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

Carnegie Mellon

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

Electrical & Computer ENGINEERING

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

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

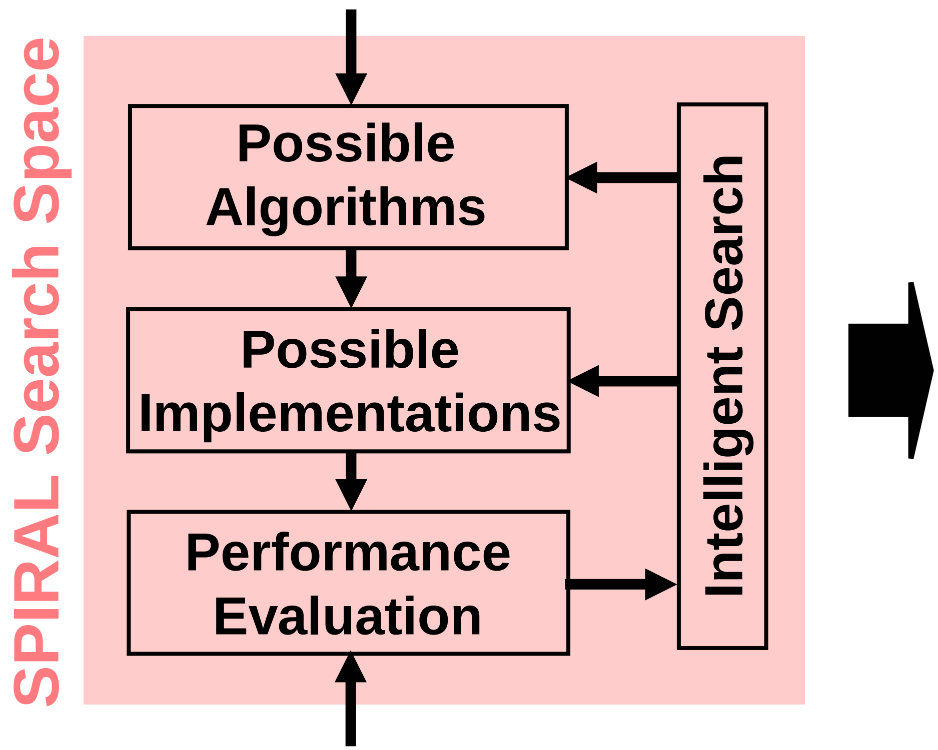
A library generator for highly optimized signal processing algorithms

SPIRAL Approach

Carnegie Mellon

given → **DSP Transform**

(DFT, DCT, Wavelets etc.)



→ **adapted implementation**

given → **Computing Platform**

(Pentium III, Pentium 4, Athlon, SUN, PowerPC, Alpha, …)

SPIRAL

Electrical & Computer ENGINEERING

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):

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

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

Carnegie Mellon

SPIRAL

Electrical & Computer ENGINEERING

DSP Algorithms: Terminology

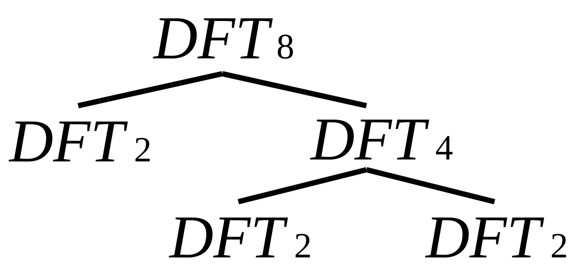
Transform DFT_n **parameterized matrix**

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



- 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 $\Rightarrow$ 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**

$$B \otimes A = L_n^{mn}\left(A \otimes B\right) L_m^{mn}$$

A* and *B* square size *m* and *n

- **Commutation property $\Rightarrow$ further variations**

$$F_N = L_S^{RS}\left(I_S \otimes F_R\right) L_R^{RS} T_S^{RS}\left(I_R \otimes F_S\right) L_R^{RS}$$

$$F_N = \left(F_R \otimes I_S\right) T_S^{RS} L_S^{RS}\left(F_S \otimes I_R\right)$$

$$\left(F_2 \otimes I_2\right) x = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \hline x_2 \\ x_3 \end{bmatrix} \quad \left(I_2 \otimes F_2\right) = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \hline x_2 \\ x_3 \end{bmatrix}$$

- **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)

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

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

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

modified discrete cosine transform

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

two-dimensional transform

$$T \otimes T$$

discrete wavelet transform

$$\mathrm{DTWT}_{2^n} = \left(\mathrm{DTWT}_{2^{n-1}} \oplus \mathrm{I}_{2^{n-1}}\right) \underbrace{\mathrm{L}^{2^n}_{2^{n-1}} \left(\mathrm{I}_{2^{n-1}} \otimes_{l-2} W\right)}_{H}$$

Others: filtering, Haar, Hartley, …

Rules = Breakdown Strategies

$$DCT_2^{(II)} \rightarrow diag\left(1, 1/\sqrt{2}\right) \cdot F_2$$
base case

$$DCT_n^{(II)} \rightarrow P \cdot \left(DCT_{n/2}^{(II)} \oplus DCT_{n/2}^{(IV)}\right) \cdot \left(I_{n/2} \otimes F_2\right)^Q$$
recursive

$$DCT_n^{(IV)} \rightarrow S \cdot DCT_n^{(II)} \cdot D$$
translation

$$DCT_n^{(IV)} \rightarrow M_1 \cdots M_r$$
iterative

$$DFT_n \rightarrow CosDFT_n + j \cdot SinDFT_n$$
recursive

$$DFT_n \rightarrow B \cdot (DCT_{n/2}^{(I)} \oplus DST_{n/2}^{(I)}) \cdot C$$
recursive

$$DFT_{nm} \rightarrow (DFT_n \otimes I_m) \cdot D \cdot (I_n \otimes DFT_m) \cdot P$$
recursive

$$CosDFT_n \rightarrow \cdots CosDFT_{n/2} \cdots DCT_{n/4}^{(II)} \cdots$$
recursive

$$SinDFT_n \rightarrow \cdots SinDFT_{n/2} \cdots DCT_{n/4}^{(II)} \cdots$$
recursive

$$WHT_{2^n} \rightarrow \prod_{i=1}^{n} (I_{2^{n_1+\ldots+n_{i-1}}} \otimes WHT_{2^{n_i}} \otimes I_{2^{n_{i+1}+\ldots+n_t}})$$
iterative/ recursive

$$MDCT_{n \times 2n} \rightarrow S \cdot DCT_n^{(IV)} \cdot P$$
translation

$$\mathrm{DTWT}_{2^n} = \left(\mathrm{DTWT}_{2^{n-1}} \oplus \mathrm{I}_{2^{n-1}}\right) \underbrace{\mathrm{L}_{2^{n-1}}^{2^n}\left(\mathrm{I}_{2^{n-1}} \otimes_{l-2} W\right)}_{H}$$

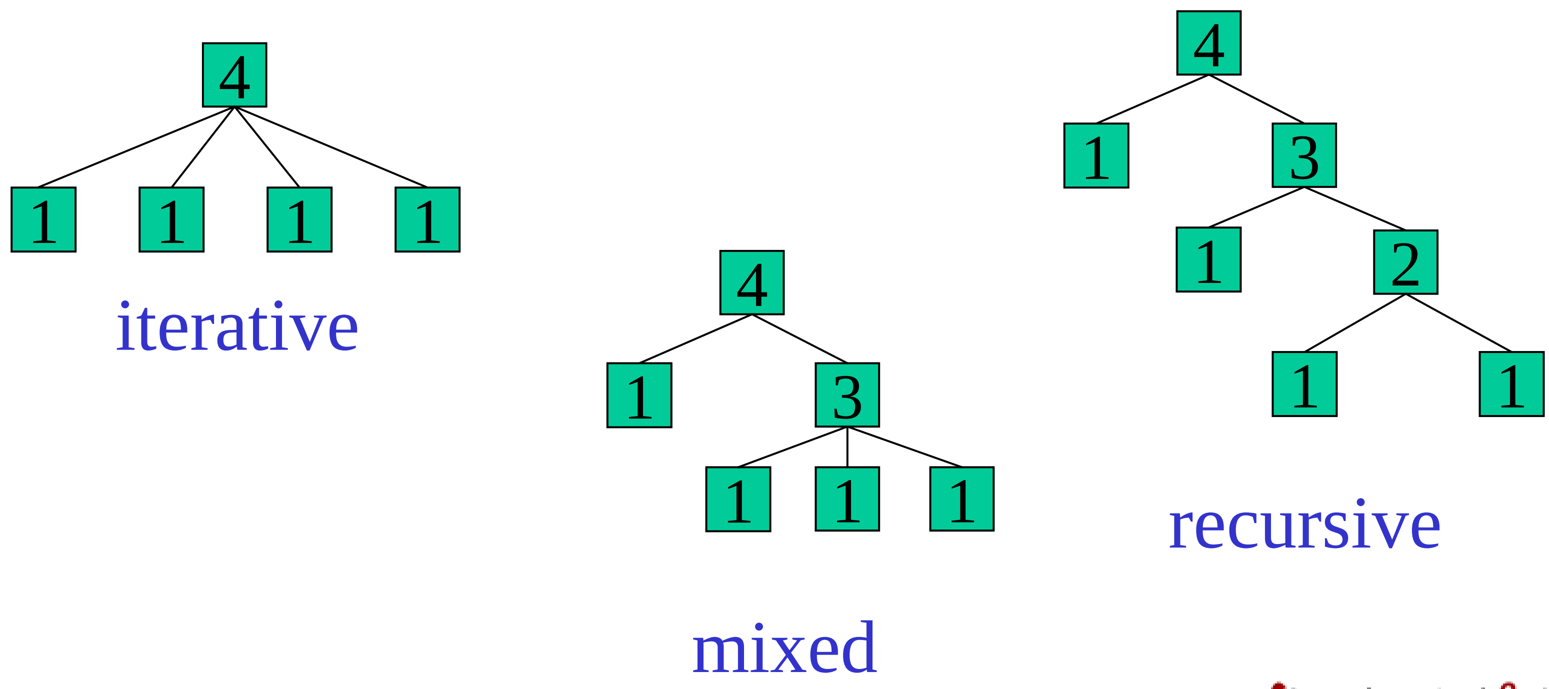
Carnegie Mellon

SPIRAL

Electrical & Computer ENGINEERING

Partition Trees

Each WHT algorithm can be represented by a tree, where a node labeled by n corresponds to WHT_{2^n}

