT_{nm} \rightarrow (DFT_{n} \otimes I_{m}) \cdot D \cdot (I_{n} \otimes DFT_{m}) \cdot P \]
\[ \text{CosDFT}_{n} \rightarrow \cdots \text{CosDFT}_{n/2} \cdots DCT_{n/4}^{(II)} \cdots \]
\[ \text{SinDFT}_{n} \rightarrow \cdots \text{SinDFT}_{n/2} \cdots DCT_{n/4}^{(II)} \cdots \]
\[ WHT_{2^{n}} \rightarrow \prod_{i=1}^{n} (I_{2^{n_{1}+\cdots+n_{i-1}}} \otimes WHT_{2^{n_{i}}} \otimes I_{2^{n_{i+1}+\cdots+n_{n}}}) \]
\[ MDCT_{n \times 2^{n}} \rightarrow S \cdot DCT_{n}^{(IV)} \cdot P \]
\[ \text{DTWT}_{2^{n}} = (\text{DTWT}_{2^{n-1}} \oplus I_{2^{n-1}}) L_{2^{n-1}}^{2^{n}} (I_{2^{n-1}} \otimes I_{2^{n-2}} W) \]

12/5/2002 IBM-Thomas T. J. Watson Res. Center
Each WHT algorithm can be represented by a tree, where a node labeled by $n$ corresponds to $WHT_{2^n}$.
Algorithms = Ruletrees = Formulas

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

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

\[ DCT_2^{(II)} \rightarrow \frac{1}{\sqrt{2}} \cdot F_2 \]
Mathematical Framework for Filters and Discrete-Time Wavelet Transforms

- Overlapped direct sum of matrices $A \ (m \times n)$ and $B \ (r \times s)$

$$A \oplus_v B = \begin{bmatrix} A \\ v \\ B \end{bmatrix} = (A \oplus B) \left( I_n \oplus_v I_s \right)$$

$$A \oplus^v B = \begin{bmatrix} A \\ v \\ B \end{bmatrix} = (I_m \oplus^v I_r)(A \oplus B)$$

- Overlapped tensor product

$$I_k \otimes_v B = B \oplus_v B \oplus_v \cdots \oplus_v B \quad \text{column overlap}
$$

$$I_k \otimes^v B = B \oplus^v B \oplus^v \cdots \oplus^v B \quad \text{row overlap}$$
Properties of Overlapped Direct Sums and Tensor Products

- **Column and row overlap direct sum**

\[
A \oplus^v B = \begin{bmatrix} A & \square \\ \square & B \end{bmatrix} = \begin{bmatrix} A^T & \square \\ \square & B^T \end{bmatrix}^T = (A^T \oplus_v B^T)^T
\]

\[
I_n \oplus^v I_s = (I_n \oplus_v I_s)^T
\]

- **Column and row overlapped tensor product**

\[
I_k \otimes^v B_{m \times n} = \left( \bigoplus_{l=1}^k B_{m \times n} \right) \cdot \left( \bigoplus_{l=1}^k I_n \right) = (I_k \otimes B_{m \times n}) \cdot (I_k \otimes v I_n)
\]

\[
I_k \otimes^v B_{m \times n} = \left( \bigoplus_{l=1}^k I_m \right)^T \cdot \left( \bigoplus_{l=1}^k (B_{m \times n})^T \right)^T = (I_k \otimes^v I_m)^T \cdot \bigoplus_{l=1}^k B_{m \times n}
\]

\[
= (I_k \otimes^v I_m) \cdot (I_k \otimes B_{m \times n})
\]
Filters

- Linear convolution in matrix form
  \[ C_n(h) = I_n \otimes^{l-1} [h_0 \ h_1 \ \cdots \ h_{l-1}]^T \]

- Block convolutions
  - Overlap-save rule
    \[ C_n(h) = (I_{n/b} \otimes^{l-1} I_{b+l-1}) (I_{n/b} \otimes C_b(h)) \]
  - Overlap-add rule
    \[ C_n(h) = (I_{m/b} \otimes C_b^T(h))(I_{m/b} \otimes_{l-1} I_{b+l-1}) E_{l-1,l-1} \quad m=n+l-1 \]

