High-Performance Code Generation for FIR Filters and the Discrete Wavelet Transform Using SPIRAL

Aca Gacic
Markus Püschel
José M. F. Moura

SPIRAL project
http://www.spiral.net

Electrical and Computer Engineering Department
Carnegie Mellon University
Motivation

- FIR filters - image enhancement, equalization, speech synthesis, biomedical signal processing, ...
- Wavelets - image compression, detection, denoising, signal recovery
- Numerically intensive - significant impact on application performance

High-Performance Implementations (How-To)

- Choose Fast Algorithm
  - Arithmetic cost – only a rough estimate of performance
  - Many algorithms with similar cost – which one?
- Design Efficient Code
  - Architecture conscious – machine dependent
  - Obsolete when platform is changed/upgraded
- Best implementation: Algorithm + Machine + Compiler
  - Requires experts in algorithms and computer architecture
  - Frequent re-implementation

Better way: automatic performance tuning
We want to close this gap

Existing FIR filtering and DWT software
Matlab®, Mathematica, S+Wavelets, WaveLab, Wave++, IPP
Application oriented – not machine specific, or
Hand-tuned code for specific platform
Automatic platform-adapted solutions not available!

We want to close this gap

Automatic performance tuning packages
ATLAS, PHiPAC, SPARSITY, FFTW, SPIRAL
Include several basic linear algebra and DSP functions
No filtering and wavelet kernels!
**SPiral System**

Generator of optimized DSP transform implementations

- **DSP transform** specifies
- User goes for a coffee

---

**Formula Generator**
- Fast algorithm as SPL formula
- Controls algorithm generation

**SPL Compiler**
- C/Fortran/SIMD code
- Controls implementation options
- Runtime on given platform

**Search Engine**
- Platform-adapted implementation
- Comes back

(or an espresso for small transforms)
SPIRAL’s Four Key Concepts

Transform

DFT<sub>n</sub> ? e<sup>j2πjkln</sup> ? DFT<sub>m</sub> ? I<sub>n</sub> ? T<sup>nm</sup> ? I<sub>m</sub> ? DFT<sub>m</sub> ? L<sup>nm</sup>

- a breakdown strategy - product of sparse matrices
- captures important structural information

Rule

DFT<sub>nm</sub> ? DFT<sub>n</sub> ? I<sub>m</sub> ? T<sup>nm</sup> ? I<sub>n</sub> ? DFT<sub>m</sub> ? L<sup>nm</sup>

- recursive application of rules
- uniquely defines an algorithm
- efficient representation
- easy manipulation

Rule Tree

- few constructs and primitives
- uniquely defines an algorithm
- can be translated into code

Parameterized matrix

Diagonal matrix (twiddles)

Kronecker product

Identity

Permutation
**Rule Trees = Formulas = Algorithms**

**Rule Trees**
- Cooley-Tukey
  - DFT₂
  - Cooley-Tukey
  - DFT₂

**Formulas**
- DFT₈ \( \Rightarrow \) (DFT₂ \( \Rightarrow \) I₂) \( \Rightarrow \) T₂ \( \Rightarrow \) I₂ \( \Rightarrow \) DFT₄ \( \Rightarrow \) T₄ \( \Rightarrow \) I₄ \( \Rightarrow \) DFT₂ \( \Rightarrow \) Cooley-Tukey

**Algorithms**
- Increasing level of abstraction
  - \( x[0], x[4], x[1], x[5], x[2], x[6], x[3], x[7] \)
  - \( X[0], X[1], X[2], X[3], X[4], X[5], X[6], X[7] \)

**Transform**
- DFT₂
- DFT₄
- DFT₈

**Increasing level of abstraction**
- \( L_DFT(I_DFT) \)
Examples: Rules and Rule Trees

**Trigonometric Rule**
\[ DFT_n \rightarrow \text{CosDFT}_n \rightarrow i \rightarrow \text{SinDFT}_n \]

**DCT-Recursive Rule**
\[ \text{CosDFT}_n \rightarrow S_n \rightarrow \text{CosDFT}_{n/2} \rightarrow \text{DCT}^{(II)} \rightarrow S_n \rightarrow L_2^n \]
\[ \text{SinDFT}_n \rightarrow S_n \rightarrow \text{SinDFT}_{n/2} \rightarrow \text{DCT}^{(II)} \rightarrow S_n \rightarrow L_2^n \]