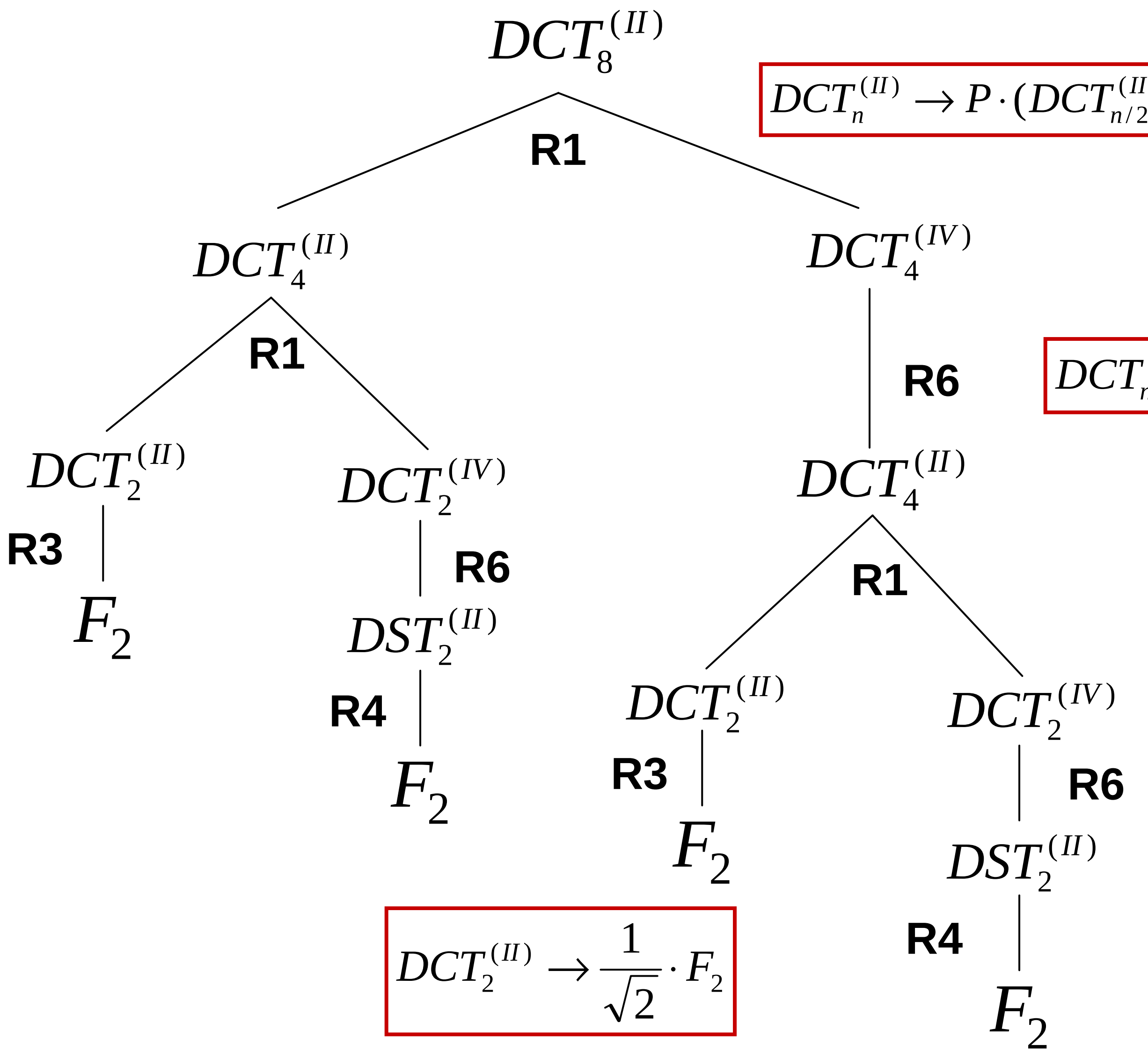


Carnegie Mellon

SPIRAL

Electrical & Computer ENGINEERING

Algorithms = Ruletrees = Formulas



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

$$\boxed{DCT_n^{(IV)} \rightarrow P \cdot DCT_n^{(II)} \cdot S}$$

$$\boxed{DCT_2^{(II)} \rightarrow \frac{1}{\sqrt{2}} \cdot F_2}$$

Carnegie Mellon

SPIRAL

Electrical & Computer ENGINEERING

Mathematical Framework for Filters and Discrete-Time Wavelet Transforms

- **Overlapped direct sum of matrices A (*m* x *n*) and B (*r* x *s*)**

column overlap $$A \oplus_v B = \begin{bmatrix} \boxed{A} & \\ & \overset{\overbrace{\ }^{v}}{\boxed{B}} \end{bmatrix} = (A \oplus B)(I_n \oplus_v I_s)$$

row overlap $$A \oplus^v B = \begin{bmatrix} \boxed{A} & \}^{v}\,\boxed{B} \end{bmatrix} = (I_m \oplus^v I_r)(A \oplus B)$$

- **Overlapped tensor product**

column overlap $$I_k \otimes_v B = \underbrace{B \oplus_v B \oplus_v \cdots \oplus_v B}_{k-fold}$$

row overlap $$I_k \otimes^v B = \underbrace{B \oplus^v B \oplus^v \cdots \oplus^v B}_{k-fold}$$

SPIRAL

Electrical & Computer ENGINEERING

Properties of Overlapped Direct Sums and Tensor Products

- **Column and row overlap direct sum**

$$A \oplus^{v} B = \begin{bmatrix} \boxed{A} & \\ & \boxed{B} \end{bmatrix} = \begin{bmatrix} \boxed{A^{T}} & \\ & \boxed{B^{T}} \end{bmatrix}^{T} = \left(A^{T} \oplus_{v} B^{T}\right)^{T}$$

$$\mathbf{I}_n \oplus^{v} \mathbf{I}_s = \left(\mathbf{I}_n \oplus_{v} \mathbf{I}_s\right)^{T}$$

- **Column and row overlapped tensor product**

$$I_k \otimes_v B_{m \times n} = \left(\bigoplus_{l=1}^{k} B_{m \times n}\right) \bullet \left(\bigoplus_{l=1}^{k}{}_{v}\, I_n\right) = \left(I_k \otimes B_{m \times n}\right) \bullet \left(I_k \otimes_v I_n\right)$$

$$I_k \otimes^{v} B_{m \times n} = \left(\bigoplus_{l=1}^{k}{}_{v}\, I_m\right)^{T} \bullet \left(\bigoplus_{l=1}^{k} \left(B_{m \times n}\right)^{T}\right)^{T} = \left(I_k \otimes_v I_m\right)^{T} \bullet \bigoplus_{l=1}^{k} B_{m \times n}$$

$$= \left(I_k \otimes^{v} I_m\right) \bullet \left(I_k \otimes B_{m \times n}\right)$$

Filters

- **Linear convolution in matrix form**

$$C_n(\underline{h}) = \mathrm{I}_n \otimes^{l-1} \begin{bmatrix} h_0 & h_1 & \cdots & h_{l-1} \end{bmatrix}^{\mathrm{T}}$$

- **Block convolutions**
  - **Overlap-save rule**

$$C_n(\underline{h}) = \left(\mathrm{I}_{n/b} \otimes^{l-1} \mathrm{I}_{b+l-1}\right)\left(\mathrm{I}_{n/b} \otimes C_b(\underline{h})\right)$$

  - **Overlap-add rule**

$$C_n(\underline{h}) = \left(\mathrm{I}_{m/b} \otimes C_b^{\mathrm{T}}(\underline{h})\right)\left(\mathrm{I}_{m/b} \otimes_{l-1} \mathrm{I}_{b+l-1}\right)E_{l-1,l-1} \qquad m=n+l-1$$

- **Circular convolution rule for circulant *K***

$$K(\underline{h}) = \mathrm{DFT}_n^{-1} \cdot \mathrm{diag}\left(\mathrm{DFT}_n \cdot \underline{h}\right) \cdot \mathrm{DFT}_n$$

$\underline{h}$-first column of K

n-length of $\underline{h}$

Carnegie Mellon

SPIRAL

Electrical & Computer ENGINEERING

Discrete-Time Wavelet Transform

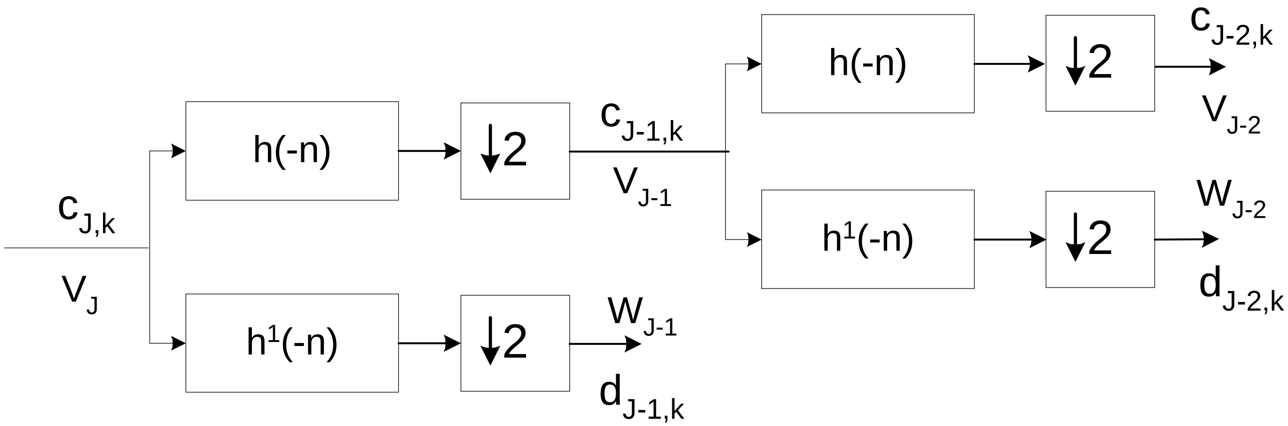
- **DTWT scaling and wavelet expansion coefficients**

$$c_{0,k} = \sum_n x_n \, h_J\left(2^J k - n\right) \qquad \text{scaling coefficients}$$

$$d_{j,k} = \sum_n x_n \, h^1_{J-j}\left(2^{J-j} k - n\right) \quad j = 0, \ldots, J-1 \qquad \text{wavelet coefficients}$$

- **Recursive algorithm (Mallat)**

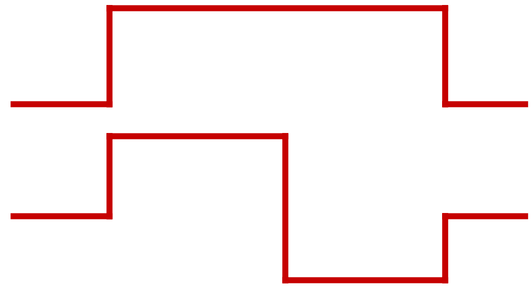
$$c_{j,k} = \sum_m h_{m-2k} c_{j+1,m} \qquad d_{j,k} = \sum_m h^1_{m-2k} c_{j+1,m}$$



Haar Wavelets – Example

- **Haar wavelets = square waves**

$$h = \frac{1}{\sqrt{2}}[\,1,\ 1],\quad h^1 = \frac{1}{\sqrt{2}}[\,1, -1]$$



- **First stage: $V_2 \Rightarrow V_1 \oplus W_1$**

$$H = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} = L^4_2 \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} = L^4_2\left(I_2 \otimes F_2\right) \qquad \begin{bmatrix} \underline{c}_1 \\ \underline{d}_1 \end{bmatrix} = H\,\underline{c}_2$$

