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

Carnegie Mellon

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

Carnegie Mellon

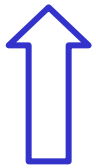
Existing FIR filtering and DWT software

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

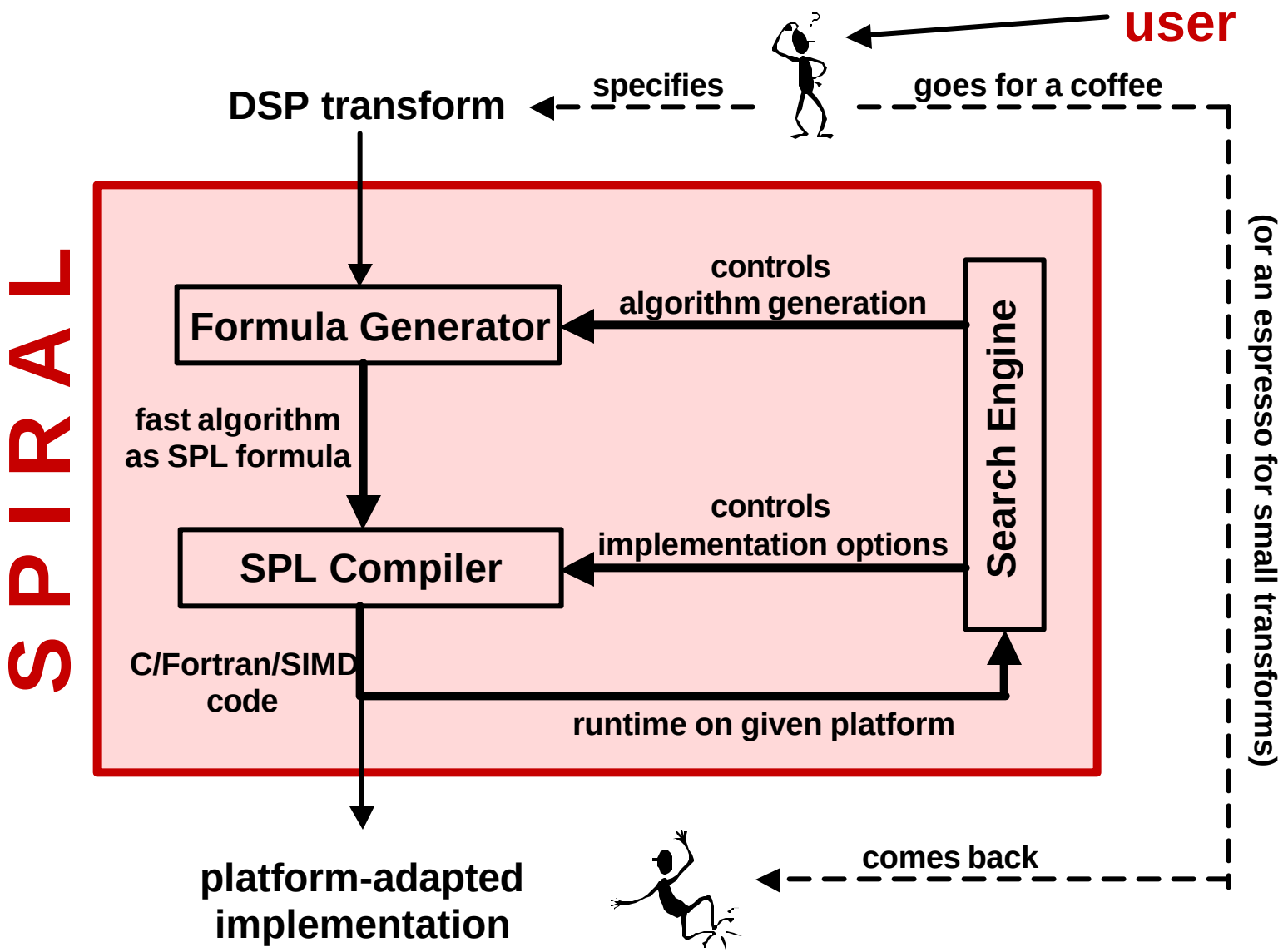
Include several basic linear algebra and DSP functions

No filtering and wavelet kernels !

Carnegie Mellon

SPIRAL system

Generator of optimized DSP transform implementations



Carnegie Mellon

SPIRAL

SPIRAL's Four Key Concepts

Transform

$$\mathrm{DFT}_n = \left[e^{2\pi i kl/n} \right]_{k,l=0,\ldots,n-1}$$

parameterized matrix

Rule

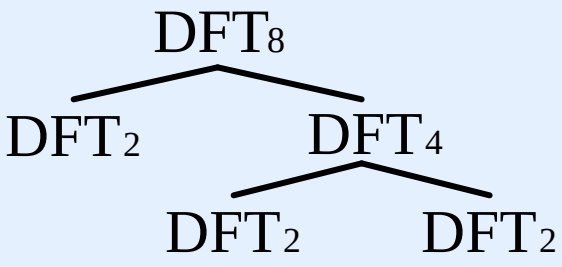
$$\mathrm{DFT}_{nm} = (\mathrm{DFT}_n \otimes I_m)\, T^{nm}_n\, (I_n \otimes \mathrm{DFT}_m)\, L^{nm}_m$$

Diagonal matrix (twiddles)

Kronecker product — Identity — Permutation

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

Rule Tree



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

Formula

$$\mathrm{DFT}_8 = (F_2 \otimes I_4)\, T^8_4\, (I_2 \otimes (I_2 \otimes F_2))\, L^8_2$$