**Type-Split Rule**
\[ \text{DCT}^{(II)}_n \rightarrow P_n \rightarrow \text{DCT}_{n/2}^{(II)} \rightarrow \text{DCT}^{(IV)}_n \rightarrow \text{IFIR} \rightarrow \text{HTR} \rightarrow F_2 \rightarrow \text{P}^n \]

**Type-Conversion Rule**
\[ \text{DCT}^{(IV)}_n \rightarrow S_n \rightarrow \text{DCT}^{(II)}_n \rightarrow D_n \]

**Recursive RHT Rule**
\[ \text{RHT}_{2^k} \rightarrow \text{RHT}_{2^k} \rightarrow \text{I}_{2^k} \rightarrow F_2 \rightarrow \text{I}_{2^k} \rightarrow L_2^k \]

SPIRAL also includes search over implementation options, such as the **degree of loop unrolling**
FIR Filters and the DWT in SPIRAL: Challenges

- Existing filtering and wavelet algorithm representations do not capture all important structural information.
- SPIRAL uses a concise math framework to represent many algorithms for several transforms (DFT, DCTs, DHT, RDFT, ...).
- Filtering and wavelet algorithms have considerably different structure from the current SPIRAL transforms.
  - Current framework not enough to capture algorithms’ structure.
  - Existing algorithm representations need to be “spiralized” – i.e., captured concisely using the rule formalism.

Need changes on several levels:
- New transforms, constructs, rules, translation templates.
- Modified search.
- Need to rephrase several concepts of the framework.

Existing framework has to be redesigned to accommodate new constructions and decomposition strategies.

New framework has to be integrated in SPIRAL.
**FIR Filters: Rules and Algorithms**

**FIR Filter Transform**

\[ y_n = h_n \ast x_n \]

\[ F_n(h) = h_0 \ast x_0 + h_1 \ast x_1 + \cdots + h_k \ast x_k \]

**Baseline Rule**

\[ F_n(h) \ast I_n \Rightarrow h^T \]

**SPIRAL Rule** captures the structure

**New constructs**

- Column and row overlapped direct sum
- Overlapped tensor product

**Time Domain Methods**

**Overlap-Add Rule**

\[ F_n(h) \ast I_{n/b} \Rightarrow F_b(h) \]

- Localized access of the input data
- Input is blocked into length b segments
**Overlap-Save Rule**

\[ F_n \mathbf{h} \ L \ k_1 \ \mathbf{m}_b \ \ k_1 \ F_b \mathbf{h} \ R \ k_1 \]

- Localized storage of the output
- Output segments are computed independently

**Blocking Rule**

\[ F_n \mathbf{h} I_n/b \ \mathbf{v}_i \ b \ T \mathbf{h}_i \]

- Multilevel blocking
- Control over input & output locality

---

**Frequency Domain Methods**

**Expanded Circulant Rule**

\[ F_n \mathbf{h} \ \mathbf{C} \mathbf{E}_{k,n} \ \mathbf{h} \mathbf{E}_{n,k} \]

- Circulant matrix
- Sparse matrix
- Real DFT

**Cyclic-RDFT Rule**

\[ \mathbf{C} \mathbf{RDF}_{n/2} \ \mathbf{I}_2 \ \mathbf{X}(W^k) \mathbf{I}_{n/2} \]

- Time domain - better data access structure
- Frequency domain - potentially reduce arithmetic cost
DWT: Rules and Algorithms

Filter Bank – Mallat’s Algorithm

\[ \{x_k \} = \{c_{j,k} \} \]

\[ \begin{align*}
  g(z^{-1}) & \rightarrow \downarrow 2 \rightarrow \{d_{j-1,k} \} \\
  h(z^{-1}) & \rightarrow \downarrow 2 \rightarrow \{c_{j-1,k} \} \\
  h(z^{-1}) & \rightarrow \downarrow 2 \rightarrow \{d_{j,k} \} \\
  h(z^{-1}) & \rightarrow \downarrow 2 \rightarrow \{c_{j,k} \} \\
  & \quad \ldots \\
  h(z^{-1}) & \rightarrow \downarrow 2 \rightarrow \{c_{0,k} \}
\end{align*} \]