- **The process is repeated for the upper half of the output**

$$HT_{2^n} = \left(HT_{2^{n-1}} \oplus I_{2^{n-1}}\right) \underbrace{L^{2^n}_{2^{n-1}}\left(I_{2^{n-1}} \otimes F_2\right)}_{H},\quad HT_2 = F_2$$

Discrete-Time Wavelet Transform

- **Discrete-Time Wavelet Transform (DTWT) rule**

$$\mathrm{DTWT}_{2^n} = \left(\mathrm{DTWT}_{2^{n-1}} \oplus \mathrm{I}_{2^{n-1}}\right) \underbrace{\mathrm{L}^{2^n}_{2^{n-1}}\left(\mathrm{I}_{2^{n-1}} \otimes_{l-2} W\right)}_{H}$$

- **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**

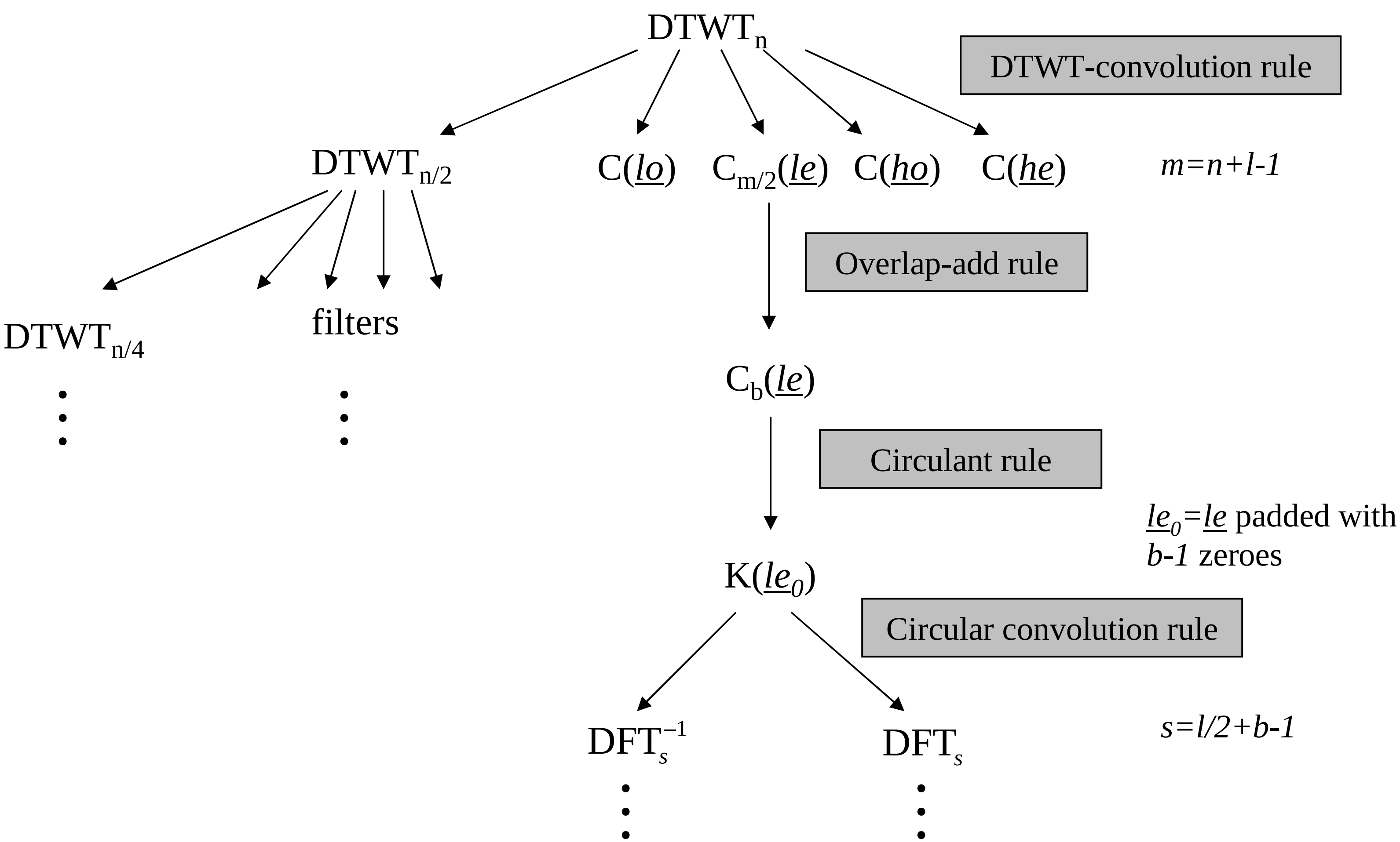
$$H = \Big(\big([1 \quad 1] \otimes \mathrm{I}_{m/2} \big) \cdot \big(C^{\mathrm{T}}_{m/2}(\underline{lo}) \oplus C^{\mathrm{T}}_{m/2}(\underline{le}) \big) \cdot \mathrm{L}^n_2 \oplus$$
$$\big([1 \quad 1] \otimes \mathrm{I}_{m/2} \big) \cdot \big(C^{\mathrm{T}}_{m/2}(\underline{ho}) \oplus C^{\mathrm{T}}_{m/2}(\underline{he}) \big) \cdot \mathrm{L}^n_2 \Big) \cdot \left(\begin{bmatrix} 1 \\ 1 \end{bmatrix} \otimes \mathrm{I}_n \right)$$

$\underline{lo}$-lowpass odd coeffs., $\underline{le}$-lowpass even coeffs.

$\underline{ho}$-highpass odd coeffs., $\underline{he}$-highpass even coeffs.

Carnegie Mellon

DTWT Ruletree – Example



Formula for a DCT, size 16

$$
\begin{aligned}
&[(2,16,9,5,3)(4,15,8,13,7)(6,14,10,12,11),16]\cdot\\
&(([(2,8,5,3)(4,7),8]\cdot(([(2,4,3),4]\cdot((\mathrm{diag}(1,\sqrt{\tfrac{1}{2}})\cdot\mathrm{DFT}_2)\oplus([(1,2),2]\cdot\mathrm{R}_{\frac{13}{8}\pi})^{[(1,2),2]})\cdot\\
&(\mathbf{1}_2\otimes\mathrm{DFT}_2)^{[(2,4,3),4]})\oplus(\mathrm{diag}(\tfrac{1}{2\cos(\frac{1}{16}\pi)},\tfrac{1}{2\cos(\frac{3}{16}\pi)},\tfrac{1}{2\cos(\frac{5}{16}\pi)},\tfrac{1}{2\cos(\frac{7}{16}\pi)})\cdot(\mathbf{1}_2\otimes\mathrm{DFT}_2)^{[(2,4,3),4]}\cdot\\
&((\mathrm{DFT}_2\cdot\mathrm{diag}(1,\sqrt{\tfrac{1}{2}}))\oplus([(1,2),2]\cdot\mathrm{R}_{\frac{13}{8}\pi})^{[(1,2),2]})\cdot[(2,3,4),4]\cdot\begin{bmatrix}1&0&0&0\\1&1&0&0\\0&1&1&0\\0&0&1&1\end{bmatrix})^{[(1,4)(2,3),4]})\cdot\\
&(\mathbf{1}_4\otimes\mathrm{DFT}_2)^{[(2,8,5,3)(4,7),8]})\oplus([(2,5,4,3,7,6,8),8]\cdot(\mathbf{1}_2\otimes(\mathbf{1}_2\oplus([(1,2),2]\cdot\mathrm{R}_{\frac{7}{4}\pi})))\cdot\\
&(\mathbf{1}_2\otimes\mathrm{DFT}_2\otimes\mathbf{1}_2)\cdot(\mathbf{1}_4\oplus([(1,2),2]\cdot\mathrm{R}_{\frac{13}{8}\pi})\oplus([(1,2),2]\cdot\mathrm{R}_{\frac{1}{8}\pi}))\cdot(\mathbf{1}_1\otimes\mathrm{DFT}_2\otimes\\
&\mathbf{1}_4)\cdot(([(1,2),2]\cdot\mathrm{R}_{\frac{49}{32}\pi})\oplus([(1,2),2]\cdot\mathrm{R}_{\frac{53}{32}\pi})\oplus([(1,2),2]\cdot\mathrm{R}_{\frac{57}{32}\pi})\oplus([(1,2),2]\cdot\\
&\mathrm{R}_{\frac{61}{32}\pi}))\cdot[(2,8)(4,6),8])^{[(1,8)(2,7)(3,6)(4,5),8]})\cdot\\
&(\mathbf{1}_8\otimes\mathrm{DFT}_2)^{[(2,16,9,5,3)(4,15,8,13,7)(6,14,10,12,11),16]}
\end{aligned}
$$

Helpful Concept

DSP Transforms		Formal Languages
DSP transform (of size)	⟷	Non-terminal symbol (with attribute)
Rule	⟷	Rule (production)
Formula/Algorithm	⟷	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 $\rightarrow$ Algorithm

- SPL and SPL Compiler

  Algorithm $\rightarrow$ Implementation

- Search Engine

  How to find the best implementation

- SPIRAL system

  Everything taken together

- Conclusions

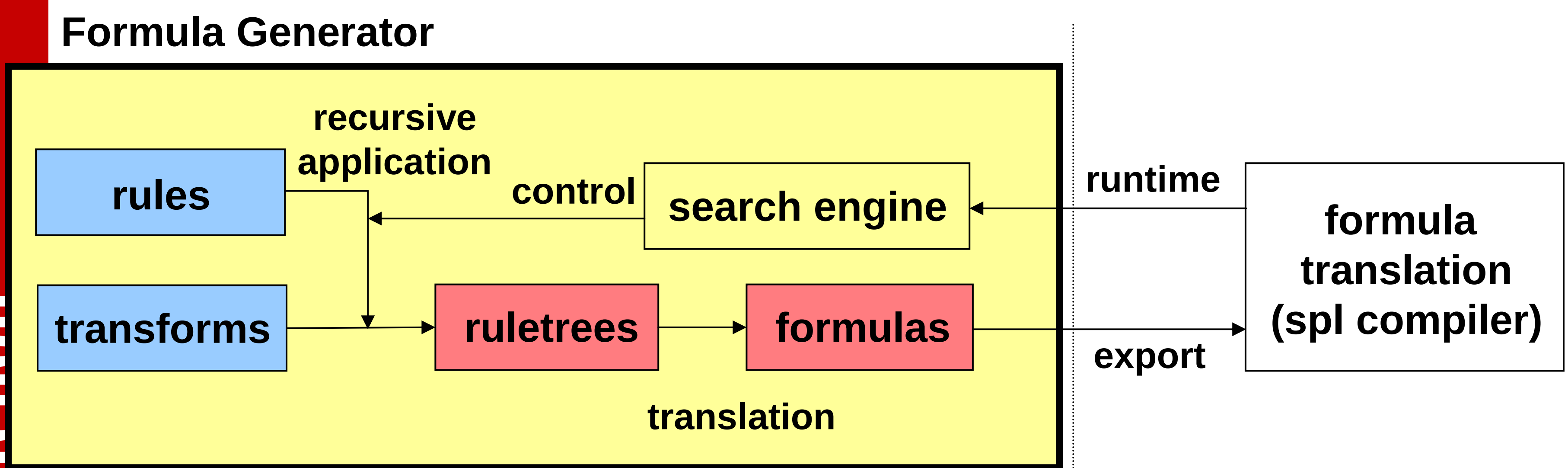
Carnegie Mellon

SPIRAL

Electrical & Computer ENGINEERING

Formula Generation

data base (extensible!)
data type



cut here for other optimization problems

- 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

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

Carnegie Mellon

SPIRAL

Electrical & Computer ENGINEERING

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: ← **allows extension of SPL**
```
(define name formula)
(template formula (i-code-list)
```
- directives for code generation
```
#codetype real/complex
#unroll on/off
```
← **controls loop unrolling**

