SPIRAL: A Generator for Platform-Adapted Libraries of Signal Processing Algorithms

Markus Püschel

Faculty
• José Moura (CMU)
• 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
Sponsor

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 in Scientific Computing

(specific 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
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.)

Possible Algorithms

Possible Implementations

Performance Evaluation

Intelligent Search

SPIRAL Search Space

adapted implementation

given Computing Platform
(Pentium III, Pentium 4, Athlon, SUN, PowerPC, Alpha, ... )
DSP Transform

Algorithm
Cooley/Tukey FFT (size 4):

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

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

\[
DFT_n \quad \text{parameterized matrix}
\]

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

- 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
DSP Transforms

**discrete Fourier transform**

\[ DFT_n = \left[ \exp \left( 2\pi k l / n \right) \right] \]

**Walsh-Hadamard transform**

\[ WHT_{2^k} = DFT_2 \otimes \cdots \otimes DFT_2 \]

**discrete cosine and sine Transforms (16 types)**

\[ DCT^{(II)}_n = \left[ \cos(k(l + 1/2)\pi / n) \right] \]

\[ DCT^{(IV)}_n = \left[ \cos((k + 1/2)(l + 1/2)\pi / n) \right] \]

\[ DST^{(I)}_n = \left[ \sin(k\pi / n) \right] \]

\[ MDCT_{n \times 2^n} = \left[ \cos \left( (k + (n + 1)/2)(l + 1/2)\pi / n \right) \right] \]

\[ T \otimes T \]

**modified discrete cosine transform**

**two-dimensional transform**

**Others:** filters, discrete wavelet transforms, Haar, Hartley, …
**Rules = Breakdown Strategies**

\[
\begin{align*}
DCT_2^{(II)} & \rightarrow \text{diag} \left(1, 1 / \sqrt{2}\right) \cdot F_2 \\
DCT_n^{(II)} & \rightarrow P \cdot \left(DCT_{n/2}^{(II)} \oplus DCT_{n/2}^{(IV)}\right) \cdot \left(I_{n/2} \otimes F_2\right)^O \\
DCT_n^{(IV)} & \rightarrow S \cdot DCT_n^{(II)} \cdot D \\
DCT_n^{(IV)} & \rightarrow M_1 \cdots M_r \\
DFT_n & \rightarrow B \cdot \left(DCT_{n/2}^{(I)} \oplus DST_{n/2}^{(I)}\right) \cdot C \\
DFT_{nm} & \rightarrow \left(DFT_{n} \otimes I_m\right) \cdot D \cdot \left(I_n \otimes DFT_m\right) \cdot P \\
F_n(h) & \rightarrow \left(I_{n/d} \otimes^k I_{d+k}\right) \cdot \left(I_{n/d} \otimes F_d(h)\right) \\
F_n(h) & \rightarrow \text{Circ}(\bar{h}) \cdot E \\
DWT_n(W) & \rightarrow \left(DWT_{n/2}(W) \oplus I_{n/2}\right) \cdot P \cdot \left(I_{n/2} \otimes_k W\right) \cdot E \\
WHT_{2^n} & \rightarrow \prod_{i=1}^{n} \left(I_{2^{m_i+\ldots+n_i-1}} \otimes WHT_{2^{n_i}} \otimes I_{2^{n_{i+1}+\ldots+n_t}}\right) \\
MDCT_{n \times 2^n} & \rightarrow S \cdot DCT_n^{(IV)} \cdot P
\end{align*}
\]
Formula for a DCT, size 16