- Circular convolution rule for circulant \( K \)
  \[ K(h) = \text{DFT}_{n}^{-1} \cdot \text{diag}(\text{DFT}_{n} \cdot h) \cdot \text{DFT}_{n} \]
  \( h \)-first column of \( K \)
  \( n \)-length of \( h \)
Discrete-Time Wavelet Transform

- DTWT scaling and wavelet expansion coefficients
  \[ c_{0,k} = \sum_{n} x_n h_j \left( 2^j k - n \right) \]  
  scaling coefficients

  \[ d_{j,k} = \sum_{n} x_n h_{j-1}^1 \left( 2^{j-1} k - n \right) \]  
  wavelet coefficients

- Recursive algorithm (Mallat)

  \[ c_{j,k} = \sum_{m} h_{m-2k} c_{j+1,m} \]  
  \[ d_{j,k} = \sum_{m} h_{m-2k}^1 c_{j+1,m} \]
Haar Wavelets – Example

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

- First stage: \( V_2 \Rightarrow V_1 \oplus W_1 \)
  \[
  H = \begin{bmatrix}
  1 & 1 & 0 & 0 \\
  0 & 0 & 1 & 1 \\
  0 & 0 & -1 & 0 \\
  0 & 0 & 1 & -1 \\
  \end{bmatrix}
  = L_2^4 \begin{bmatrix}
  1 & 1 & 0 & 0 \\
  1 & -1 & 0 & 0 \\
  0 & 0 & 1 & 1 \\
  0 & 0 & 1 & -1 \\
  \end{bmatrix}
  = L_2^4 \left( I_2 \otimes F_2 \right)
  \begin{bmatrix}
  c_1 \\
  d_1 \\
  \end{bmatrix}
  = H c_2
  \]

The process is repeated for the upper half of the output

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

- Discrete-Time Wavelet Transform (DTWT) rule

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

- 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

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

\- lo-lowpass odd coeffs., \ le-lowpass even coeffs.
\- ho-highpass odd coeffs., \ he-highpass even coeffs.
DTWT Ruletree – Example

DTWTₙ

DTWTₙ/2

DTWTₙ/4

filters

C(lo)

Cₘ/₂(le)

C(ho)

C(he)

DTWT-convolution rule

m=n+l-1

Overlap-add rule

Cₜ(le)

Circulant rule

K(le₀)

Circular convolution rule

DFT⁻¹

DFT

s⁻=l/2+b-1

l=₀=le padded with b-1 zeroes
Formula for a DCT, size 16

\[
\begin{align*}
&[(2, 16, 9, 5, 3) (4, 15, 8, 13, 7) (6, 14, 10, 12, 11), 16] \\
&(([(2, 8, 5, 3) (4, 7), 8] \cdot ((([2, 4, 3], 4) \cdot ((\text{diag}(1, \sqrt{1/2}) \cdot \text{DFT}_2) \oplus ([(1, 2), 2 \cdot R_{18/8 \pi}]^{(1,2)}, 2)) \cdot (\text{DFT}_2)^{(2,4,3),4} ) \oplus (\text{diag}(\frac{1}{2 \cos(1/16 \pi)}, \frac{1}{2 \cos(3/16 \pi)}, \frac{1}{2 \cos(5/16 \pi)}, \frac{1}{2 \cos(7/16 \pi)}) \cdot (1 \otimes \text{DFT}_2))^{(2,4,3),4} ] \\
&(\text{DFT}_2 \cdot \text{diag}(1, \sqrt{1/2}) \oplus ([(1, 2), 2 \cdot R_{18/8 \pi}]^{(1,2)}, 2)) \cdot [(2, 3, 4), 4] \cdot \begin{bmatrix}
1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1
\end{bmatrix}^{(1,4)(2,3),4}) \\
&(1 \otimes \text{DFT}_2)^{(2,8,5,3)(4,7),8} \oplus ([(2, 5, 4, 3, 7, 6, 8), 8] \cdot (1 \otimes (1 \oplus ([2, 2 \cdot R_{4/4 \pi}]))) \\
&(1 \otimes \text{DFT}_2 \otimes 1) \cdot (1 \otimes ([(1, 2), 2 \cdot R_{18/8 \pi}) \oplus ([(1, 2), 2 \cdot R_{18/8 \pi}] \cdot (1 \otimes \text{DFT}_2 \otimes 1) \cdot ([(1, 2), 2 \cdot R_{49/32 \pi}) \oplus ([(1, 2), 2 \cdot R_{53/32 \pi}) \oplus ([(1, 2), 2 \cdot R_{61/32 \pi}) \oplus ([(1, 2), 2 \cdot R_{61/32 \pi}]) \cdot [(2, 8)(4, 6, 8)]^{(1,8)(2,7)(3,6)(4,5),8}) \\
&(1 \otimes \text{DFT}_2)^{(2,16,9,5,3)(4,15,8,13,7)(6,14,10,12,11),16}]
\end{align*}
\]
Helpful Concept