Carnegie Mellon

SPIRAL

Electrical & Computer ENGINEERING

SPL Compiler, 4-point FFT

```
(compose (tensor (F 2) (I 2)) (T 4 2)
  (tensor (I 2) (F 2)) (L 4 2))
```

fast algorithm as **formula** as **SPL program**

#codetype

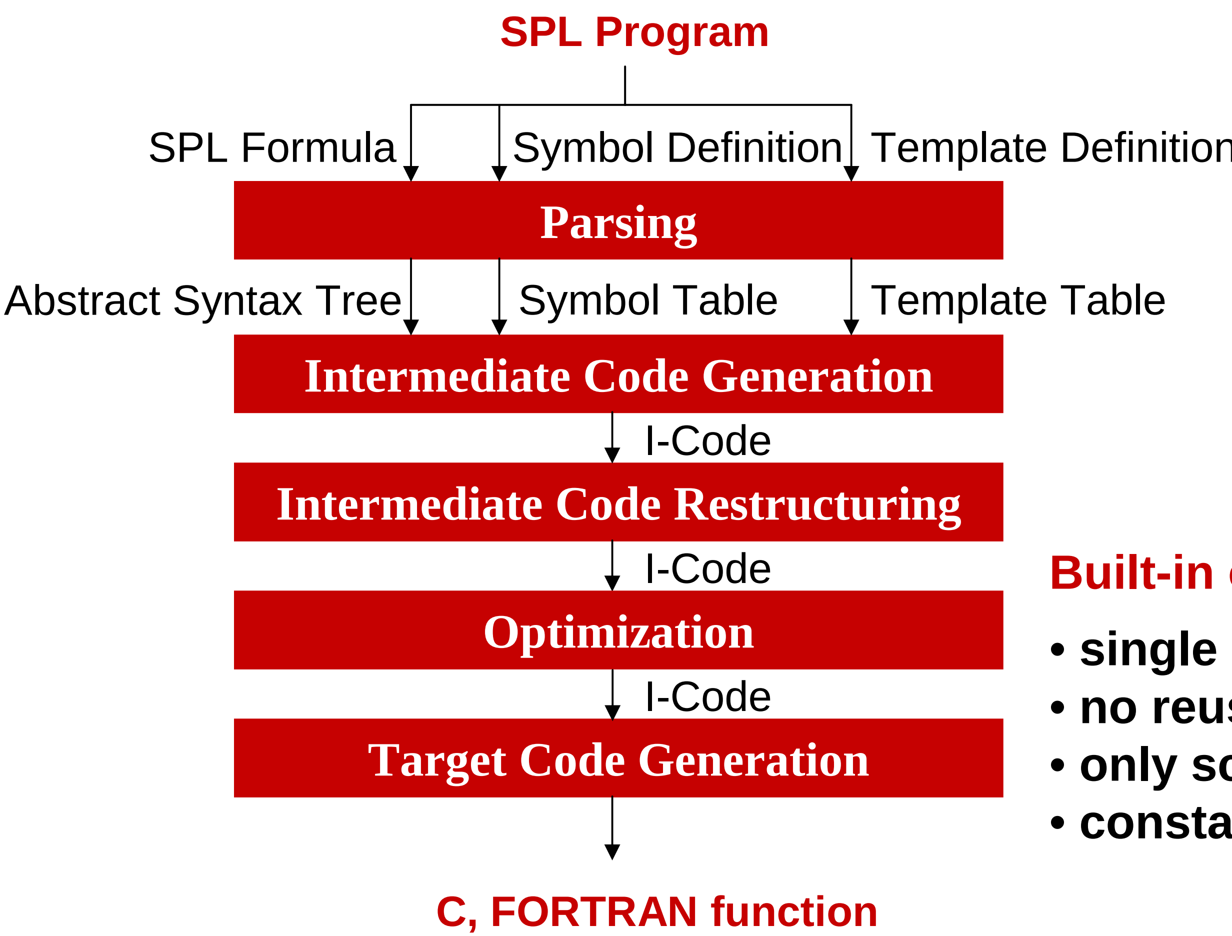
complex

real

```
f0   = x(1) + x(3)
f1   = x(1) - x(3)
f2   = x(2) + x(4)
f3   = x(2) - x(4)
f4   = (0.00d0,-1.00d0)*f(3)
y(1) = f0 + f2
y(2) = f0 - f2
y(3) = f1 + f4
y(4) = f1 - f4
```

```
r0   = x(1) + x(5)
r1   = x(1) - x(5)
r2   = x(2) + x(6)
r3   = x(2) - x(6)
r4   = x(3) + x(7)
r5   = x(3) - x(7)
r6   = x(4) + x(8)
r7   = x(4) - x(8)
y(1) = r0 + r4
y(2) = r1 + r5
y(3) = r0 - r4
y(4) = r1 - r5
y(5) = r2 + r7
y(6) = r3 - r6
y(7) = r2 - r7
y(8) = r3 + r6
```

SPL Compiler: Summary



Built-in optimizations:

- **single static assignment code**
- **no reuse of temporary vars**
- **only scalar temporary vars**
- **constants precomputed**

Extensible through templates

Templates

```
(template
  (F n)[ n >= 1 ]
  ( do i=0,n-1
      y(i)=0
      do j=0,n-1
        y(i)=y(i)+W(n,i*j)*x(j)
      end
    end ))
```

Pattern

Condition

I-code

Code Generation and Template Matching

(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:

```
do i = 0,1
  y(i) = 0
  do j = 0,1
    y(i) = y(i)+W(2,i*j)*x(j)
  end
end
```

Unrolling & Optimization

```
Y(0)=x(0)+x(1)
y(1)=x(0)-x(1)
```

Carnegie Mellon

SPIRAL

Electrical & Computer ENGINEERING

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:

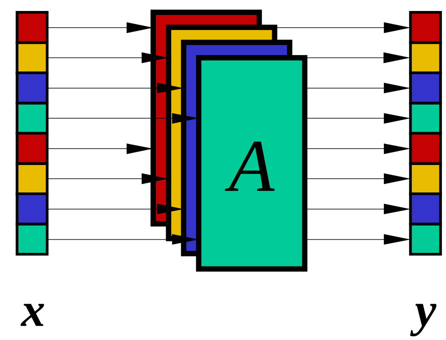
- SIMD instructions are architecture specific
- No common API (usually assembly hand coding)
- Performance very sensitive to memory access
- Automatic vectorization very limited

Vector Code Generation from SPL Formulas

Naturally vectorizable construct

$$A \otimes I_4$$

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**
ν	**SIMD vector length**

Vectorization in two steps:

1. **Formula manipulation using manipulation rules**
2. **Code generation (vector code + C code)**

SPL Compiler: Vector Code Generation

$$DFT_{16} = \left(DFT_4 \otimes I_4\right) \cdot T_4^{16} \cdot \left(I_4 \otimes DFT_4\right) \cdot L_4^{16}$$

Symbolic vectorization (automatic formula manipulation)

$$\overline{DFT}_{16} = \left(\left(I_4 \otimes L_4^8\right) \cdot \left(\overline{DFT}_4 \otimes I_4\right) \cdot \overline{T'}_4^{16}\right) \cdot \left(\left(I_4 \otimes L_2^8\right)\left(L_4^{16} \otimes I_2\right)\left(I_4 \otimes L_4^8\right) \cdot \left(\overline{DFT}_4 \otimes I_4\right) \cdot \left(I_4 \otimes L_2^8\right)\right)$$

