Algorithms and Computation in Signal Processing

special topic course 18-799B
spring 2005
24th and 25th Lecture Apr. 07 and 12, 2005

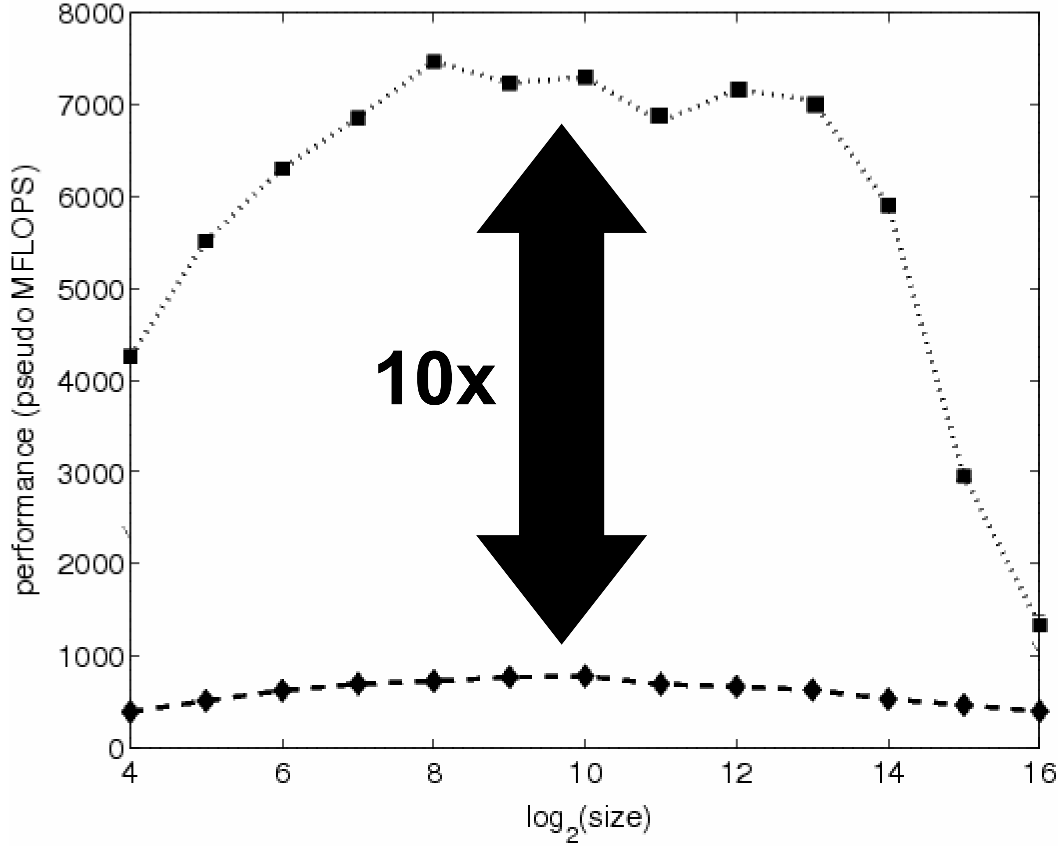
Instructor: Markus Pueschel

TA: Srinivas Chellappa

Research Projects

- **Presentations last week of April (26th and 28th)**
  - We distribute the dates in the lecture on the 12th
  - Presentations 20 minutes + 5 minutes questions (~17-20 slides)
- **Research paper**
  - Due April 20th, the only thing that may be missing are some (but not all) experimental results
  - You'll get feedback from me
  - Final version with feedback incorporated due one week after your presentation
- **Remarks**
  - Follow guide to benchmarking!
  - Try different sets of compiler flags to be sure
  - Do a cost analysis