DSP Transforms

- DSP transform (of size)
- Rule
- Formula/Algorithm

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 → Algorithm
- SPL and SPL Compiler
  Algorithm → Implementation
- Search Engine
  How to find the best implementation
- SPIRAL system
  Everything taken together
- Conclusions
**Formula Generation**

- **Formula Generator**
  - **rules**
  - **transforms**
  - **ruletrees**
  - **formulas**
  - **search engine**

- **Recursive application**
- **Control**
- **Translation**
- **Runtime**
- **Export**

- **Data base (extensible!)**
- **Data type**

- **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**

*Cut here for other optimization problems*
Formula Generator: Summary

Concept:

- easy extensible
- formulas as ruletrees (efficient representation, easy manipulation)
- interface to spl
- allows implementation of search methods

Implementation:

- efficient implementation moves bottleneck to spl compiler
- fully symbolic derivation/manipulation of formulas
- verification
- tools to analyze structure, arithmetic cost of formulas
- interfaces with search engine
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
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

allows extension of SPL
controls loop unrolling
SPL Compiler, 4-point FFT

\[
(\text{compose} \ (\text{tensor} \ (F \ 2) \ (I \ 2)) \ (T \ 4 \ 2) \\
\text{(tensor} \ (I \ 2) \ (F \ 2)) \ (L \ 4 \ 2))
\]

\[
\begin{align*}
\text{f0} & = x(1) + x(3) \\
\text{f1} & = x(1) - x(3) \\
\text{f2} & = x(2) + x(4) \\
\text{f3} & = x(2) - x(4) \\
\text{f4} & = (0.00d0,-1.00d0)*f(3)
\end{align*}
\]

\[
\begin{align*}
y(1) & = \text{f0} + \text{f2} \\
y(2) & = \text{f0} - \text{f2} \\
y(3) & = \text{f1} + \text{f4} \\
y(4) & = \text{f1} - \text{f4}
\end{align*}
\]

\[
\begin{align*}
\text{r0} & = x(1) + x(5) \\
\text{r1} & = x(1) - x(5) \\
\text{r2} & = x(2) + x(6) \\
\text{r3} & = x(2) - x(6) \\
\text{r4} & = x(3) + x(7) \\
\text{r5} & = x(3) - x(7) \\
\text{r6} & = x(4) + x(8) \\
\text{r7} & = x(4) - x(8)
\end{align*}
\]

\[
\begin{align*}
y(1) & = \text{r0} + \text{r4} \\
y(2) & = \text{r1} + \text{r5} \\
y(3) & = \text{r0} - \text{r4} \\
y(4) & = \text{r1} - \text{r5} \\
y(5) & = \text{r2} + \text{r7} \\
y(6) & = \text{r3} - \text{r6} \\
y(7) & = \text{r2} - \text{r7} \\
y(8) & = \text{r3} + \text{r6}
\end{align*}
\]