Mapping to C code + vector API

```
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
...

Organization

- Mathematical Framework

  Transforms, Rules, and Formulas
- Formula Generator

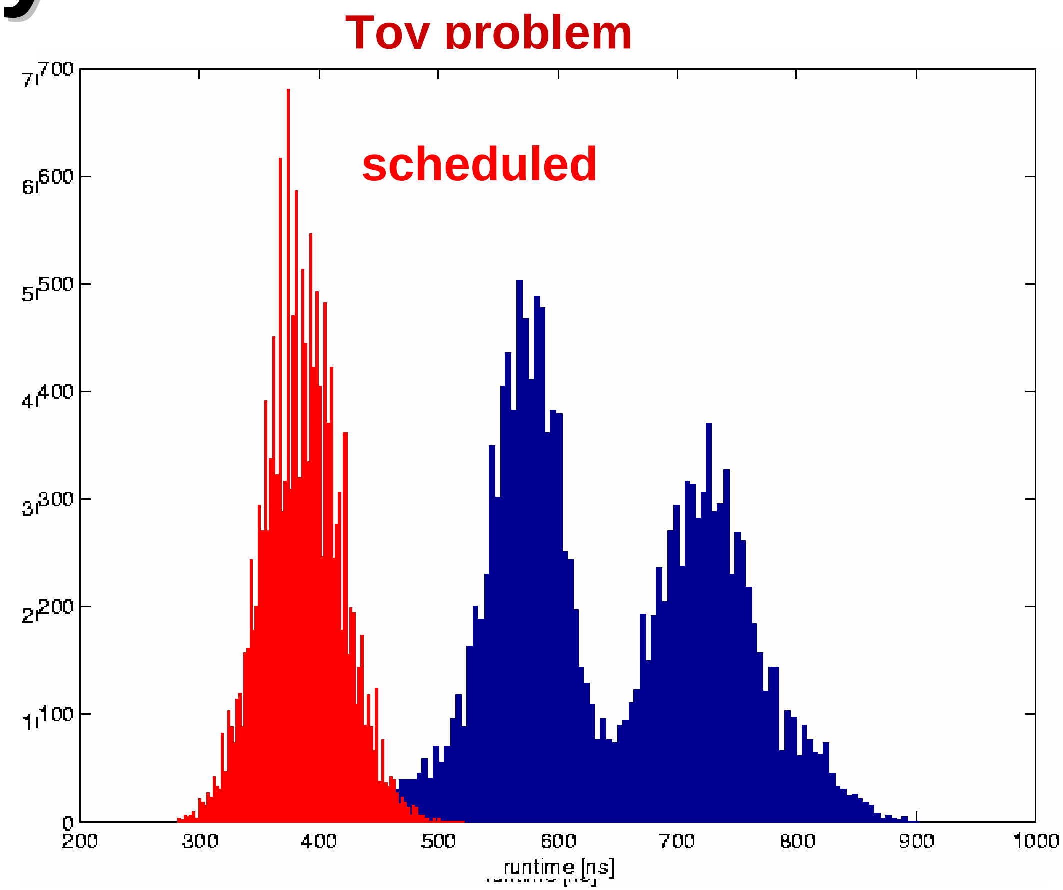
  Transform $\rightarrow$ Algorithm
- SPL and SPL Compiler

  Algorithm $\rightarrow$ Implementation
- **Search Engine**

  How to find the best implementation
- SPIRAL system

  Everything taken together
- Conclusions

Why Search?

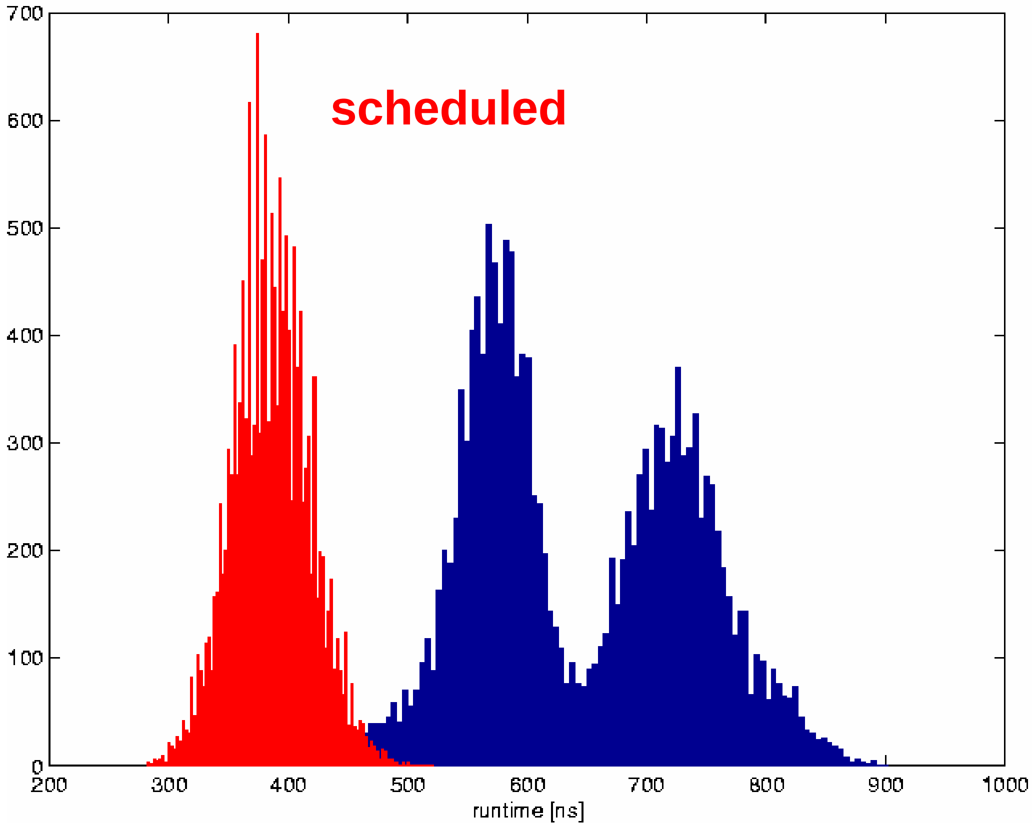


DCT IV- size 2^4

~31000 formulas

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

Why Search?



Toy problem:

DCT, type IV, size 16

~31000 formulas

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

Number of Formulas/Algorithms

k	# DFT, size 2^k	# DCT-IV, size 2^k
1	1	1
2	6	10
3	40	126
4	296	31242
5	27744	1924443362
6	162570361280	7343815121631354242
7	~1.01 • 10^27	~1.07 • 10^38
8	~2.31 • 10^61	~2.30 • 10^76
9	~2.86 • 10^133	~1.06 • 10^153