The Problem Again



DFT size

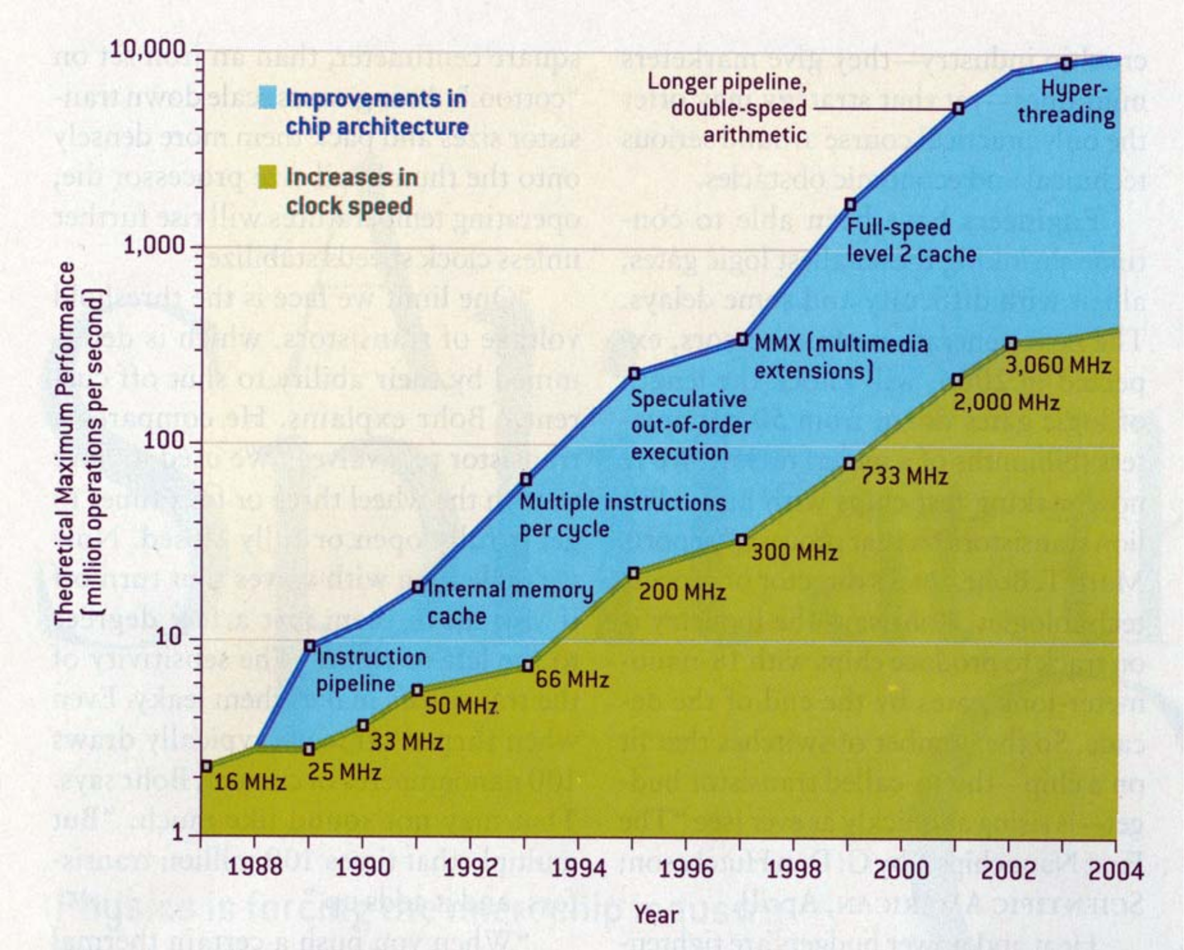
Intel vendor library (hand-optimized assembly code) **but also FFTW, SPIRAL generated code**

reasonable implementation (Numerical recipes. GNU scientific library)

Writing fast numerical code is a tough problem

Moore's Law

- **Moore's Law: exponential (x2 in ~18 months) increase number of transistors/chip**



source: Scientific American, Nov 2004, p. 98

But everything has its price …

Moore's Law: Consequences

- **Computers are very complex**
  - multilevel memory hierarchy
  - special instruction sets beyond standard C programming model
  - undocumented hardware optimizations
- **Consequences:**
  - Runtime depends only roughly on the operations
  - Runtime behavior is hard to understand
  - Compiler development can hardly keep track
  - **The best code (and algorithm) is platform-dependent**
  - **It is very difficult to write really fast code**
- **Computers evolve fast**
  - Highly tuned code becomes obsolete almost as fast as it as written
- **It'll get rather worse: Multicoresystems**