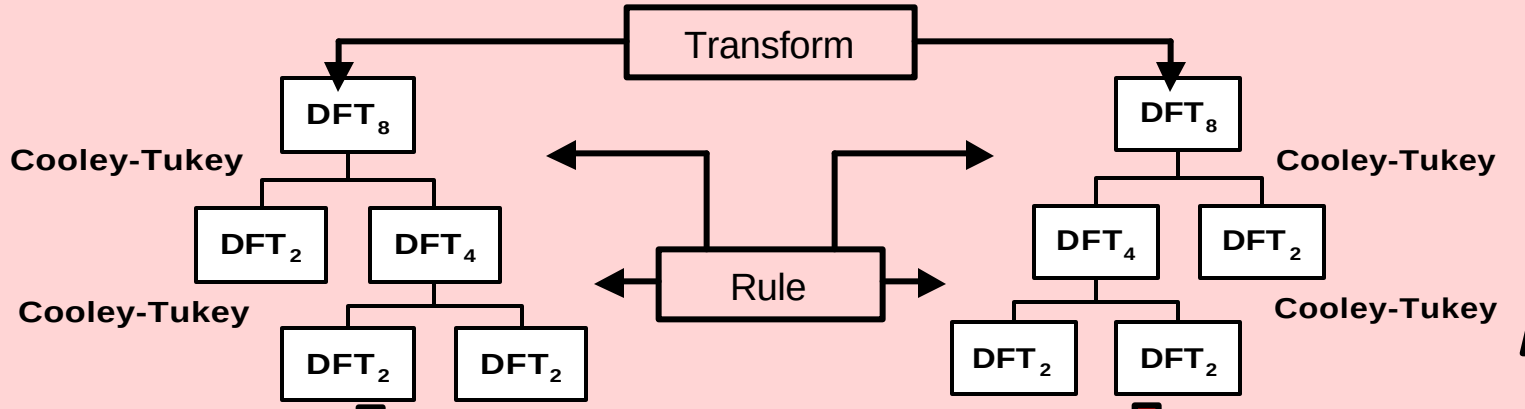
- few constructs and primitives
- uniquely defines an algorithm
- can be translated into code

Carnegie Mellon

SPIRAL

Rule Trees = Formulas = Algorithms

Rule Trees



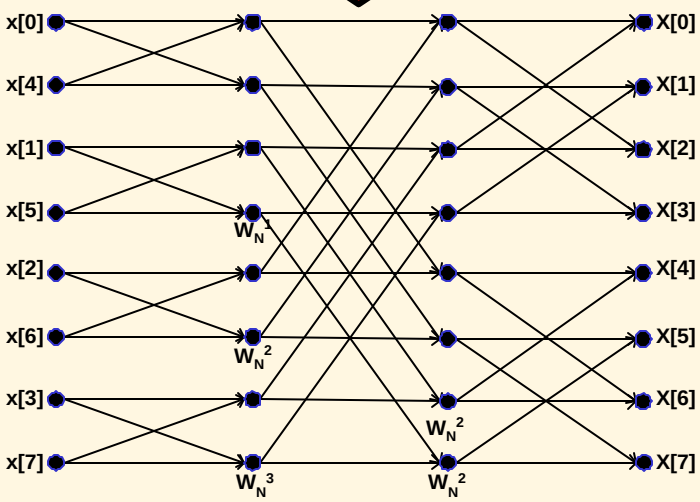
Formulas

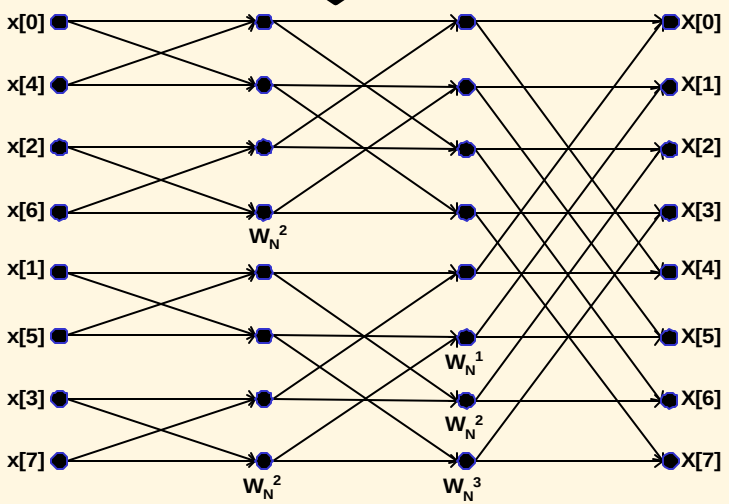
$$DFT_8 \;?\; (DFT_2 \;?\; I_4)\;?T_2^8\;?(I_2 \;?\; DFT_4)\;?L_4^8$$

$$DFT_8 \;?\; (DFT_4 \;?\; I_2)\;?T_4^8\;?(I_4 \;?\; DFT_2)\;?L_2^8$$

$$(DFT_2 \;?\; I_2)\;?T_2^4\;?(I_2 \;?\; DFT_2)\;?L_2^4$$

Algorithms





Increasing level of abstraction

Carnegie Mellon

SPIRAL

Examples: Rules and Rule Trees

Trigonometric Rule

$$\mathrm{DFT}_n \rightarrow \mathrm{CosDFT}_n + i \cdot \mathrm{SinDFT}_n$$

DCT-Recursive Rule

$$\mathrm{CosDFT}_n \rightarrow S_n \cdot \left(\mathrm{CosDFT}_{n/2} + \mathrm{DCT}^{(II)}_{n/4} \cdot S_n \cdot L^n_2\right)$$

$$\mathrm{SinDFT}_n \rightarrow S_n \cdot \left(\mathrm{SinDFT}_{n/2} + \mathrm{DCT}^{(II)}_{n/4} \cdot S_n \cdot L^n_2\right)$$

Type-Split Rule

$$\mathrm{DCT}^{(II)}_n \rightarrow P_n \cdot \left(\mathrm{DCT}^{(II)}_{n/2} + \mathrm{DCT}^{(IV)}_{n/2} \cdot P_n\right) \cdot \left(\mathrm{I}_{n/2} + F_2 \cdot P_n\right)$$