DWT of \( \{x_n\} \)

Mallat Rule

\[ \text{DWT}_n(h, g) \land \text{DWT}_{n/2}(h, g) \land I_{n/2} \land 2 \land F_{n/2}(h) \land F_{n/2}(g) \land E_{n,l,r}^* \]

Fused Mallat Rule

\[ \text{DWT}_n(h, g) \land \text{DWT}_{n/2}(h, g) \land I_{n/2} \land H_n(h, g) \land E_{n,l,r}^* \]

- Redundant computations removed
- Filter structure lost

Extension matrix

\[ H_n(h, g) \land I_{n/2} \land I_{n/2} \land H^T \land I_{n/2} \land g^T \]

single-level DWT
Polyphase Representation

\[ g(z^{-1}) \downarrow 2 \quad \{x_k\} \quad \equiv \quad \{x_k\}_{even} \quad \downarrow 2 \quad P(z^{-1}) \]

\[ h(z^{-1}) \downarrow 2 \quad \{x_k\}_{odd} \quad \downarrow 2 \quad P(z^{-1}) \]

Polyphase Rule

Gateway to frequency domain computation

\[ H_n(h,g) \equiv F_{n/2}^T(h_o) \equiv F_{n/2}^T(h_e)!! F_{n/2}^T(g_o) \equiv F_{n/2}^T(g_e)!! L_{n/2}^{n/2} \]

Lifting Scheme

Primal
(Predict)

\[ h_e(z^{1}) \quad h_o(z^{1}) \quad K \quad 0 \quad s_i(z^{1}) \quad 1 \quad 0 \quad s_0(z^{1}) \quad 1 \quad 0 \]

Dual
(Update)

\[ g_e(z^{1}) \quad g_o(z^{1}) \quad 0 \quad 1/K \quad 1 \quad 0 \quad t_0(z^{1}) \quad 1 \quad 0 \]

Lifting steps

- Asymptotically reduce cost by 50%
- In-place computations

Lifting Rule

\[ H_n(h,g) \equiv K \equiv F_{n/2}^T(s_i) \equiv F_{n/2}^T(t_o) \equiv L_{n/2}^{n/2} \]
Examples: FIR Filter and DWT Rule Trees

Exceedingly large number of trees even for modest size transforms

All these rule trees form an extensive search space of alternative algorithms in SPIRAL
Time-Domain Methods
FIR Filter

- overlap-add, overlap-save, and blocking
- all methods have the same cost
- baseline method = overlap-add with b=1
- sanity check – percentage of the peak performance

60-70% improvement over baseline
Same arithmetic cost

comparison of different time-domain methods
Frequency Domain vs. Time Domain

**FIR Filter**
- Frequency domain – potentially lower cost
- Time domain – better data access patterns

**Circulant**
- “half–dense” circulants emerge in frequency domain methods

Considerable discrepancy between arithmetic costs and actual runtimes

- FD methods better on large “fat” circulants
- Filter lengths > 64

- arithmetic cost cross-over
- runtime cross-over
Daubechies D₄ DWT
Short Wavelet

- single-level DWT using D₄
- 3 different classes of rule trees based on the initial rule
  - Fused Mallat Rule (direct method)
  - Polyphase Rule (frequency domain)
  - Lifting Rule (lifting scheme)

Frequency domain trees not found
Lifting scheme – 30% lower cost – slower
Daubechies $D_{30}$ DWT Experiment
Long Wavelet

Different methods are found for different size ranges
Best implementations vary from machine to machine
Summary

Fast automatically generated code for FIR filters and DWT
- Significant contribution – no existing solutions

Preliminary results
- Arithmetic cost not a reliable predictor of performance
- Best implementation not obvious for a human programmer
- Best implementations vary over different computer platforms

What lies ahead?

Further algorithmic and implementation optimizations
- Fusion of the extension, exploiting in-place structure, scheduling,..

Entire class of wavelets and extension methods
- Orthogonal, biorthogonal, frames,…
- Periodic, symmetric, polynomial extension, zero-padding

Extend to other wavelet transforms
- 2D-DWT, M-band, wavelet packet transform

Utilize special instruction set extensions
- Single Instruction Multiple Data (SIMD)
- Fused Multiply-Add (FMA)