Solution #1: Brute Force

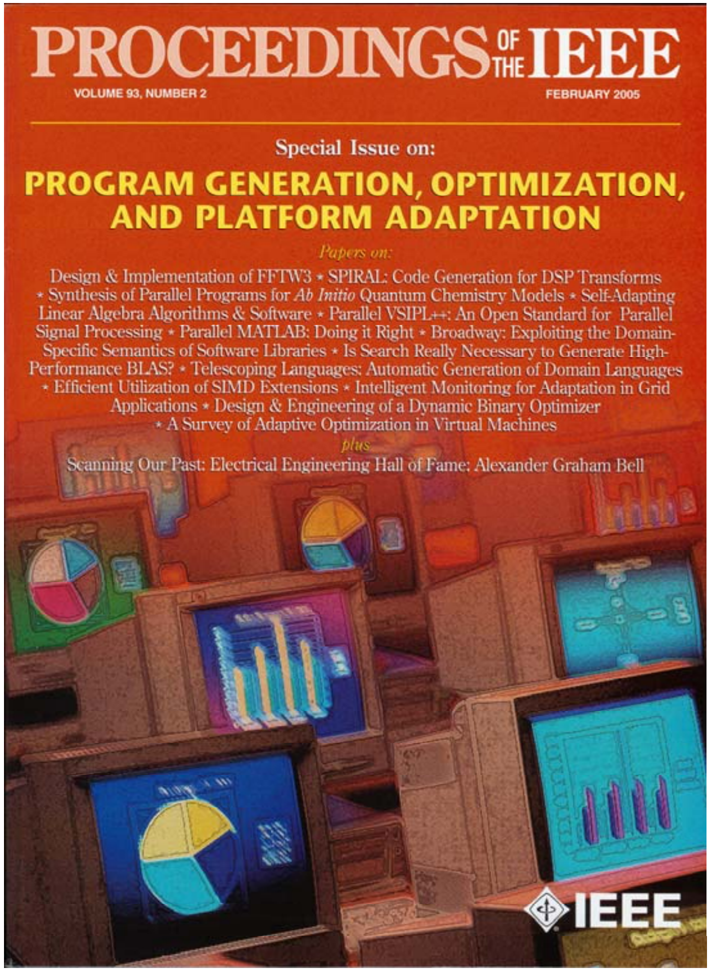
- **Thousands of programmers hand-write and hand-tune (assembly) code for the same numerical problems and for every platform and whenever a new platform comes out?**

Hmm…..

(but it’s current practice)