Type-Conversion Rule

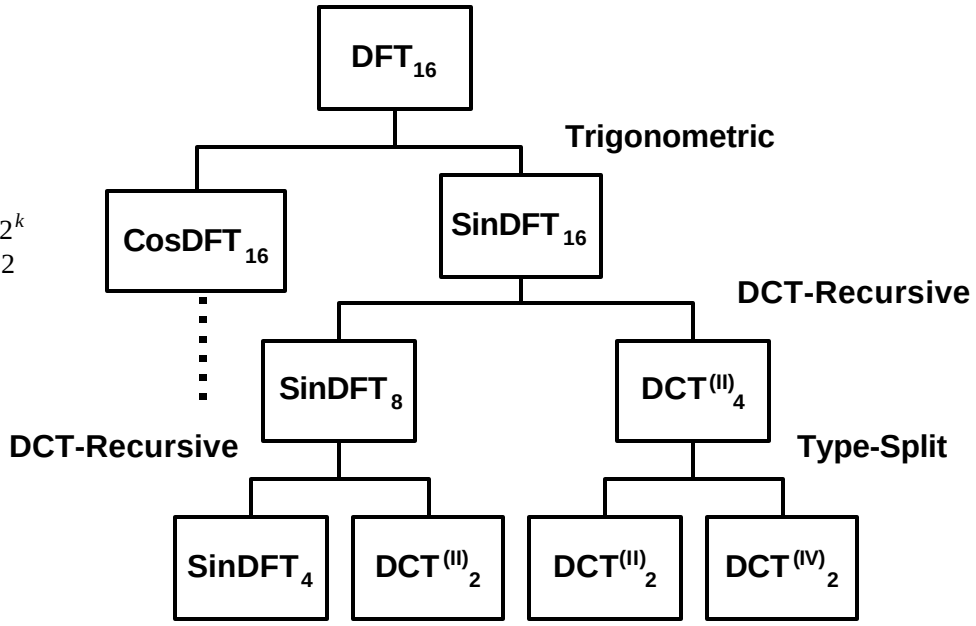
$$\mathrm{DCT}^{(IV)}_n \rightarrow S_n \cdot \mathrm{DCT}^{(II)}_n \cdot D_n$$

Recursive RHT Rule

$$\mathrm{RHT}_{2^k} \rightarrow \left(\mathrm{RHT}_{2^{k-1}} + \mathrm{I}_{2^{k-1}}\right) \cdot F_2 + \mathrm{I}_{2^{k-1}} \cdot L^{2^k}_2$$

Number of rule trees grows exponentially with

- ~~Size of the transform~~
- ~~Number of applicable rules~~



SPIRAL also includes search over implementation options, such as the degree of loop unrolling

Carnegie Mellon

SPIRAL

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

$$y_n = h_n * x_n = \sum_{i=0}^{k-1} h_i x_{n-i}$$

standard "sum" notation

$$F_n(\mathbf{h}) = \begin{bmatrix} h_0 & & & \\ h_1 & h_0 & & \\ \vdots & h_1 & \ddots & \\ h_{k-1} & \ddots & \ddots & h_0 \\ & h_{k-1} & & h_1 \\ & & \ddots & \vdots \\ & & & h_{k-1} \end{bmatrix}$$

SPIRAL transform notation

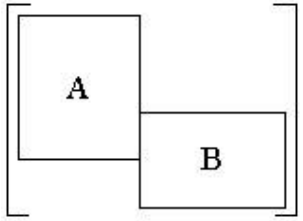
Baseline Rule

$$F_n(\mathbf{h}) = I_n \otimes_{k-1} \mathbf{h}^T$$

SPIRAL rule
captures the structure

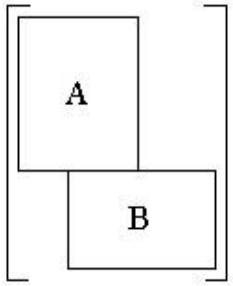
New constructs

$A \oplus^k B$

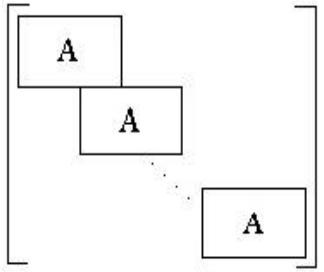


column and row overlapped direct sum

$A \oplus_k B$



$I_s \otimes^k A$



overlapped tensor product

Time Domain Methods

Overlap-Add Rule

$$F_n(\mathbf{h}) = I_{n/b} \otimes_{k-1} F_b(\mathbf{h})$$

concise rule notation

- Localized access of the input data
- Input is blocked into length b segments

Carnegie Mellon

SPIRAL

Overlap-Save Rule

$$F_n \left[\mathbf{h} \right] \; T \left[\mathbf{h}_L \right] \; \sum_{k=1} \; I_{m/b} \otimes \; _{k=1} F_b^T \left[\mathbf{h} \right] \; \sum_{k=1} \; T \left[\mathbf{h}_R \right]$$

Toeplitz matrix

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

Blocking Rule

$$F_n \left[\mathbf{h} \right] \; I_{n/b} \otimes \; _{v} \sum_{i=1}^{k/b} \; ^{b} T \left[\mathbf{h}_i \right]$$

- Multilevel blocking
- Control over input & output locality

Frequency Domain Methods

Expanded Circulant Rule

$$F_n \left[\mathbf{h} \right] \; C \left[E_{k,n-1} \mathbf{h} \right] E_{n,k-1} \quad E_{n,k} \; \begin{bmatrix} I_n \\ 0_{k \times n} \end{bmatrix}$$

Cyclic-RDFT Rule

$$C \left[\mathbf{a} \right] \; \text{RDFT}_n^{-1} \; X \left[\text{RDFT}_n \mathbf{a} \right] \text{RDFT}_n$$

Circulant matrix **sparse matrix** **real DFT**

Radix-2 Cooley-Tukey RDFT Rule

$$\text{RDFT}_n \; \left[\text{RDFT}_{n/2} \; \otimes \; I_2 \right] X(w^{kl}) \left[I_{n/2} \; \otimes \; \text{RDFT}_2 \right] L_2^n$$