- differ in data flow not in arithmetic cost
- exponential search space

Search Methods Available in SPIRAL

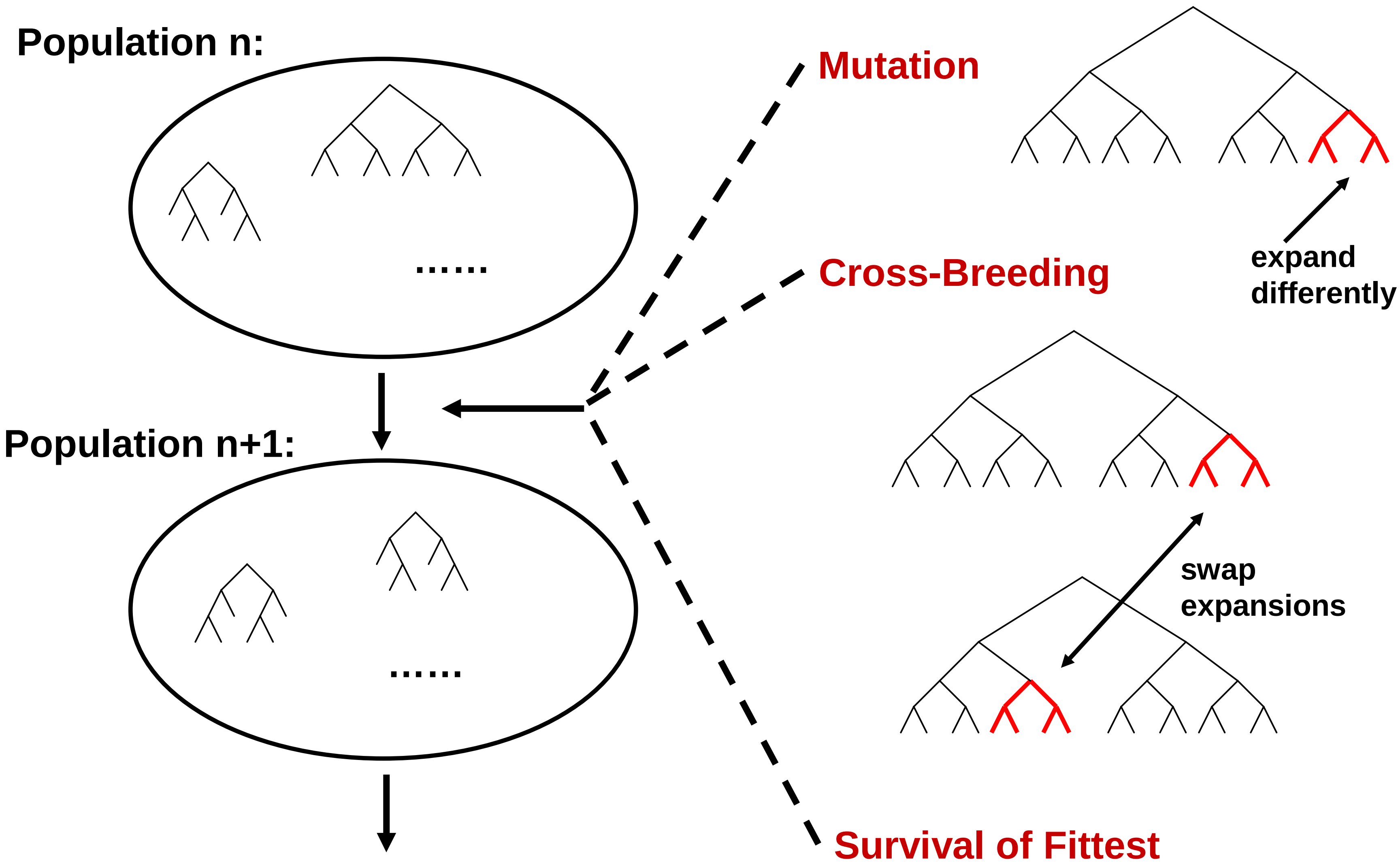
- Exhaustive Search
- Dynamic Programming (DP)
- Random Search
- Hill Climbing
- STEER (similar to a genetic algorithm)

	Possible Sizes	Formulas Timed	Results
Exhaust	Very small	All	Best
DP	All	10s-100s	(very) good
Random	All	User decided	fair/good
Hill Climbing	All	100s-1000s	Good
STEER	All	100s-1000s	(very) good

Search over

- algorithm space and
- implementation options (degree of unrolling)

STEER


Population n:
......
Population n+1:
......
Mutation
expand differently
Cross-Breeding
swap expansions
Survival of Fittest

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

Size	Number of Formulas Generated	Generated Included the Fastest Known	Top N Fastest Known Formulas in Generated
2^{12}	101	yes	77
2^{13}	86	yes	4
2^{14}	101	yes	70
2^{15}	86	yes	11
2^{16}	101	yes	68
2^{17}	86	yes	15
2^{18}	101	yes	25
2^{19}	86	yes	16