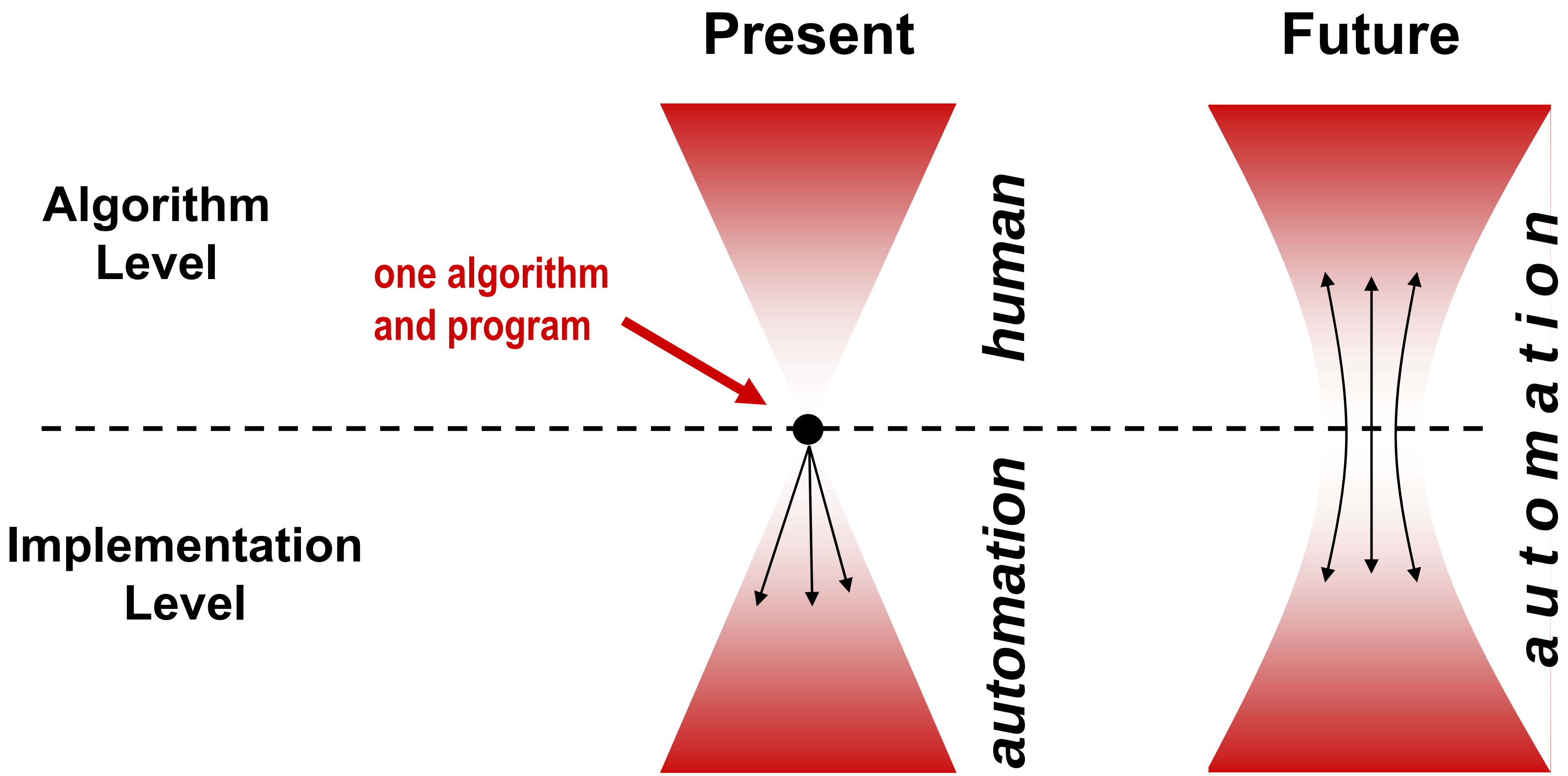
Solution #2: New Approaches to Code Optimization and Code Creation

- **ATLAS**: Code generation/optimization for BLAS
- **SPARSITY/BeBop**: Code generation/optimization for sparse linear algebra routines
- **FFTW**: Self-adaptive DFT library + DFT kernel generator
- **SPIRAL**: Code generation/optimization for linear signal transforms



Proceedings of the IEEE special issue, Feb. 2005

Common Philosophy



a new breed of domain-aware approaches/tools push automation beyond what is currently possible applies for software and hardware design alike

SPIRAL `www.spiral.net`

Sponsors:
DARPA
NSF ACR-0234293
NSF ITR/NGS-0325687

and:
Cylab, CMU
Austrian Science Fund
Intel
ITRI, Taiwan
ENSCO, Inc.

~40 Publications

Overview paper:

Markus Püschel, José M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan Singer, Jianxin Xiong, Franz Franchetti, Aca Gacic, Yevgen Voronenko, Kang Chen, Robert W. Johnson, and Nick Rizzolo, **SPIRAL: Code Generation for DSP Transforms,** Proceedings of the IEEE

Team:
James C. Hoe (ECE, CMU)
Jeremy Johnson (CS, Drexel)
José M. F. Moura (ECE, CMU)
David Padua (CS, UIUC)
Markus Püschel (ECE, CMU)
Manuela Veloso (CS, CMU)
Robert W. Johnson (Quarry Comp. Inc.)
Christoph Überhuber (Math, TU Wien

Students/PostDocs:
Bryan W. Singer (CS, CMU)
Jianxin Xiong (CS, UIUC)
Srinivas Chellappa (ECE, CMU)
Franz Franchetti (ECE, CMU, before TU Vienna)
Aca Gacic (ECE, CMU)
Yevgen Voronenko (ECE, CMU)
Anthony Breitzman (CS, Drexel)
Kang Chen (CS, Drexel)
Pinit Kumhom (ECE, Drexel)
Adam Zelinski (ECE, CMU)
Peter Tummeltshammer (CS, TU Vienna)
…

Spiral

- Code generation from scratch for linear digital signal processing (DSP) transforms (DFT, DCT, DWT, filters, ….)
- Automatic optimization and platform-tuning at the algorithmic level and the code level
- Different code types supported (scalar, vector, FMA, fixed-point, multiplierless, …)

Goal: A flexible, extensible code generation framework that can survive time (to whatever extent possible) for an entire domain of algorithms

Research question: To what extent is it possible to abolish handcoding and handoptimization?

