Generating Platform-Adapted DSP Libraries Using SPIRAL

José Moura (CMU)
Jeremy Johnson (Drexel)
Robert Johnson (MathStar Inc.)
David Padua (UIUC)
Viktor Prasanna (USC)
Markus Püschel (CMU)
Bryan Singer (CMU)
Manuela Veloso (CMU)
Jianxin Xiong (UIUC)

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.
SPIRAL

Automates the

Implementation

- cuts development costs
- coding less error-prone

Optimization

- code manipulation techniques like, e.g., unrolling
cannot be done by hand in reasonable time
- allows systematic exploration of alternatives
both at algorithmic level and code optimizations

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,
platform-adapted signal processing transforms
Organization

- SPIRAL approach
- SPIRAL system
- Some experimental results
- Recent work
Key Observations

- For every DSP transform there are exponential many different algorithms, which do not differ in arithmetic cost
- The best algorithm is highly platform dependent
- The best algorithm is hard to determine
SPIRAL Methodology

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

SPIRAL Search Space

Possible Algorithms

Possible Implementations

Performance Evaluation

Intelligent Search

adapted implementation

given → Computer Architecture

Uniprocessor:
- Pentium
- SUN
- Alpha

Multiprocessor
- Hardware
DSP Algorithms: Example 4-point DFT

Cooley/Tukey FFT (size 4):

Fourier transform \( DFT_4 \) = \( (DFT_2 \otimes I_2) \cdot T^4_2 \cdot (I_2 \otimes DFT_2) \cdot L^4_2 \)

Kronecker product \( \otimes \)  Identity \( I \)  Permutation \( L \)

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

- product of structured sparse matrices
- mathematical notation
Transforms, Rules, & Formulas

DSP transform

\[ \text{DFT}_{nm} \]

a matrix

Rule

\[ \text{DFT}_{nm} \rightarrow (\text{DFT}_n \otimes I_m) \cdot D \cdot (I_n \otimes \text{DFT}_m) \cdot P \]

- a breakdown strategy
- product of sparse matrices

Formula

\[ \text{DFT}_8 = (F_2 \otimes I_4) \cdot D \cdot (I_2 \otimes (I_2 \otimes F_2 \cdots )) \cdot P \]

- arises from recursive application of rules
- product of sparse matrices
- uniquely defines an algorithm
Algorithms = Ruletrees = Formulas

\[
\begin{align*}
DCTII_8 & \\
R1 & \\
DCTII_4 & \\
R1 & DCTIV_4 \\
DCTII_2 & \\
R3 & F_2 \\
DCTIV_2 & \\
R6 & DSTII_2 \\
R4 & F_2 \\
DCTII_4 & \\
R1 & \\
DCTII_2 & \\
R3 & F_2 \\
DCTIV_2 & \\
R6 & DSTII_2 \\
R4 & F_2 \\
\end{align*}
\]

\[
DCTII_n \rightarrow P \cdot (DCTII_{n/2} \oplus DCTIV_{n/2}) \cdot (F_2 \otimes I_{n/2})
\]

\[
DCTIV_n \rightarrow P \cdot DCTII_n \cdot S
\]

\[
DCTII_2 \rightarrow \frac{1}{\sqrt{2}} \cdot F_2
\]
Number of Formulas/Algorithms

Currently 12 transforms and 31 rules:

<table>
<thead>
<tr>
<th>k</th>
<th># DFTs, size $2^k$</th>
<th># DCTIVs, 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>$\approx 1.01 \times 10^{27}$</td>
<td>$\approx 1.07 \times 10^{38}$</td>
</tr>
<tr>
<td>8</td>
<td>$\approx 2.31 \times 10^{61}$</td>
<td>$\approx 2.30 \times 10^{76}$</td>
</tr>
<tr>
<td>9</td>
<td>$\approx 2.86 \times 10^{133}$</td>
<td>$\approx 1.06 \times 10^{153}$</td>
</tr>
</tbody>
</table>

exponential search space
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 Compiler, 4-point FFT

Fast algorithm as formula as SPL program

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

#codetype

- complex
- real

$f_0 = x(1) + x(3)$
$f_1 = x(1) - x(3)$
$f_2 = x(2) + x(4)$
$f_3 = x(2) - x(4)$
$f_4 = (0.00d0, -1.00d0) * f(3)$

$y(1) = f_0 + f_2$
$y(2) = f_0 - f_2$
$y(3) = f_1 + f_4$
$y(4) = f_1 - f_4$

$r_0 = x(1) + x(5)$
$r_1 = x(1) - x(5)$
$r_2 = x(2) + x(6)$
$r_3 = x(2) - x(6)$
$r_4 = x(3) + x(7)$
$r_5 = x(3) - x(7)$
$r_6 = x(4) + x(8)$
$r_7 = x(4) - x(8)$

$y(1) = r_0 + r_4$
$y(2) = r_1 + r_5$
$y(3) = r_0 - r_4$
$y(4) = r_1 - r_5$
$y(5) = r_2 + r_7$
$y(6) = r_3 - r_6$
$y(7) = r_2 - r_7$
$y(8) = r_3 + r_6$
The SPL Compiler