2^{20}	101	yes	16

Some Experimental Results

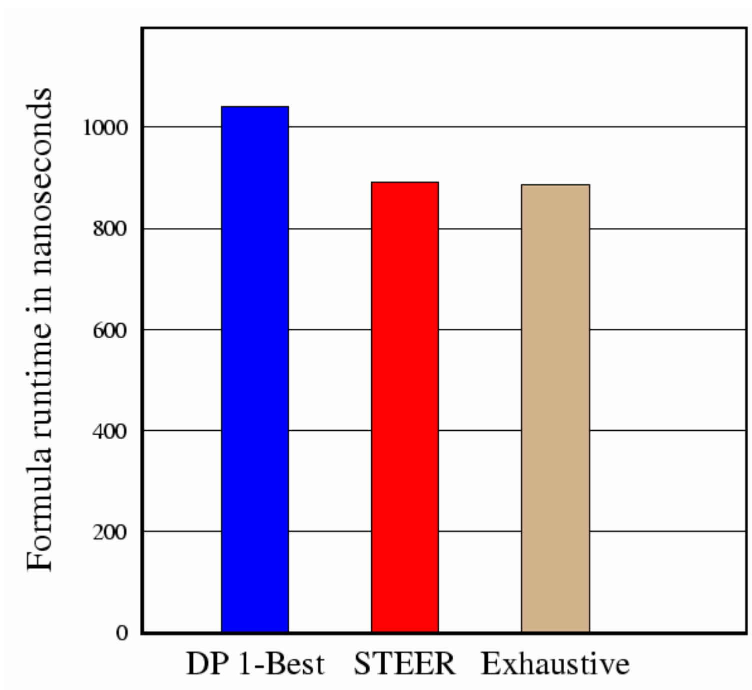
Carnegie Mellon

SPIRAL

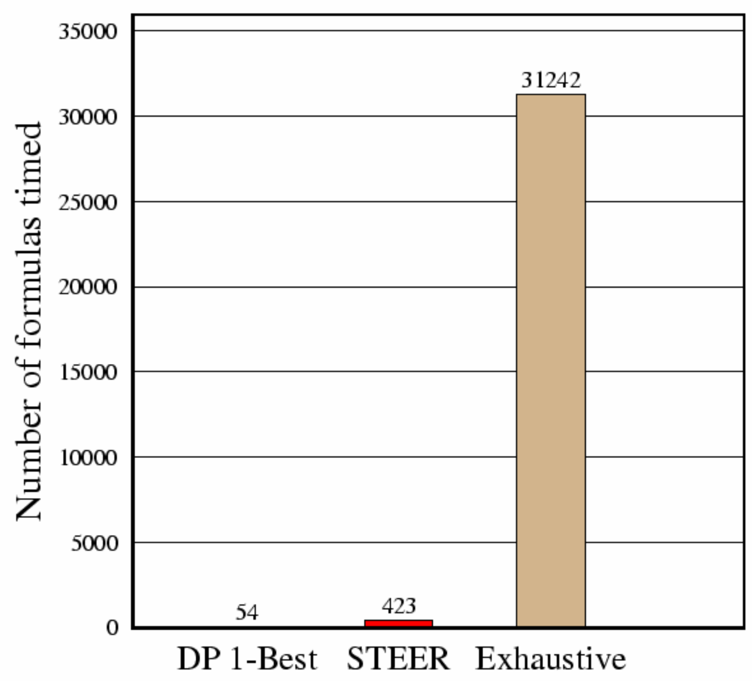
Electrical & Computer ENGINEERING

DCT Type IV Size 16

Fastest Found Formulas



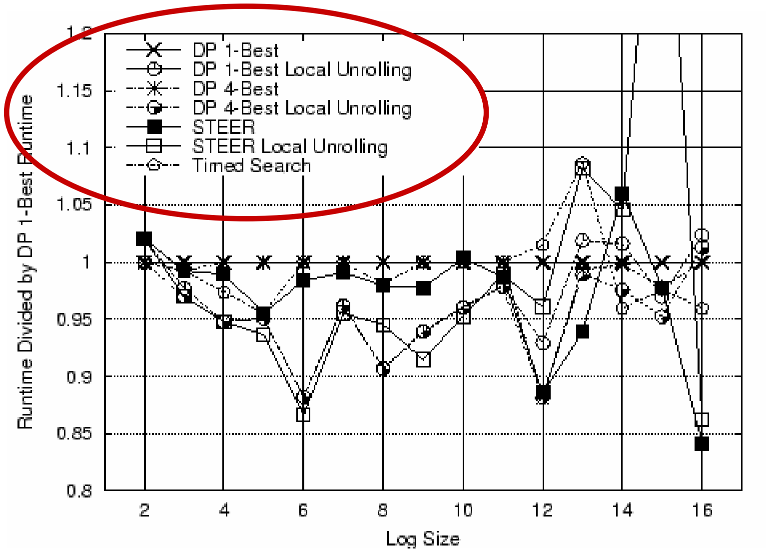
Number of Formulas Timed



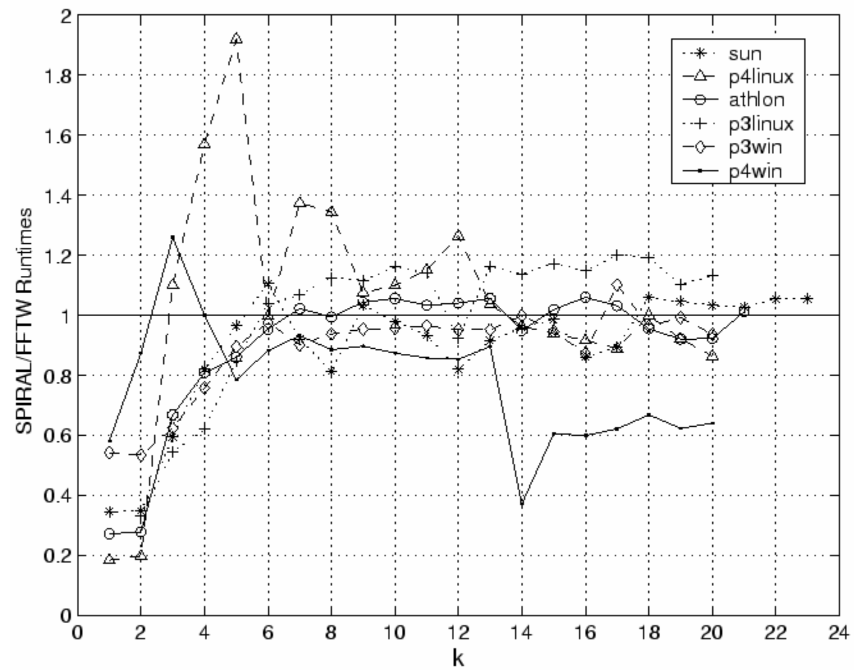
Experimental Results

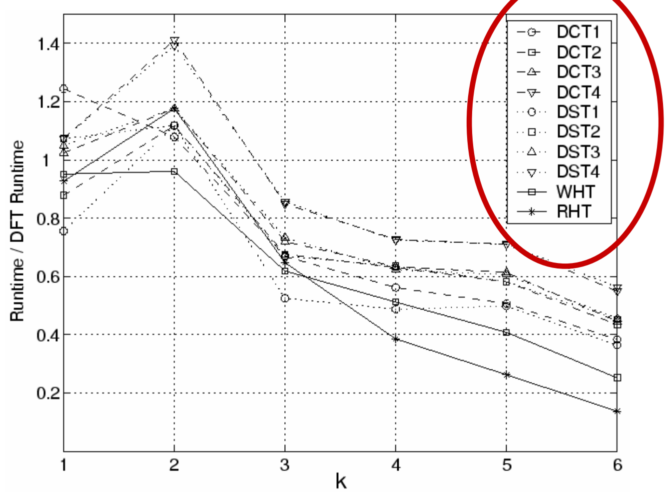
Carnegie Mellon

search methods (applicable to all transforms)



high performance code (compared with FFTW)





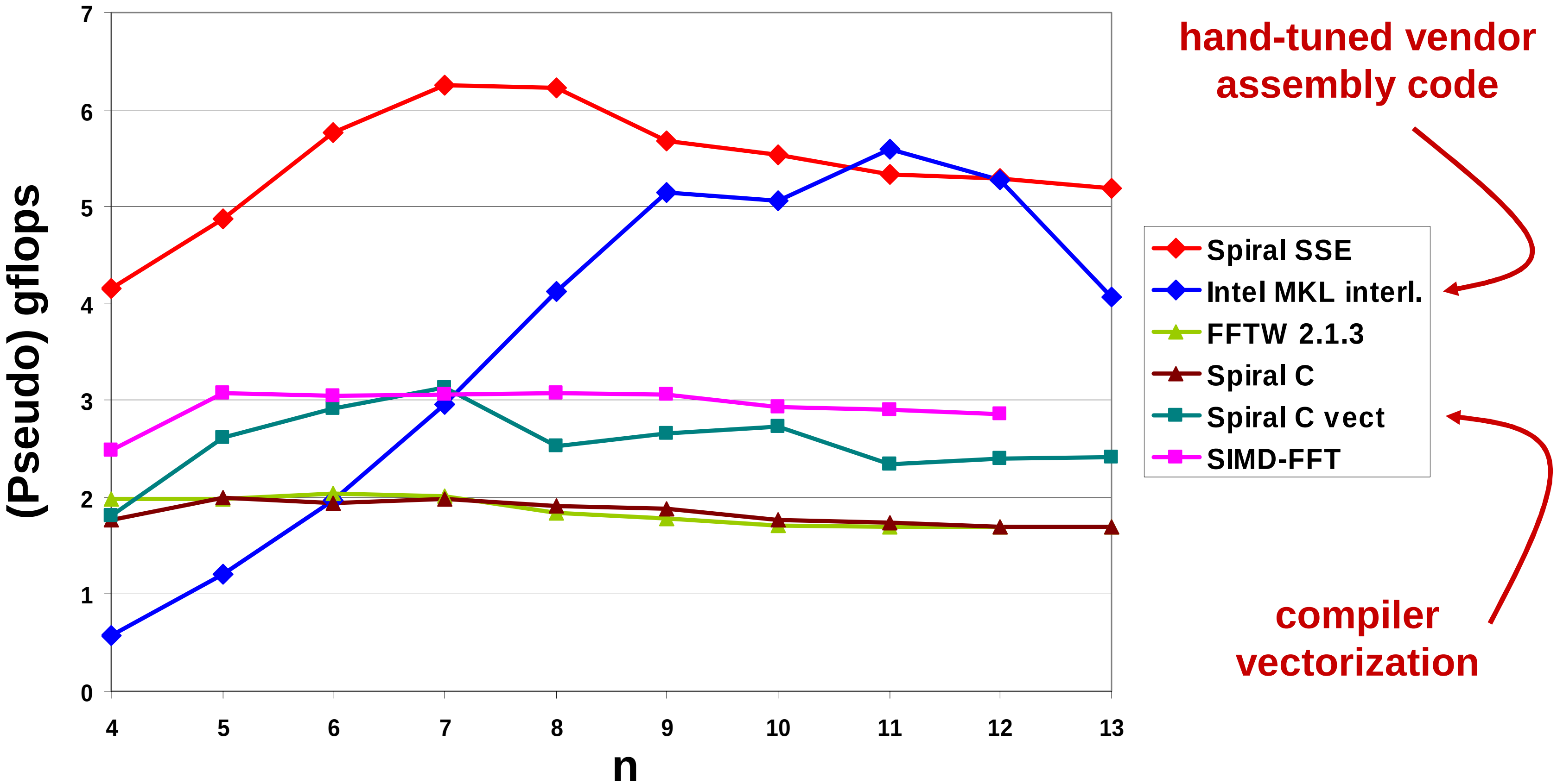
different transforms

SPIRAL

Electrical & Computer ENGINEERING

Some Experimental Results

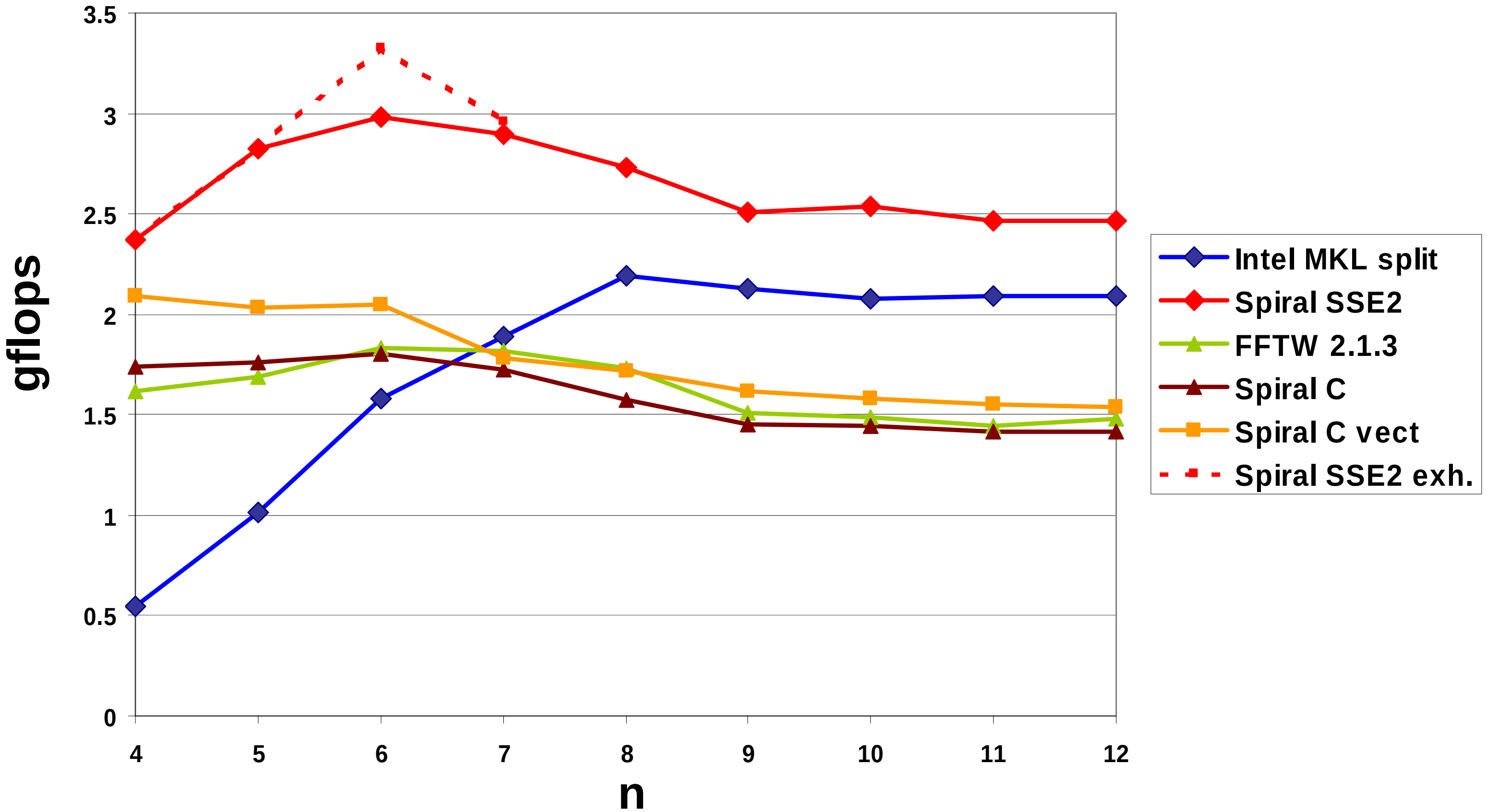
Generated DFT Code: Pentium 4, SSE



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**

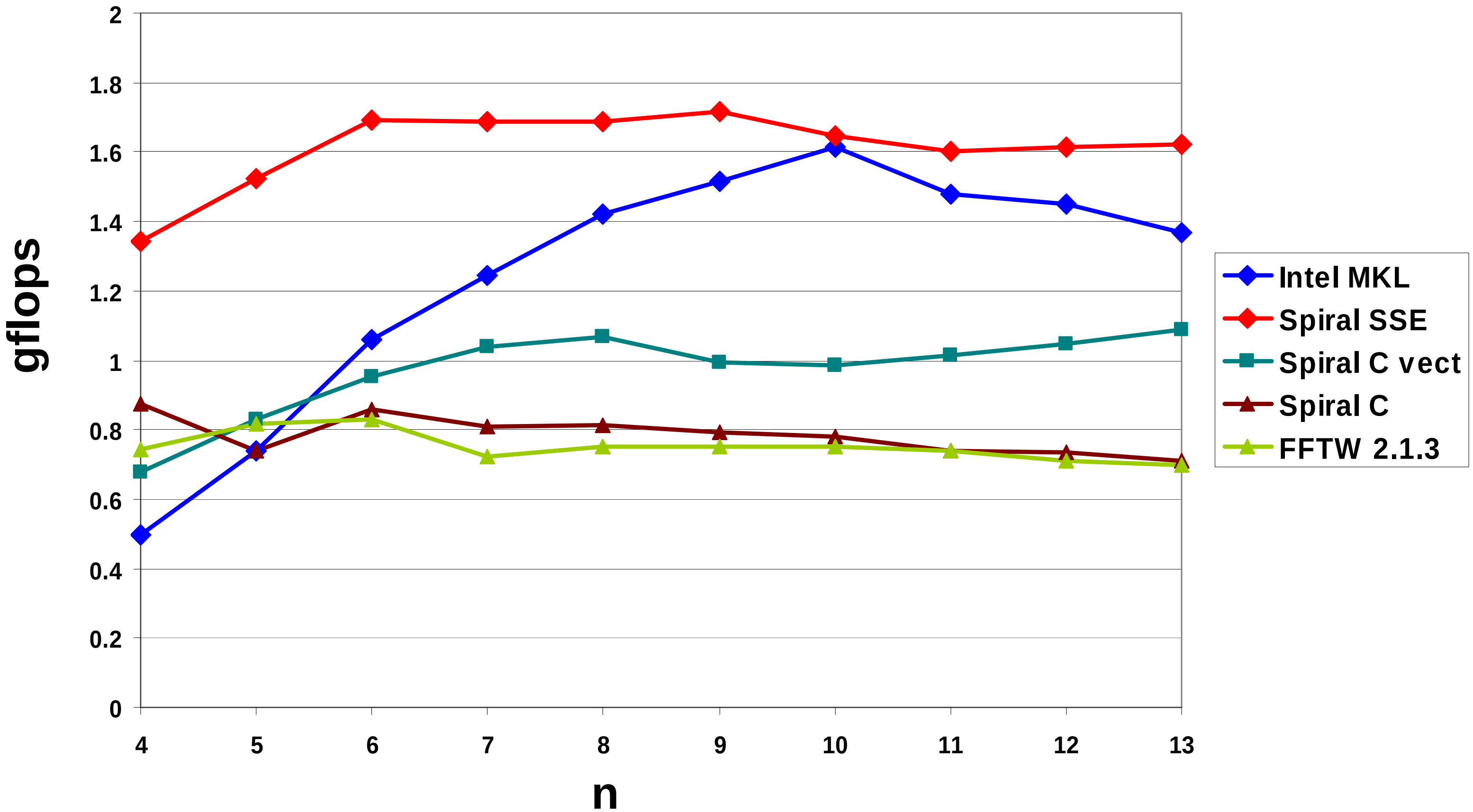
Generated DFT Code: Pentium 4, SSE2



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

Generated DFT Code: Pentium III, SSE