Code Generation and Tuning as Optimization Problem

T	a DSP transform to be implemented
P	the target platform
$\mathcal{I} = \mathcal{I}(\mathrm{T}, \mathrm{P})$	set of possible implementations of T on P
$\mathrm{C} = \mathrm{C}(\mathrm{T}, \mathrm{I}, \mathrm{P})$	cost of implementation I of T on P

The implementation of T that is tuned to P is given by:

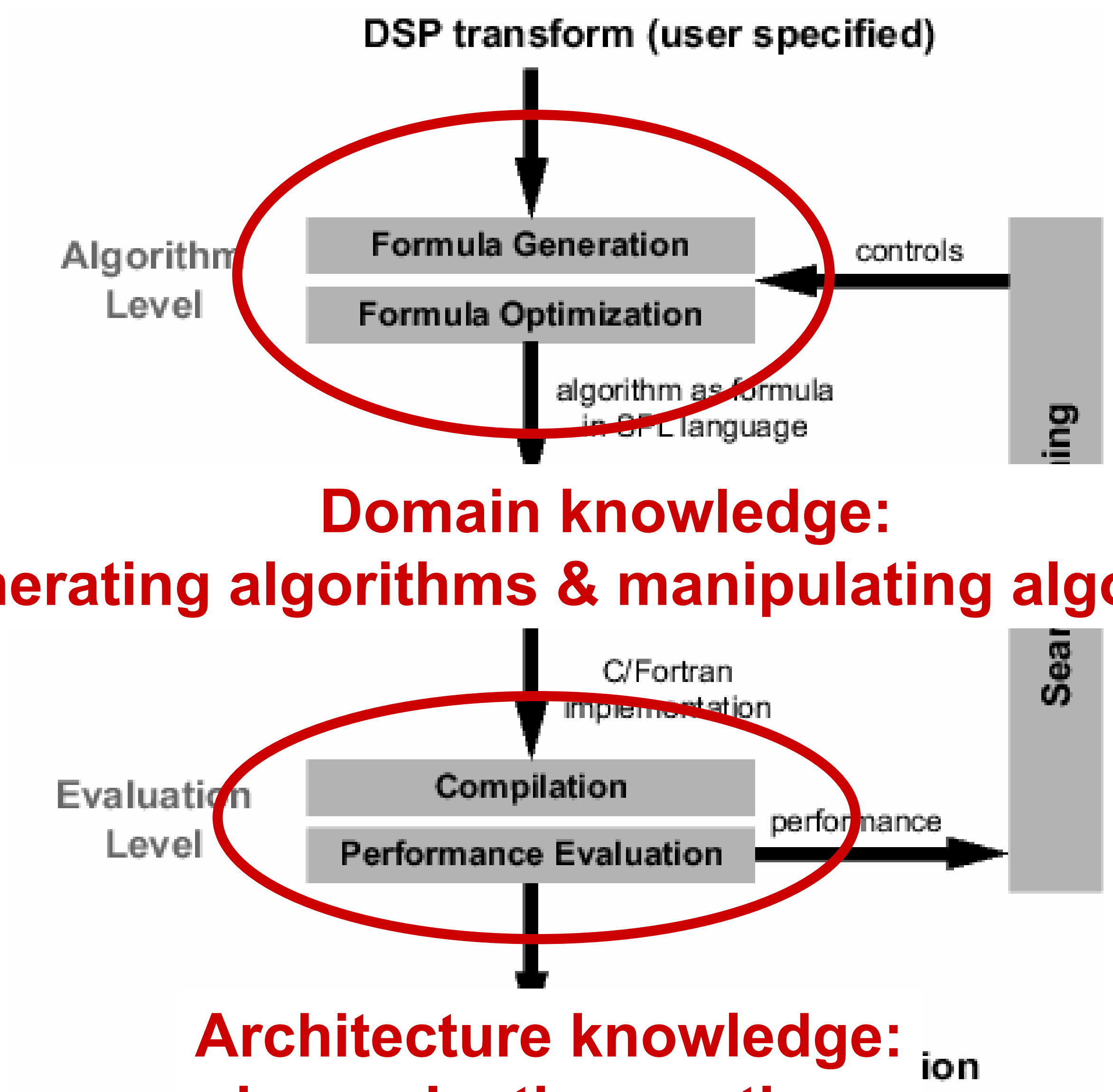
$$\widehat{\mathrm{I}} = \widehat{\mathrm{I}}(\mathrm{P}) = \arg\min_{\mathrm{I} \in \mathcal{I}(\mathrm{P})} \mathrm{C}\left(\mathrm{T}, \mathrm{P}, \mathrm{I}\right)$$