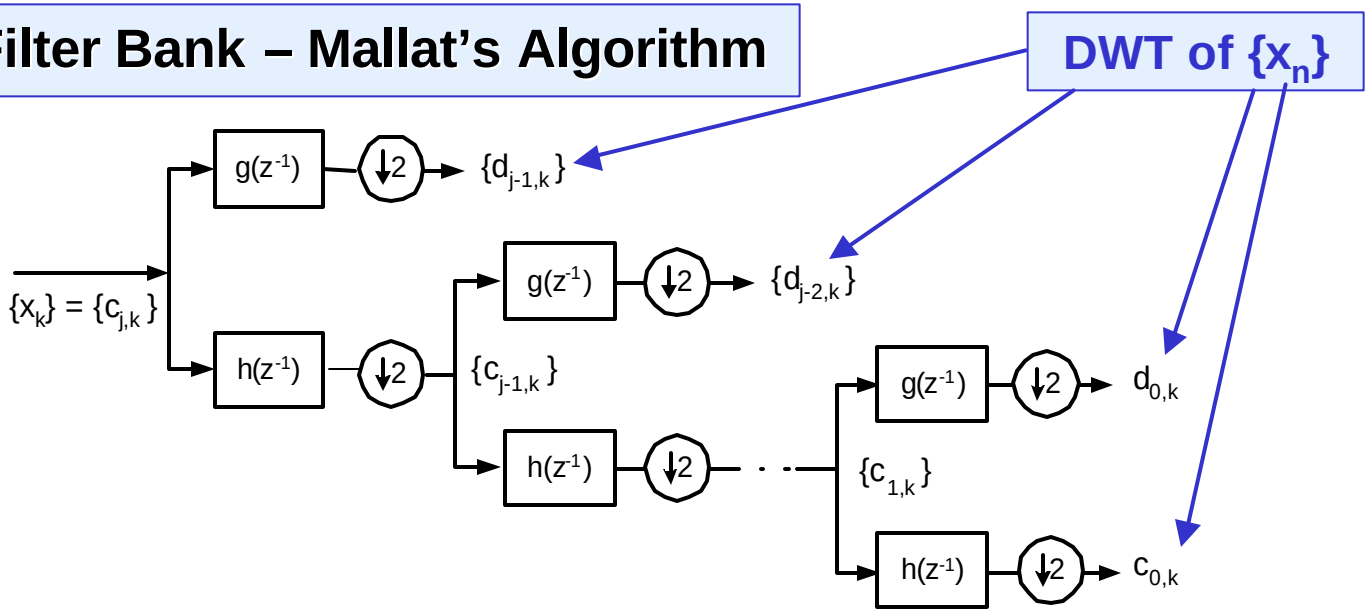
Time domain - better data access structure
Frequency domain - potentially reduce arithmetic cost

Carnegie Mellon

SPIRAL

DWT: Rules and Algorithms

Filter Bank – Mallat's Algorithm



Mallat Rule

Extension matrix

$$\mathrm{DWT}_n(\mathbf{h},\mathbf{g}) \to \begin{bmatrix} \mathrm{DWT}_{n/2}(\mathbf{h},\mathbf{g}) \to \mathrm{I}_{n/2} \otimes \mathbf{2} \otimes F_{n/2}(\mathbf{h}) \to \\ \to \mathrm{I}_{n/2} \otimes \mathbf{2} \otimes F_{n/2}(\mathbf{g}) \end{bmatrix} \mathbf{E}^*_{n,l,r}$$

Fused Mallat Rule

$$\mathrm{DWT}_n(\mathbf{h},\mathbf{g}) \to \begin{bmatrix} \mathrm{DWT}_{n/2}(\mathbf{h},\mathbf{g}) \to \mathrm{I}_{n/2} \otimes H_n(\mathbf{h},\mathbf{g}) \end{bmatrix} \mathbf{E}^*_{n,l,r}$$

- **Redundant computations removed**
- **Filter structure lost**

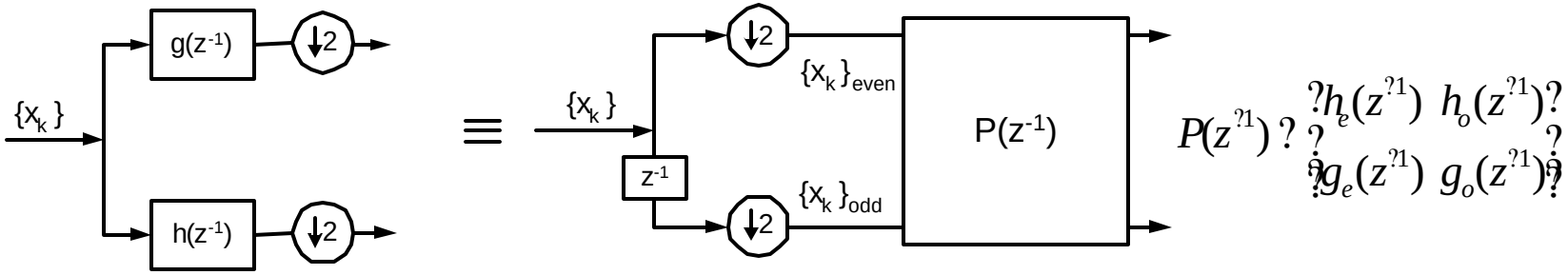
$$H_n(\mathbf{h},\mathbf{g}) \to \begin{bmatrix} \mathrm{I}_{n/2} \otimes_{k\to 2} \mathbf{h}^T \\ \mathrm{I}_{n/2} \otimes_{k\to 2} \mathbf{g}^T \end{bmatrix}$$

single-level DWT

Carnegie Mellon

SPIRAL

Polyphase Representation



$$P(z^{-1}) = \begin{bmatrix} h_e(z^{-1}) & h_o(z^{-1}) \\ g_e(z^{-1}) & g_o(z^{-1}) \end{bmatrix}$$