- Parsing
  - SPL Formula
  - Symbol Definition
  - Template Definition
  - Abstract Syntax Tree
  - Symbol Table
  - Template Table

- Intermediate Code Generation
  - I-Code

- Intermediate Code Restructuring
  - I-Code

- Optimization
  - I-Code

- Target Code Generation
  - FORTRAN, C
Search Methods Available in SPIRAL

- Exhaustive Search
- Dynamic Programming (DP)
- Random Search
- 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>Good</td>
</tr>
<tr>
<td>Random</td>
<td>All</td>
<td>User decided</td>
<td>Poor to fair</td>
</tr>
<tr>
<td>STEER</td>
<td>All</td>
<td>100s-1000s</td>
<td>Very good</td>
</tr>
</tbody>
</table>

- Search over new user-defined transforms and breakdown rules
- Search over formulas and options to SPL compiler
Summary: SPIRAL Architecture

DSP transform
(symbolically specified)

Formula generator
(rule based)

DSP algorithm as SPL program
(on out of many possible)

SPL compiler

Performance evaluation

Search engine

feedback loop
Organization

- SPIRAL approach
- SPIRAL system
- Some experimental results
- Recent work
The SPIRAL System: Implementation

- Infrastructure of SPIRAL is based on the computer algebra system and language GAP (http://www-gap.dcs.st-and.ac.uk/~gap/)
  - command line interface
  - symbolic (exact) computation with DSP formulas
  - full-fledged programming environment
- Formula generator and search engine implemented in GAP
- SPL compiler implemented in C
The SPIRAL System: Main Features

- Easy installation from one source on
  - Unix based systems (configure – make)
  - native Windows systems (Visual C/Intel compiler make)
- DSP transforms: DFT, DCTs, DSTs, WHT, Haar transform, …
- new transforms can easily be included
- multi-dimensional transforms automatically supported
- composed DSP transforms supported
- verification of generated code
- programming environment included (GAP)
- online documentation

**download at:**  [www.ece.cmu.edu/~spiral](http://www.ece.cmu.edu/~spiral)
Implementing a DFT of size 1024 in C:

SPIRAL command prompt

spiral> S := Transform("DFT", 1024);
spiral> Implement(S, rec(search := "DP", language := "c"));

search method: dynamic programming

target language

C function in working directory
SPIRAL System Examples II

Implementing an 8 x 8 DCT of type 2 in Fortran:

```
spiral> S := Transform("DCT2", 8);
spiral> S1 := TensorSPL(S, S);
spiral> Implement(S1, rec(search := "STEER", language := "f77"));
```

search method: STEER
SPIRAL System Examples III

Implementing a composed transform in C:

```plaintext
spiral> S1 := Transform("DFT", 8);
spiral> S2 := DiagSPL([1, 2, 4, 2, 3, 5, 1, -2]);
spiral> S3 := Transform("DCT3", 8);
spiral> S := S1 * S2 * S3;
spiral> Implement(S, rec(search := "TimedSearch",
    timeLimit := 30,
    language := "c");
```

search method:
timed search 30 minutes

a DCT type 3 followed by scaling followed by a DFT
Organization

- SPIRAL approach
- SPIRAL system
- Some experimental results
- Recent work
Search Space and Varying Performance

WHT(2^{10}): 51,819 (binary) ruletrees = formulas

- large spread in runtime
- not due to arithmetic cost
- good ones are rare
Comparison Search Methods I

DCT, type IV, size 16

Fastest Found Formulas

<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></td>
<td></td>
<td></td>
</tr>
<tr>
<td>800</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>600</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Number of Formulas Timed

<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></td>
<td></td>
<td>31242</td>
</tr>
<tr>
<td>30000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DP and STEER perform well
Comparison Search Methods II

across transforms of size 16
SPIRAL vs. FFTW
(lower = better)

Pentium III/Linux/gcc
Athlon/Linux/gcc
Pentium III/Win2000/Intel compiler
comparable performance
Organization

- SPIRAL approach
- SPIRAL system
- Some experimental results
- Recent work
Learning instead of Searching

- **Method:**
  - Runs a number of formulas of one size
  - Analyzes the cache misses caused by different parts of the formulas
  - Then design fastest formulas of different sizes, even larger sizes!

- Designs fast formulas of sizes that it has never even timed before
- Designed fastest known formulas for 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>
SPIRAL SIMD

joint work with
Franz Franchetti, Christoph Ŧüberhuber,
Technical University Vienna

- Portable SIMD Support (SSE; planned: SSE2, AltiVec),
  based on Compiler Support
- Handle $A \otimes I_n$ and $I_n \otimes A$
- Support for Diagonals and Permutations
- Unrolled code and loop code

DFT$_2 \otimes I_4$
Experimental Results

Pentium4
SSE - float
Windows 2000
Intel C++ Compiler 5.0
Spiral 3.1

FFT: Benchmark

DCT2xDCT2: Speed-up

WHT
FFT

SPIRAL SIMD
Intel MKL

Speed-up
Summary

- SPIRAL
  - generates platform-adapted code for linear DSP transforms
  - is extensible to include new transforms
  - easily installs on a variety of platforms
- The generated code is verified and very competitive

download at:  www.ece.cmu.edu/~spiral