Problems:

- How to characterize and generate the set of implementations?
- How to efficiently minimize C?

Spiral exploits the domain-specific structure to implement a solver for this optimization problem

Spiral's architecture



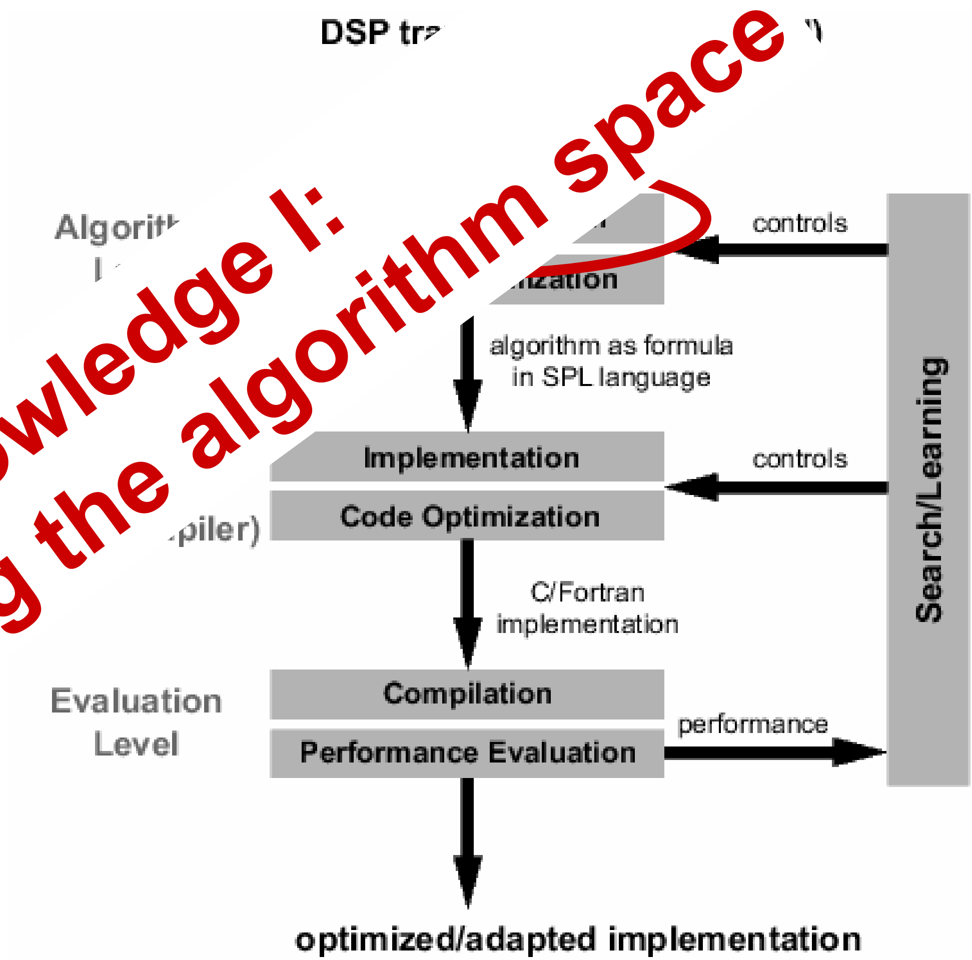
Domain knowledge: Generating algorithms & manipulating algorithms