Polyphase Rule

Gateway to frequency domain computation

$$H_n(\mathbf{h},\mathbf{g}) = \begin{bmatrix} F_{n/2}^T(\mathbf{h}_o) & F_{n/2}^T(\mathbf{h}_e) \\ F_{n/2}^T(\mathbf{g}_o) & F_{n/2}^T(\mathbf{g}_e) \end{bmatrix} L_2^{n,k,2}$$

Lifting Scheme

Primal (Predict) **Dual (Update)**

Lifting steps

$$\begin{bmatrix} h_e(z^{-1}) & h_o(z^{-1}) \\ g_e(z^{-1}) & g_o(z^{-1}) \end{bmatrix} = \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix} \begin{bmatrix} 1 & s_i(z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ t_i(z^{-1}) & 1 \end{bmatrix} \cdots \begin{bmatrix} 1 & s_0(z^{-1}) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ t_0(z^{-1}) & 1 \end{bmatrix}$$

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

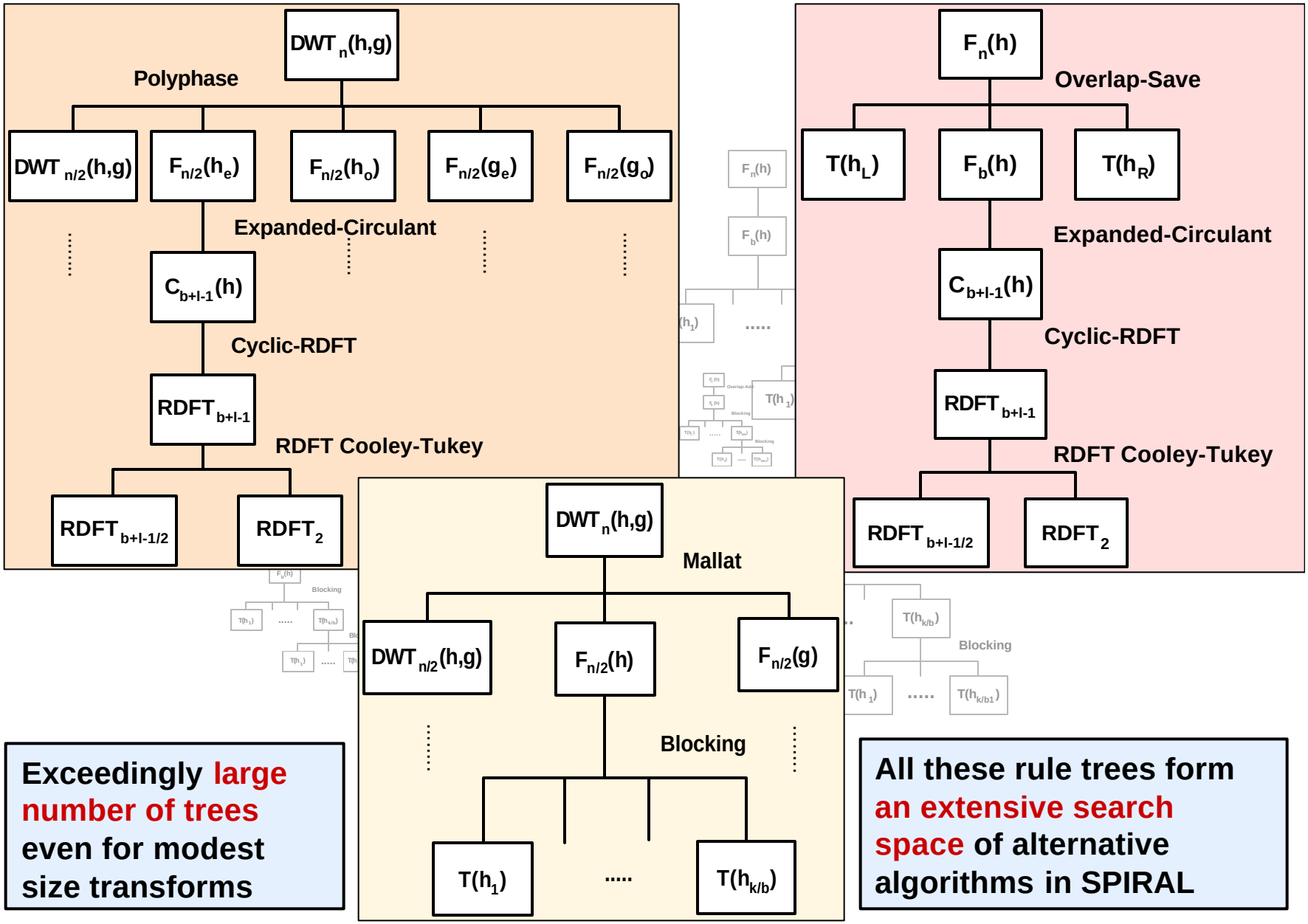
Lifting Rule

$$H_n(\mathbf{h},\mathbf{g}) = \begin{bmatrix} K I_{n/2} & 0 \\ 0 & 1/K I_{n/2} \end{bmatrix} \begin{bmatrix} I_{n/2} & F_{n/2}^T(\mathbf{s}_i) \\ 0 & I_{n/2} \end{bmatrix} \cdots \begin{bmatrix} I_{n/2} & 0 \\ F_{n/2}^T(\mathbf{t}_o) & I_{n/2} \end{bmatrix} L_2^{n,k,2}$$