12/5/2002 IBM-Thomas T. J. Watson Res. Center
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
Templates

\[
(\text{template} \quad (F \ n)[ \ n \geq 1 ] \\
( \text{do} \ i=0, n-1 \\
\quad y(i)=0 \\
\quad \text{do} \ j=0, n-1 \\
\quad \quad y(i) = y(i) + W(n, i*j) * x(j) \\
\quad \text{end} \\
\text{end} ))
\]
(F 2) matches pattern (F n) and assigns 2 to n. Because n=2 satisfies the condition n>=1, the following i-code is generated from the template:

\[
\begin{align*}
\text{do } i &= 0,1 \\
y(i) &= 0 \\
\text{do } j &= 0,1 \\
y(i) &= y(i) + W(2, i*j) * x(j) \\
\text{end} \\
\text{end}
\end{align*}
\]

\[
\begin{align*}
Y(0) &= x(0) + x(1) \\
y(1) &= x(0) - x(1)
\end{align*}
\]
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

\[ A \otimes I_4 \]

(Current) generic construct completely vectorizable:

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

Vectorization in two steps:

1. Formula manipulation using manipulation rules
2. Code generation (vector code + C code)
**SPL Compiler: Vector Code Generation**

\[
DFT_{16} = (DFT_4 \otimes I_4) \cdot T_4^{16} \cdot (I_4 \otimes DFT_4) \cdot L_4^{16}
\]

Symbolic vectorization
(automatic formula manipulation)

\[
DFT_{16} = \left( (I_4 \otimes L_4^8) \cdot (DFT_4 \otimes I_4) \cdot T_4^{16} \right) \cdot \left( (I_4 \otimes L_2^8)(L_4^{16} \otimes I_2)(I_4 \otimes L_4^8) \cdot (DFT_4 \otimes I_4)(I_4 \otimes L_2^8) \right)
\]

Mapping to C code + vector API

```c
LOAD_VECT(xl0, x + 0);
LOAD_VECT(xl4, x + 16);
f0 = SIMD_SUB(xl0, xl4);
LOAD_VECT(xl1, x + 4);
LOAD_VECT(xl5, x + 20);
f1 = SIMD_SUB(xl1, xl5);
... 
yl7 = SIMD_SUB(f1, f4);
STORE_L_8_4(yl6, yl7, y + 24);
yl2 = SIMD_SUB(f0, f5);
yl3 = SIMD_ADD(f1, f4);
STORE_L_8_4(yl2, yl3, y + 8);
```

SSE
SSE2
AltiVec

12/5/2002 IBM-Thomas T. J. Watson Res. Center
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
Why Search?

DCT IV- size $2^4$

~31000 formulas

Search in algorithm space in SPIRAL:
Exhaustive & Random Search, DP, Hill climbing, Genetic Algorithms
Beyond search: design of optimal tree
Why Search?

- maaaany different formulas
- large spread in runtimes, even for modest size
- precisely equal arithmetic cost
- best formula is platform-dependent

Toy problem:
DCT, type IV, size 16
~31000 formulas
## 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>~1.01 $\cdot 10^{27}$</td>
<td>~1.07 $\cdot 10^{38}$</td>
</tr>
<tr>
<td>8</td>
<td>~2.31 $\cdot 10^{61}$</td>
<td>~2.30 $\cdot 10^{76}$</td>
</tr>
<tr>
<td>9</td>
<td>~2.86 $\cdot 10^{133}$</td>
<td>~1.06 $\cdot 10^{153}$</td>
</tr>
</tbody>
</table>

- differ in data flow not in arithmetic cost
- exponential search space
## 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>Exhaustive Search</td>
<td>Very small</td>
<td>All</td>
<td>Best</td>
</tr>
<tr>
<td>Dynamic Programming (DP)</td>
<td>All</td>
<td>10s-100s</td>
<td>(very) good</td>
</tr>
<tr>
<td>Random Search</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

Survival of Fittest

expand differently

swap expansions
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

### Fast Formula Generation Results

<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>
Some Experimental Results
DCT Type IV Size 16