Architecture knowledge: by evaluating runtime

From Transform to Algorithm (Formula)

Input:
Transform specification

Output:
Fast algorithm as formula



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

- mathematical notation exhibits structure: SPL (signal processing language)
- Suitable for computer representation
- contains all information to generate code

SPL: Definition (BNF)

- **Description language for linear DSP algorithms**
- **Definition (BNF):**

$$
\begin{array}{rcll}
\langle\mathsf{spl}\rangle & ::= & \langle\mathsf{generic}\rangle \mid \langle\mathsf{symbol}\rangle \mid \langle\mathsf{transform}\rangle \mid & \\
 & & \langle\mathsf{spl}\rangle \cdot \cdots \cdot \langle\mathsf{spl}\rangle \mid & \text{(product)} \\
 & & \langle\mathsf{spl}\rangle \oplus \ldots \oplus \langle\mathsf{spl}\rangle \mid & \text{(direct sum)} \\
 & & \langle\mathsf{spl}\rangle \otimes \cdots \otimes \langle\mathsf{spl}\rangle \mid & \text{(tensor product)} \\
 & & \mathrm{I}_n \otimes_k \langle\mathsf{spl}\rangle \mid \mathrm{I}_n \otimes^k \langle\mathsf{spl}\rangle \mid & \text{(overlapped tensor product)} \\
 & & \overline{\langle\mathsf{spl}\rangle} \mid & \text{(conversion to real)} \\
 & & \ldots & \\
\langle\mathsf{generic}\rangle & ::= & \mathsf{diag}(a_0, \ldots, a_{n-1}) \mid \ldots & \\
\langle\mathsf{symbol}\rangle & ::= & \mathrm{I}_n \mid \mathsf{J}_n \mid \mathsf{L}^n_k \mid \mathsf{R}_\alpha \mid \mathsf{F}_2 \mid \ldots & \\
\langle\mathsf{transform}\rangle & ::= & \mathbf{DFT}_n \mid \mathbf{WHT}_n \mid \mathbf{DCT\text{-}2}_n \mid \mathbf{Filt}_n(h[z]) \mid \ldots &
\end{array}
$$

Some Definitions:

$$A \oplus B = \begin{bmatrix} A & \\ & B \end{bmatrix}$$

$$A \otimes B = \begin{bmatrix} a_{k,\ell} B \end{bmatrix}, \quad \text{where } A = \begin{bmatrix} a_{k,\ell} \end{bmatrix}$$

$$\mathsf{F}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$\mathsf{R}_\alpha = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix}$$

$$I_n \otimes A = \begin{bmatrix} A & & & \\ & A & & \\ & & \ddots & \\ & & & A \end{bmatrix}$$

DSP Algorithms: Spiral Terminology

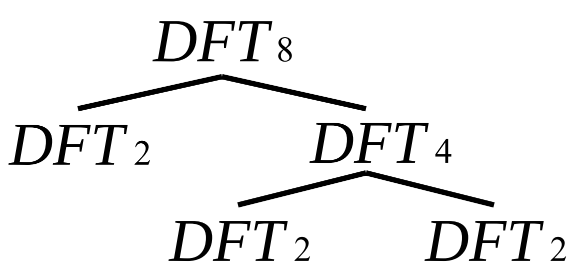
Transform DFT_n **parameterized matrix**

Rule

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

- a breakdown strategy
- product of sparse matrices

Ruletree



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

Formula

$$DFT_8 = \left(F_2 \otimes I_4\right) \cdot D \cdot \left(I_2 \otimes \left(I_2 \otimes F_2 \cdots\right)\right) \cdot P$$