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

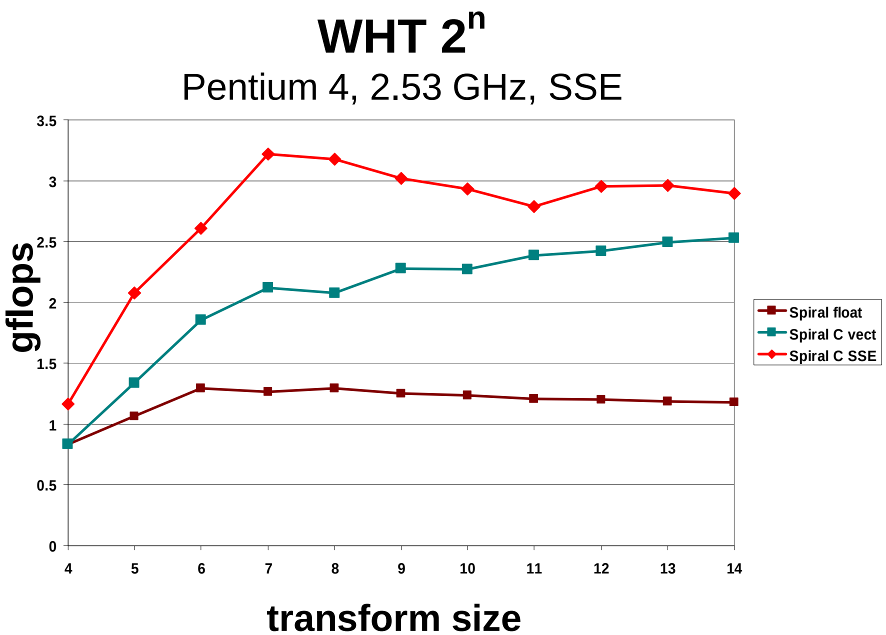
speedups (to C code) up to factor of 2.1

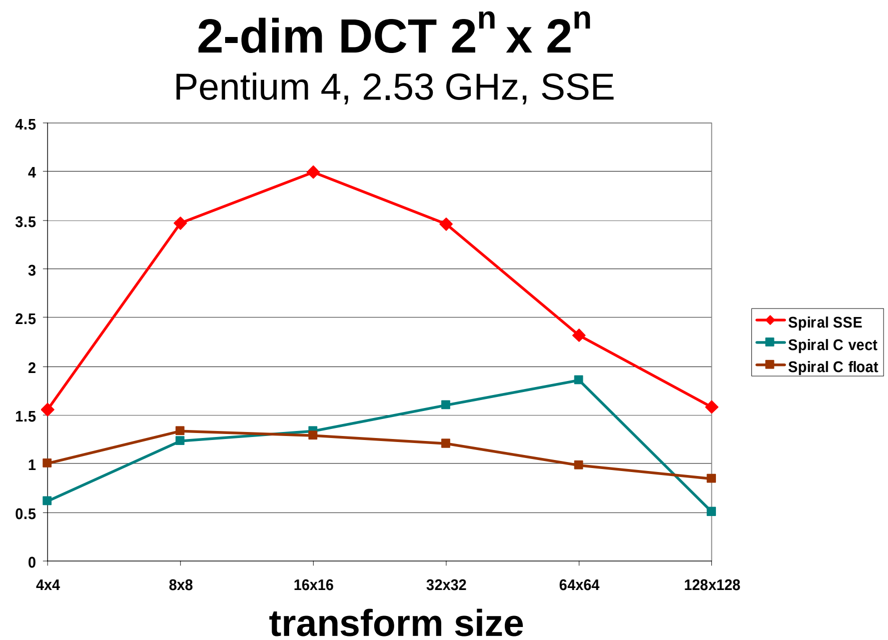
Carnegie Mellon

SPIRAL

Electrical & Computer ENGINEERING

Other transforms





- WHT has only additions
- very simple transform

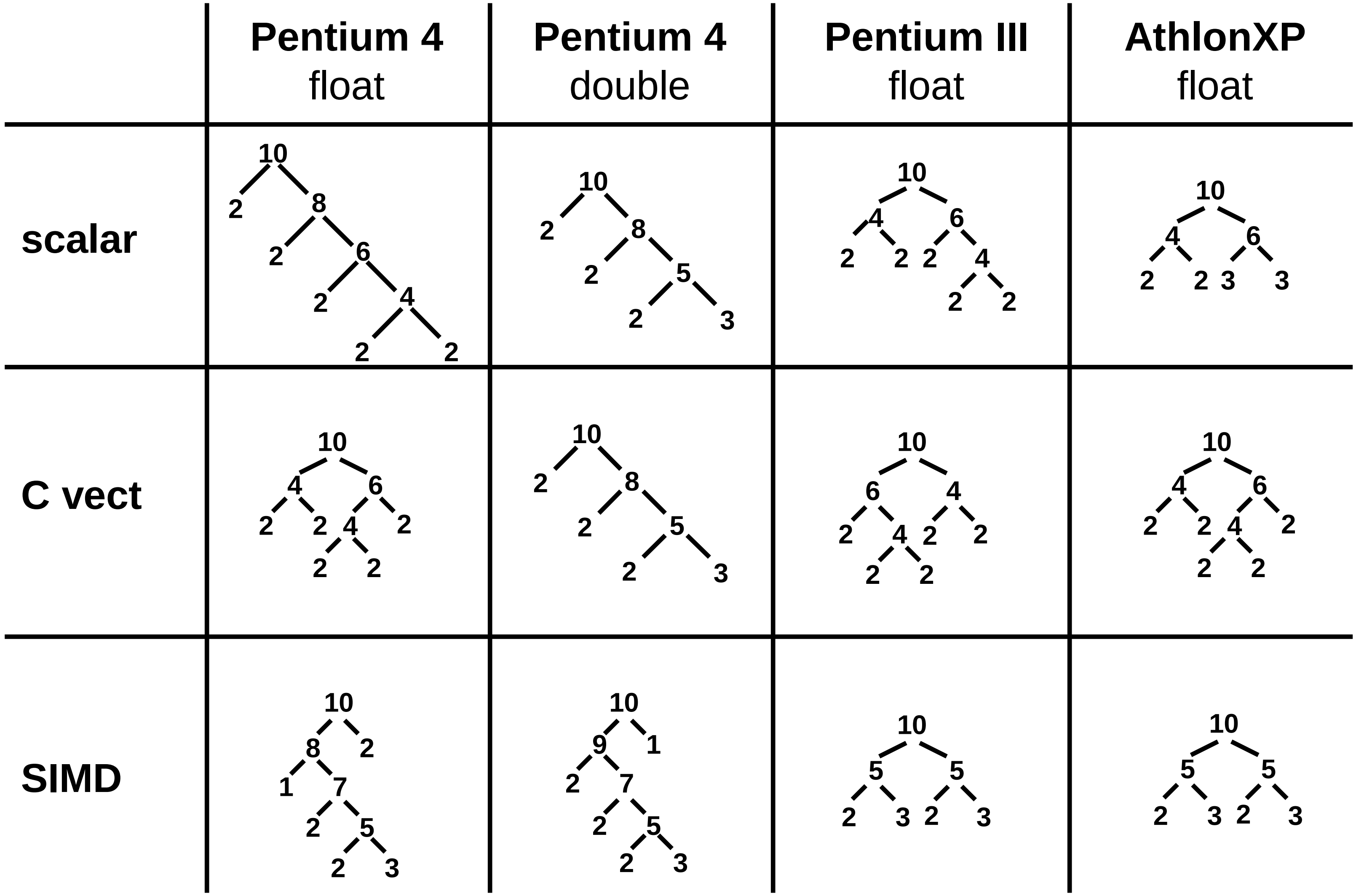
speedups (to C code) up to factor of 3

Carnegie Mellon

SPIRAL

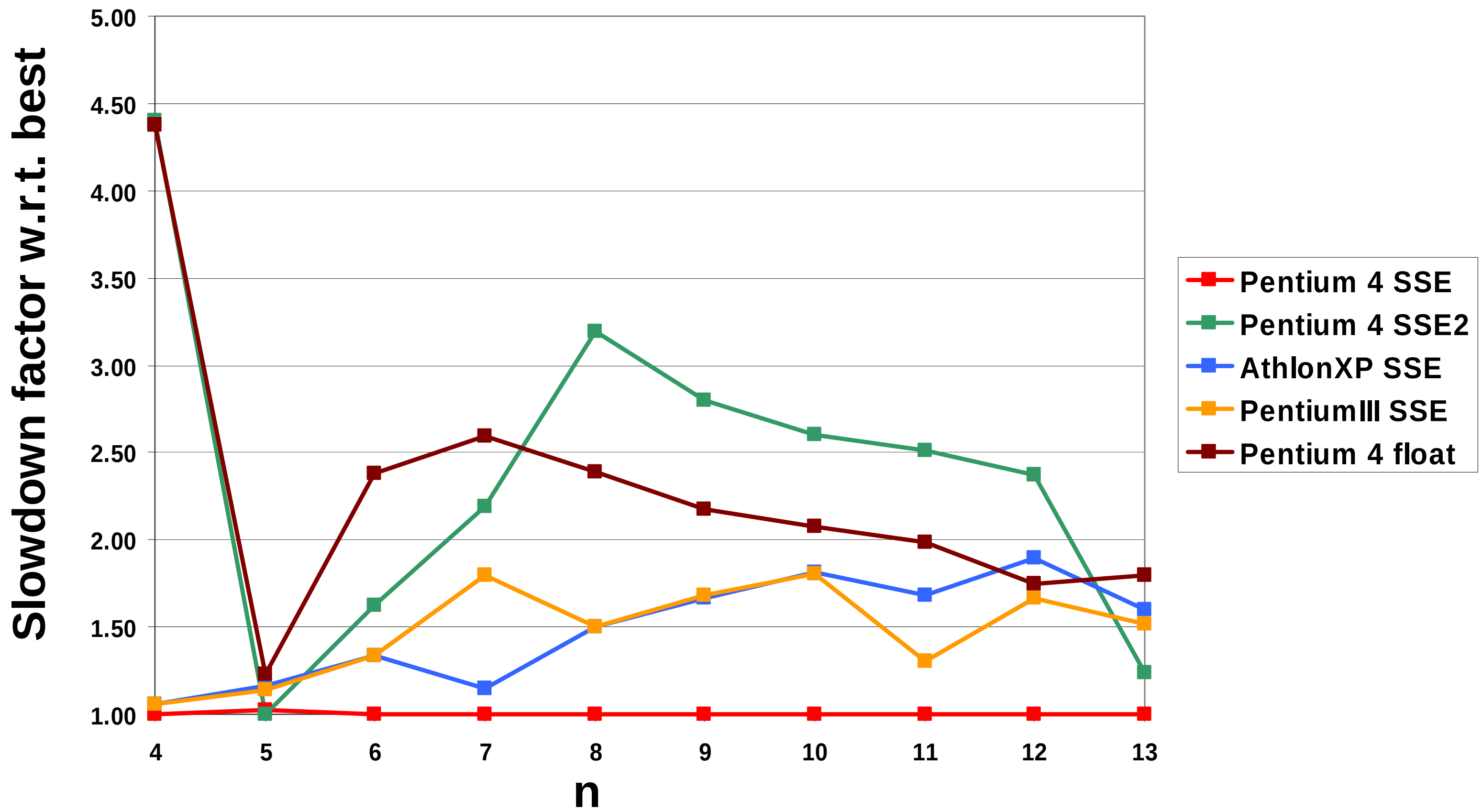
Electrical & Computer ENGINEERING

Best DFT Trees, size 2^{10} = 1024



trees platform/datatype dependent

Crosstiming of best trees on Pentium 4



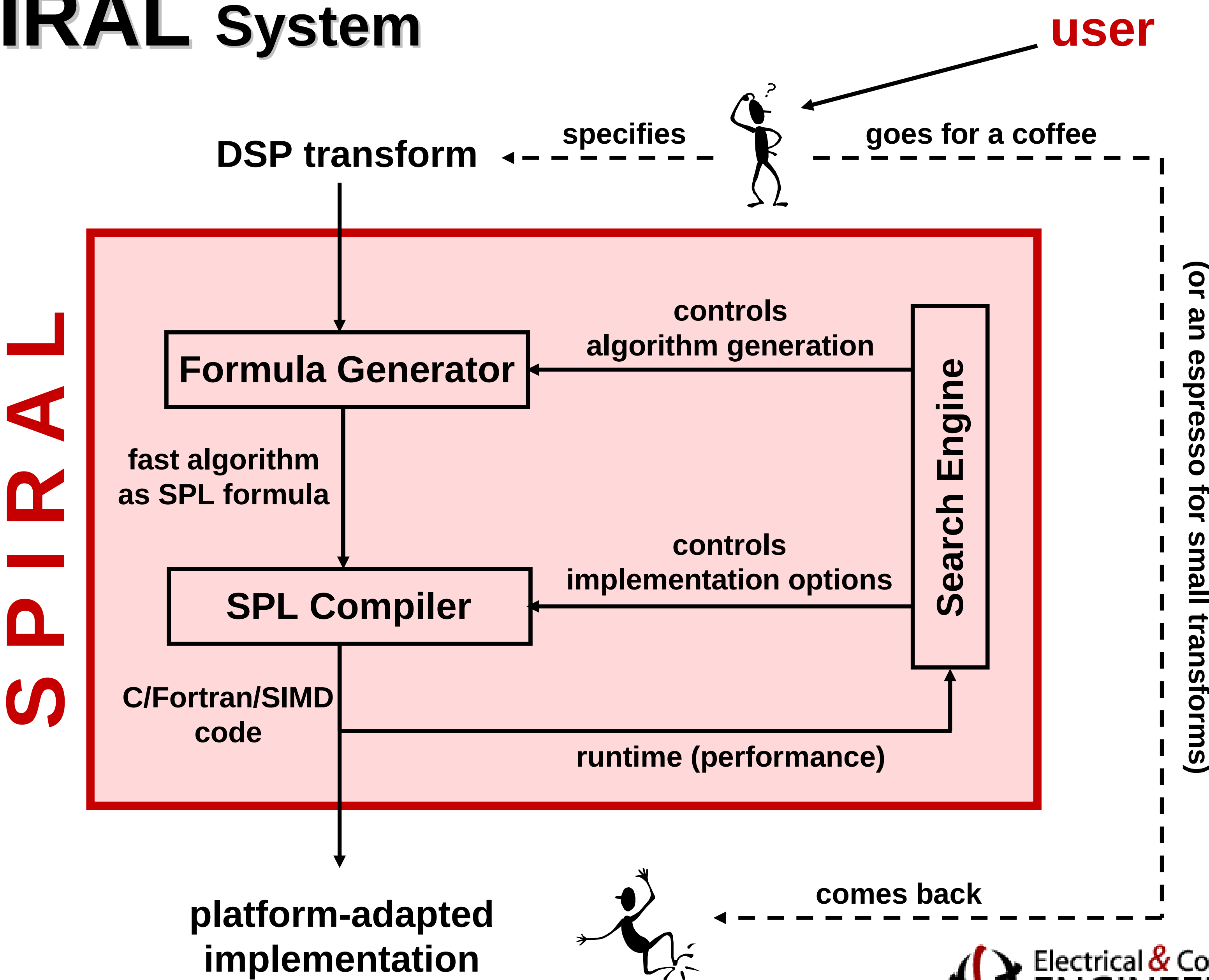
DFT 2^n *single precision*, runtime of best found of other platforms

software adaptation is necessary

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

SPIRAL System



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**

Carnegie Mellon

SPIRAL

Electrical & Computer ENGINEERING

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**