\[
[(2,16,9,5,3)(4,15,8,13,7)(6,14,10,12,11), 16).
\\(\text{([[2,8,5,3)(4,7),8]\cdot(([[2,4,3),4]\cdot((\text{diag}(1, \sqrt\frac{1}{2})\cdot\text{DFT}_2)\oplus(([(1,2),2]\cdot R^{13}_{\frac{\pi}{8}})^{(1,2),2})).}
\\(1_2 \otimes \text{DFT}_2)^{(2,4,3),4})\oplus(\text{diag}(\frac{1}{2\cos(\frac{1}{16}\pi)}, \frac{1}{2\cos(\frac{3}{16}\pi)}, \frac{1}{2\cos(\frac{5}{16}\pi)}, \frac{1}{2\cos(\frac{7}{16}\pi)})\cdot(1_2 \otimes \text{DFT}_2)^{(2,4,3),4}].
\\((\text{DFT}_2\cdot\text{diag}(1, \sqrt\frac{1}{2})\oplus(([(1,2),2]\cdot R^{13}_{\frac{\pi}{8}})^{(1,2),2})).\cdot([2,3,4],4)\cdot\left[
\begin{array}{cccc}
1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 \\
\end{array}
\right])^{(1,4)(2,3),4}).
\\(1_4 \otimes \text{DFT}_2)^{(2,8,5,3)(4,7),8})\oplus(\text{[(2,5,4,3,7,6,8),8]\cdot(1_2 \oplus((1,2),2]\cdot R^7_{\frac{\pi}{4}}))).
\\(1_2 \otimes \text{DFT}_2 \otimes 1_2) \cdot (1_4 \oplus ((1,2),2]\cdot R^{13}_{\frac{\pi}{8}} \oplus ((1,2),2]\cdot R^1_{\frac{\pi}{8}})) \cdot (1_1 \otimes \text{DFT}_2 \otimes 1_4) \cdot (([(1,2),2]\cdot R^{49}_{\frac{32\pi}}) \oplus ((1,2),2]\cdot R^{53}_{\frac{32\pi}}) \oplus ((1,2),2]\cdot R^{61}_{\frac{32\pi}}) \oplus ((1,2),2]\cdot R^{65}_{\frac{32\pi}}) \oplus ((1,2),2]\cdot R^{69}_{\frac{32\pi}}) \oplus\cdot([2,8)(4,6),8])^{(1,8)(2,7)(3,6)(4,5),8}).
\\(1_8 \otimes \text{DFT}_2)^{(2,16,9,5,3)(4,15,8,13,7)(6,14,10,12,11),16}.
DSP Transform

Algorithm (Formula)

Implementation
Formulas in SPL

• • • •

• • • •
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

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

#codetype
complex real

\[
\begin{align*}
f_0 &= x(1) + x(3) \\
f_1 &= x(1) - x(3) \\
f_2 &= x(2) + x(4) \\
f_3 &= x(2) - x(4) \\
f_4 &= (0.00d0, -1.00d0) * f(3) \\
y(1) &= f_0 + f_2 \\
y(2) &= f_0 - f_2 \\
y(3) &= f_1 + f_4 \\
y(4) &= f_1 - f_4
\end{align*}
\]

\[
\begin{align*}
r_0 &= x(1) + x(5) \\
r_1 &= x(1) - x(5) \\
r_2 &= x(2) + x(6) \\
r_3 &= x(2) - x(6) \\
r_4 &= x(3) + x(7) \\
r_5 &= x(3) - x(7) \\
r_6 &= x(4) + x(8) \\
r_7 &= x(4) - x(8) \\
y(1) &= r_0 + r_4 \\
y(2) &= r_1 + r_5 \\
y(3) &= r_0 - r_4 \\
y(4) &= r_1 - r_5 \\
y(5) &= r_2 + r_7 \\
y(6) &= r_3 - r_6 \\
y(7) &= r_2 - r_7 \\
y(8) &= r_3 + r_6
\end{align*}
\]
SPL Compiler: Summary

SPL Program

SPL Formula → Symbol Definition → Template Definition

Abstract Syntax Tree

Parsing

Intermediate Code Generation

Intermediate Code Restructuring

Optimization

Target Code Generation

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

Extensible through templates

C, FORTRAN function
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 length = 4 (4-way)
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)
DSP Transform

Algorithm (Formula)

Implementation

Search
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

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

<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>
SPIRAL system

user

does for a coffee

goes for a coffee

DSP transform

specifies

SPIRAL Formula Generator

controls
algorithm generation

fast algorithm
as SPL formula

controls
implementation options

SPL Compiler

C/Fortran/SIMD code

runtime (performance)

platform-adapted
implementation

Search Engine

 Angebote (oder an espresso for small transforms)

comes back

comes back
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, RHT, DCT/DST type I – IV, MDCT, Filters, Wavelets, Toeplitz, Circulants
- Extensible
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)

(Pentium III, SSE)
Conclusions

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

- Mathematical computer representation of algorithms
- Automatic translation of algorithms into code
- Implement constructs not transforms

Optimization as intelligent search/learning in the space of alternatives

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

http://www.ece.cmu.edu/~spiral