Carnegie Mellon

SPIRAL

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

Carnegie Mellon

SPIRAL

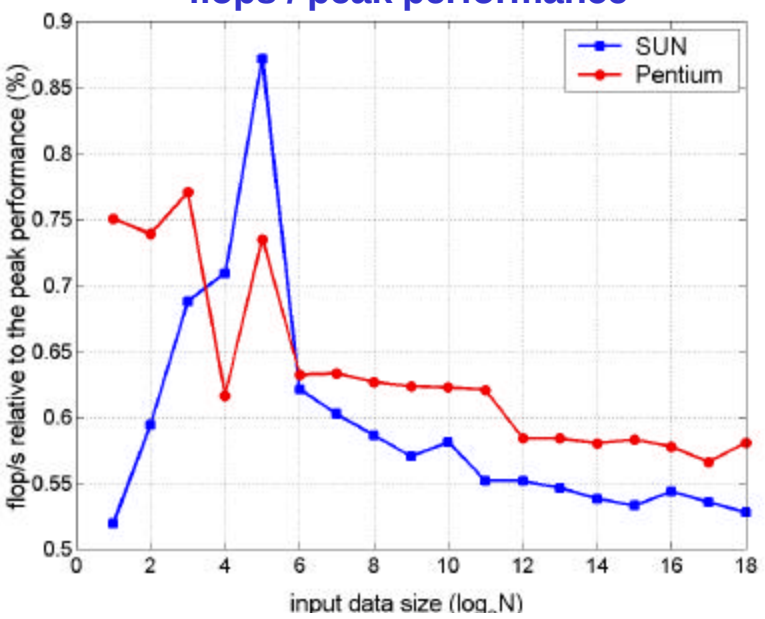
Time-Domain Methods

FIR Filter

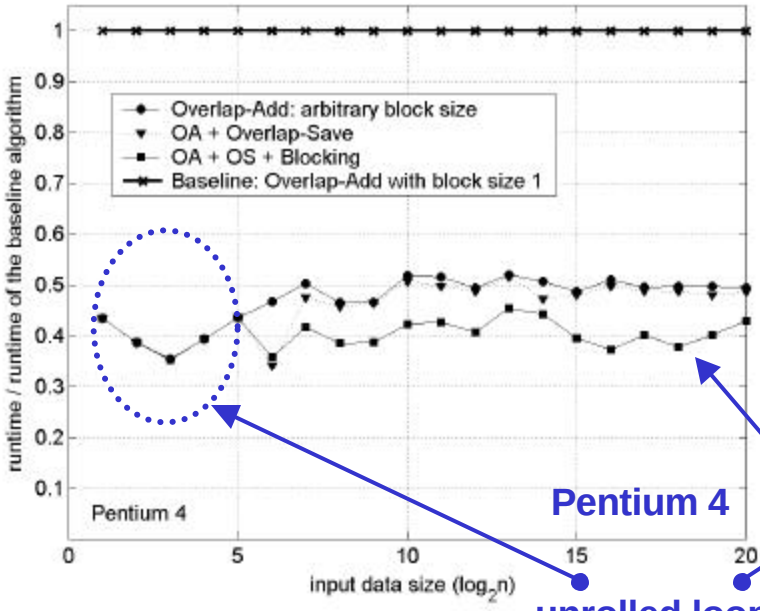
- overlap-add, overlap-save, and blocking
- all methods have the same cost
- baseline method = overlap-add with *b*=1
- sanity check – percentage of the peak performance

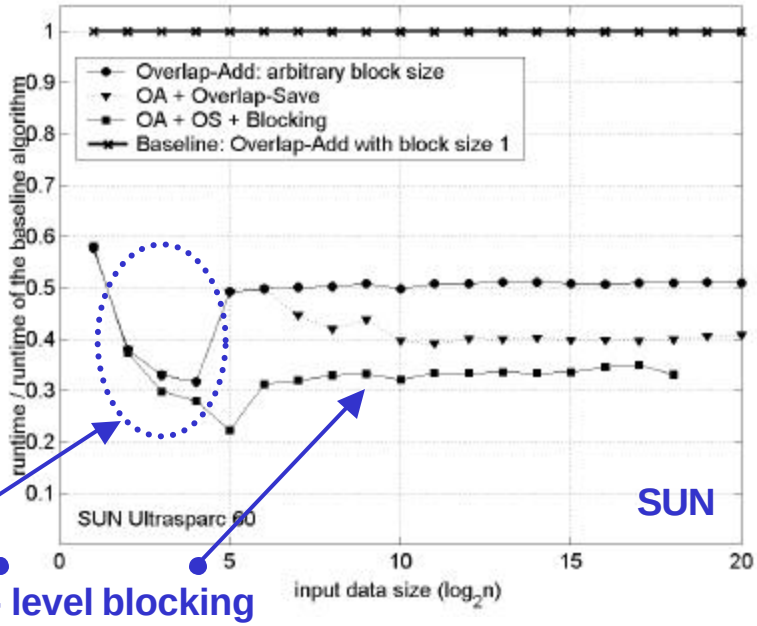
60-70% improvement over baseline
Same arithmetic cost