Fastest Found Formulas

Number of Formulas Timed

<table>
<thead>
<tr>
<th>Formula runtime in nanoseconds</th>
<th>DP 1-Best</th>
<th>STEER</th>
<th>Exhaustive</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>800</td>
<td>900</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of formulas timed</th>
<th>DP 1-Best</th>
<th>STEER</th>
<th>Exhaustive</th>
</tr>
</thead>
<tbody>
<tr>
<td>35000</td>
<td>31242</td>
<td>54</td>
<td>423</td>
</tr>
</tbody>
</table>
Experimental Results

- Search methods (applicable to all transforms)
- High performance code (compared with FFTW)
- Different transforms
Some Experimental Results
DFT $2^n$ single precision, Pentium 4, 2.53 GHz, using Intel C compiler 6.0

speedups (to C code) up to factor of 3.1
Generated DFT Code: Pentium 4, SSE2

DFT $2^n$ double precision, Pentium 4, 2.53 GHz, using Intel C compiler 6.0

speedups (to C code) up to factor of 1.8

12/5/2002  IBM-Thomas T. J. Watson Res. Center
Generated DFT Code: Pentium III, SSE

DFT $2^n$ single precision, Pentium III, 1 GHz, using Intel C compiler 6.0

speedups (to C code) up to factor of 2.1

12/5/2002 IBM-Thomas T. J. Watson Res. Center
Other transforms

- WHT has only additions
- Very simple transform

WHT $2^n$

Pentium 4, 2.53 GHz, SSE

2-dim DCT $2^n \times 2^n$

Pentium 4, 2.53 GHz, SSE

Speedups (to C code) up to factor of 3
### Best DFT Trees, size $2^{10} = 1024$

<table>
<thead>
<tr>
<th>Platform/DataType</th>
<th>Pentium 4 float</th>
<th>Pentium 4 double</th>
<th>Pentium III float</th>
<th>AthlonXP float</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scalar</strong></td>
<td><img src="image" alt="Tree" /></td>
<td><img src="image" alt="Tree" /></td>
<td><img src="image" alt="Tree" /></td>
<td><img src="image" alt="Tree" /></td>
</tr>
<tr>
<td><strong>C vect</strong></td>
<td><img src="image" alt="Tree" /></td>
<td><img src="image" alt="Tree" /></td>
<td><img src="image" alt="Tree" /></td>
<td><img src="image" alt="Tree" /></td>
</tr>
<tr>
<td><strong>SIMD</strong></td>
<td><img src="image" alt="Tree" /></td>
<td><img src="image" alt="Tree" /></td>
<td><img src="image" alt="Tree" /></td>
<td><img src="image" alt="Tree" /></td>
</tr>
</tbody>
</table>

Trees platform/datatype dependent
Crosstiming of best trees on Pentium 4

DFT $2^n$ single precision, runtime of best found of other platforms

Software adaptation is necessary

12/5/2002 IBM-Thomas T. J. Watson Res. Center
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
The SPIRAL system is a platform-adapted implementation of the DSP transform, where the user can go for a coffee (or an espresso for small transforms). The system consists of:

- **Search Engine**: Controls the implementation options and runtime (performance) of the DSP transform.
- **Formula Generator**: Specifies the fast algorithm as an SPL formula, which is then translated into platform-adapted implementation as C/Fortran/SIMD code.
- **SPL Compiler**: Takes the fast algorithm as an SPL formula and generates the implementation.

The user specifies the DSP transform and comes back to see the generated code.
SPIRAL System

- Available for download (v3.1): 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, RHT, DCT/DST type I – IV, MDCT, Filters, Wavelets, Toeplitz, Circulants
- Extensible
- New version (4.0) in preparation

- ~ 30 Publications in the areas of Signal Processing, High Performance Computing, Compilers, Machine Learning, Mathematics
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 $\rightarrow$ Algorithm
- SPL and SPL Compiler
  - Algorithm $\rightarrow$ Implementation
- Search Engine
  - How to find the best implementation
- SPIRAL system
  - Everything taken together
- Conclusions
Conclusions

SPIRAL closes the gap between math domain (algorithms) and implementation domain (programs)

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

SPIRAL does automatic optimization by intelligent search/learning in the space of alternatives

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

→ a new paradigm of software development