flops / peak performance



comparison of different time-domain methods





Pentium 4

SUN

unrolled loops

1- level blocking
blocks size 2

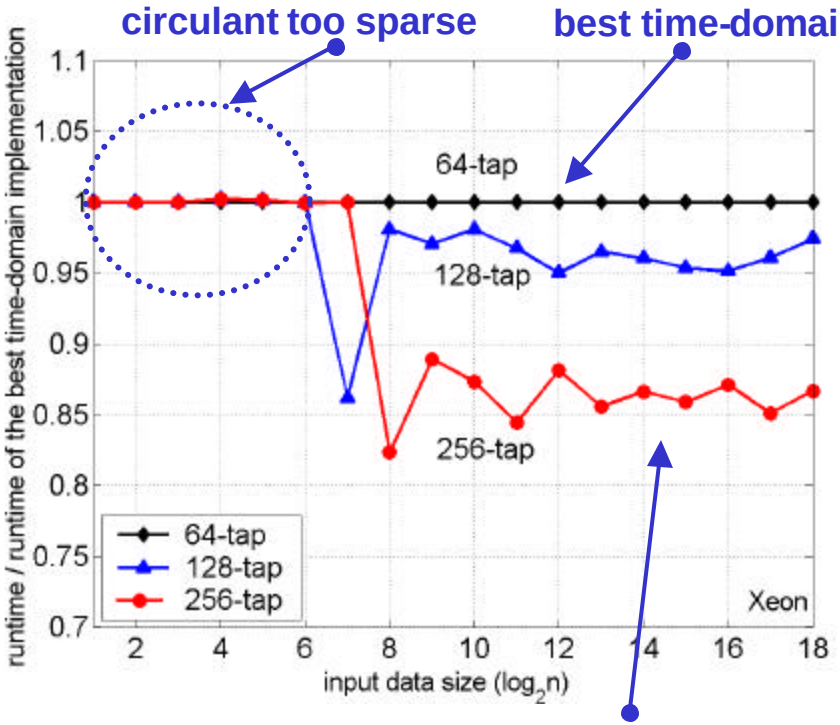
Carnegie Mellon

SPIRAL

Frequency Domain vs. Time Domain

FIR Filter

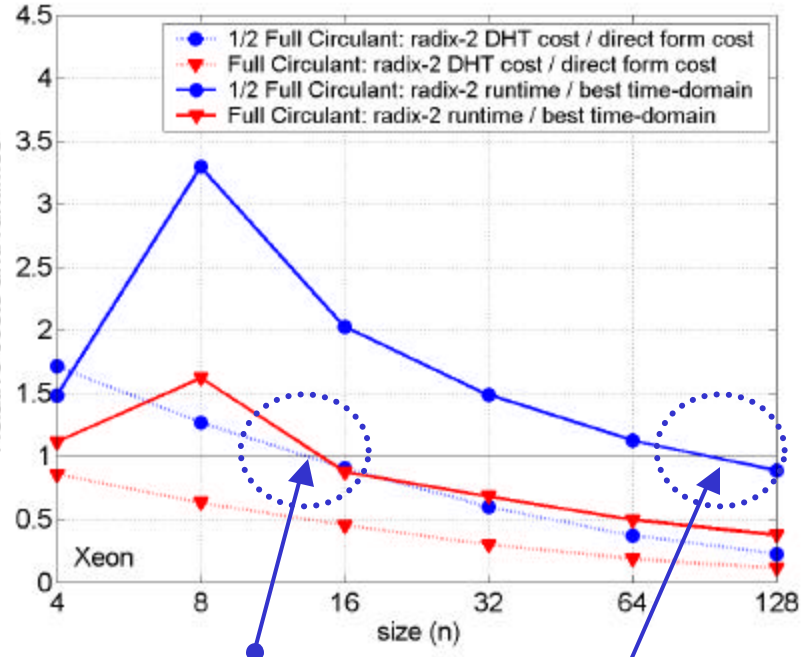
Frequency domain – potentially lower cost
Time domain – beter data access patterns



FD methods better on large "fat" circulants
Filter lengths > 64

Circulant

"half–dense" circulants emerge in frequency domain methods



arithmetic cost cross-over

runtime cross-over

Considerable discrepancy between arithmetic costs and actual runtimes

Carnegie Mellon

SPIRAL

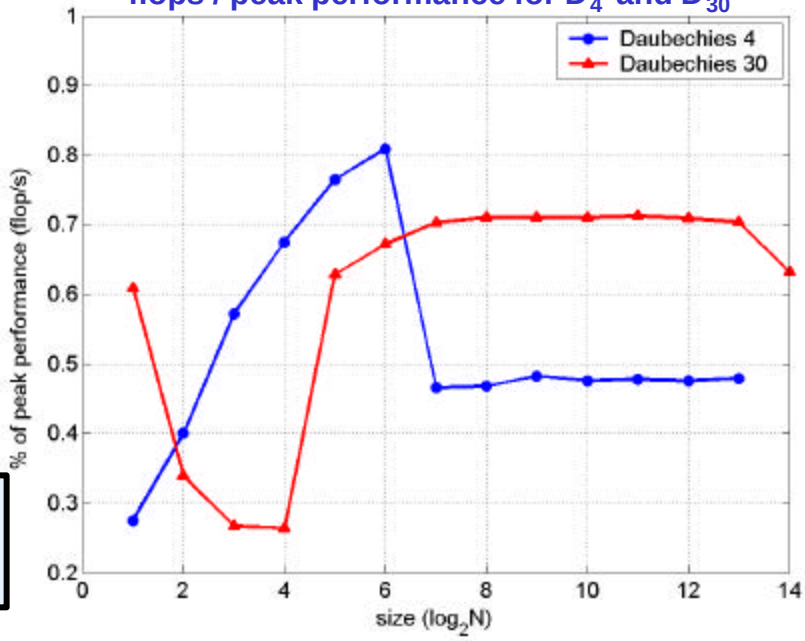
Daubechies D_4 DWT

Short Wavelet

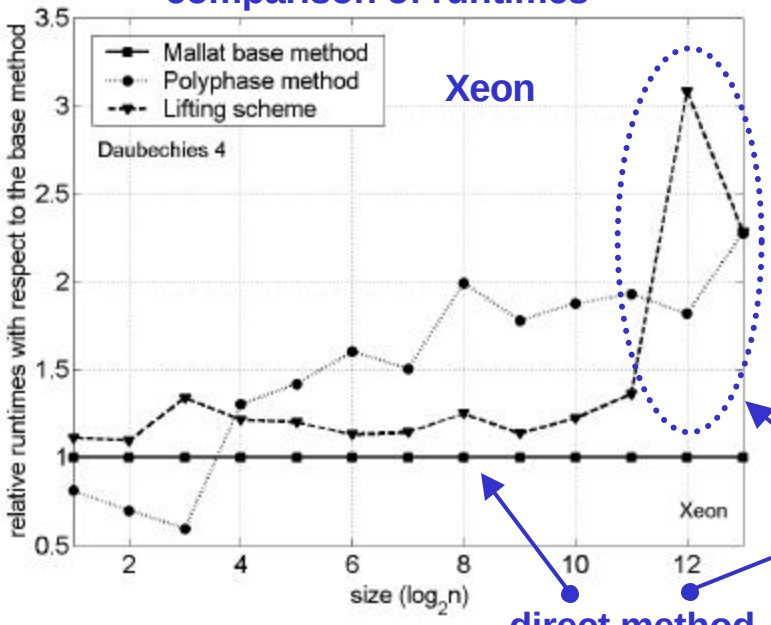
- single-level DWT using D_4
- 3 different classes of rule trees based on the initial rule
  - **Fused Mallat Rule** (direct method)
  - **Polyphase Rule** (frequency domain)
  - **Lifting Rule** (lifting scheme)

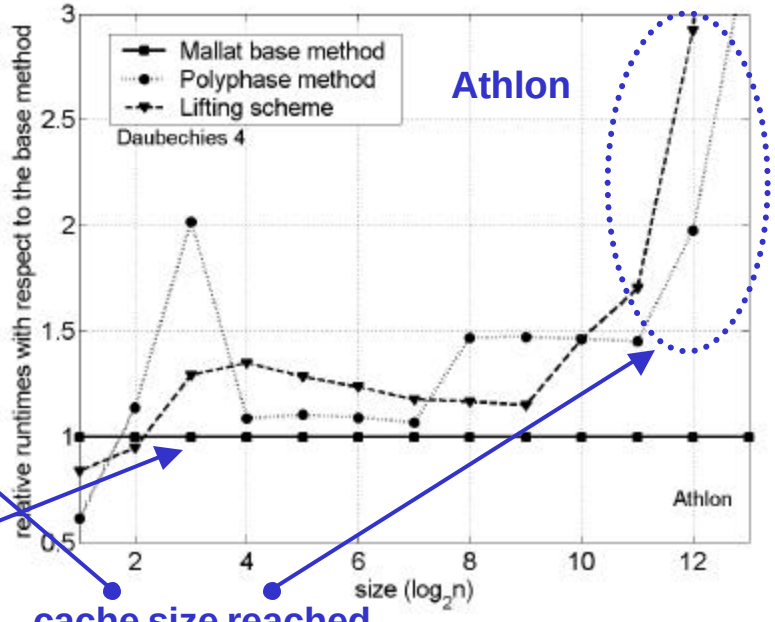
Frequency domain trees not found
Lifting scheme – 30% lower cost – slower

flops / peak performance for D_4 and D_{30}



comparison of runtimes





direct method

cache size reached

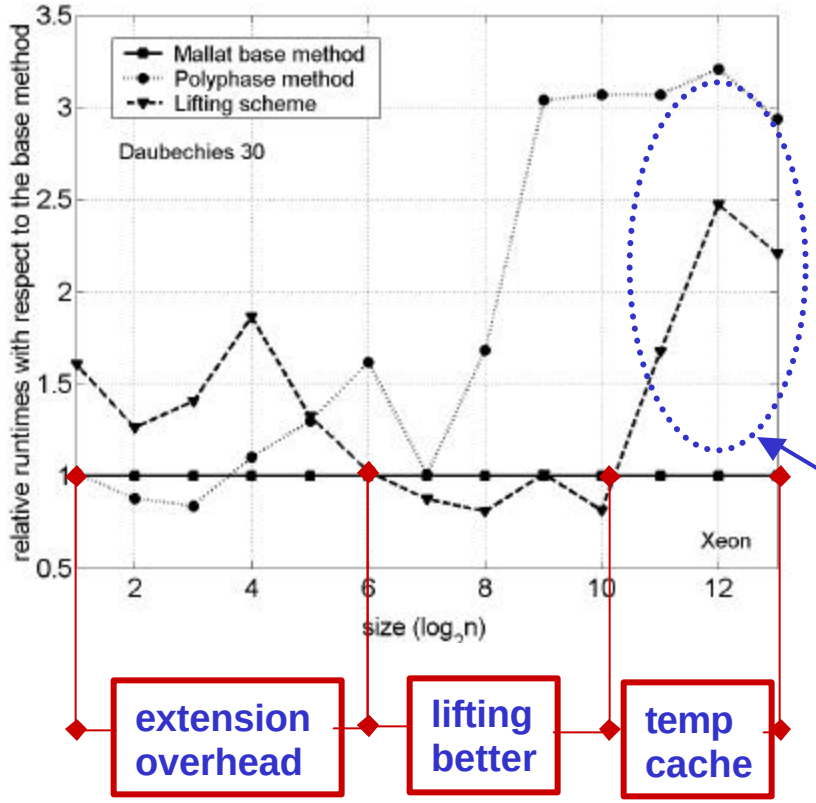
Carnegie Mellon

SPIRAL

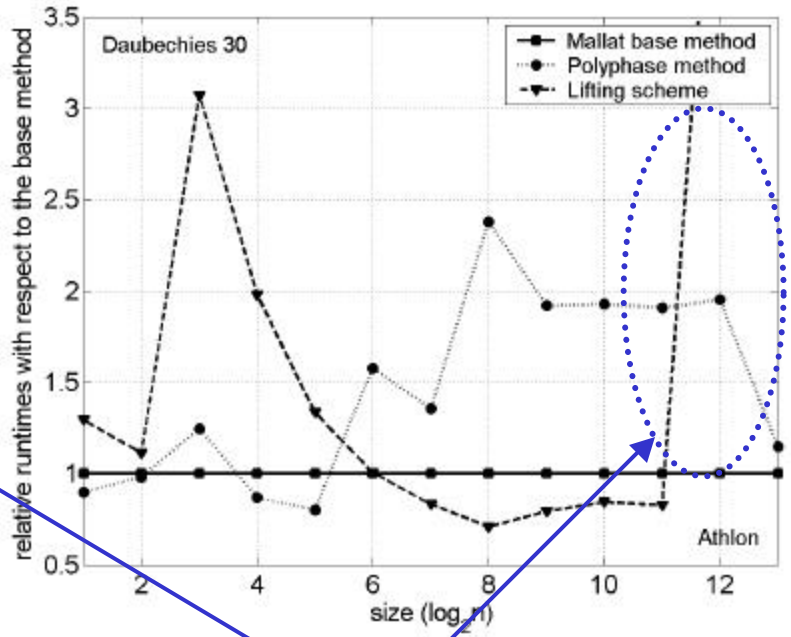
Daubechies D_{30} DWT Experiment

Long Wavelet

Xeon



Athlon



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

Carnegie Mellon

SPIRAL

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)