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

Some Transforms

$$
\begin{aligned}
\mathbf{DCT\text{-}2}_n &= \left[\cos(k(2\ell+1)\pi/2n)\right]_{0\le k,\ell<n},\\
\mathbf{DCT\text{-}3}_n &= \mathbf{DCT\text{-}2}_n^T \quad \text{(transpose)},\\
\mathbf{DCT\text{-}4}_n &= \left[\cos((2k+1)(2\ell+1)\pi/4n)\right]_{0\le k,\ell<n},\\
\mathbf{IMDCT}_n &= \left[\cos((2k+1)(2\ell+1+n)\pi/4n)\right]_{0\le k<2n,0\le\ell<n},\\
\mathbf{RDFT}_n &= [r_{k\ell}]_{0\le k,\ell<n}, \quad r_{k\ell} = \begin{cases} \cos\frac{2\pi k\ell}{n}, & k \le \lfloor\frac{n}{2}\rfloor \\ -\sin\frac{2\pi k\ell}{n}, & k > \lfloor\frac{n}{2}\rfloor \end{cases},\\
\mathbf{WHT}_n &= \begin{bmatrix} \mathbf{WHT}_{n/2} & \mathbf{WHT}_{n/2} \\ \mathbf{WHT}_{n/2} & -\mathbf{WHT}_{n/2} \end{bmatrix}, \quad \mathbf{WHT}_2 = \mathbf{DFT}_2,\\
\mathbf{DHT} &= \left[\cos(2k\ell\pi/n) + \sin(2k\ell\pi/n)\right]_{0\le k,\ell<n}.
\end{aligned}
$$

Spiral currently contains 36 transforms

Some Breakdown Rules

$$
\begin{aligned}
\mathbf{DFT}_n &\to (\mathbf{DFT}_k \otimes \mathrm{I}_m)\,\mathsf{T}^n_m(\mathrm{I}_k \otimes \mathbf{DFT}_m)\,\mathsf{L}^n_k, \qquad n = km \\
\mathbf{DFT}_n &\to P_n(\mathbf{DFT}_k \otimes \mathbf{DFT}_m)Q_n, \qquad n = km,\ \mathrm{gcd}(k,m) = 1 \\
\mathbf{DFT}_p &\to R_p^T(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1})D_p(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1})R_p, \qquad p \text{ prime} \\
\mathbf{DCT\text{-}3}_n &\to (\mathrm{I}_m \oplus \mathsf{J}_m)\,\mathsf{L}^n_m(\mathbf{DCT\text{-}3}_m(1/4) \oplus \mathbf{DCT\text{-}3}_m(3/4)) \\
&\qquad \cdot(\mathsf{F}_2 \otimes \mathrm{I}_m)\begin{bmatrix} \mathrm{I}_m & 0 \oplus -\mathsf{J}_{m-1} \\ & \frac{1}{\sqrt{2}}(\mathrm{I}_1 \oplus 2\,\mathrm{I}_m) \end{bmatrix}, \qquad n = 2m \\
\mathbf{DCT\text{-}4}_n &\to S_n\mathbf{DCT\text{-}2}_n\,\mathrm{diag}_{0\le k<n}(1/(2\cos((2k+1)\pi/4n))) \\
\mathbf{IMDCT}_{2m} &\to (\mathsf{J}_m \oplus \mathrm{I}_m \oplus \mathrm{I}_m \oplus \mathsf{J}_m)\left(\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} \otimes \mathrm{I}_m\right) \oplus \left(\begin{bmatrix} -1 \\ -1 \end{bmatrix} \otimes \mathrm{I}_m\right)\right)\mathsf{J}_{2m}\,\mathbf{DCT\text{-}4}_{2m} \\
\mathbf{WHT}_{2^k} &\to \prod_{i=1}^{t}(\mathrm{I}_{2^{k_1+\cdots+k_{i-1}}} \otimes \mathbf{WHT}_{2^{k_i}} \otimes \mathrm{I}_{2^{k_{i+1}+\cdots+k_t}}), \qquad k = k_1 + \cdots + k_t \\
\mathbf{DFT}_2 &\to \mathsf{F}_2 \\
\mathbf{DCT\text{-}2}_2 &\to \mathrm{diag}(1, 1/\sqrt{2})\,\mathsf{F}_2 \\
\mathbf{DCT\text{-}4}_2 &\to \mathsf{J}_2\,\mathsf{R}_{13\pi/8}
\end{aligned}
$$

Base case rules

Spiral contains 100+ rules

Some Breakdown Rules for Filters

$$\mathbf{Filt}_n(h(z)) \rightarrow \mathrm{I}_{\lfloor \frac{n}{b} \rfloor} \otimes_{l+r} \left(\left[\; \big| \; \right]_{i=0}^{\left\lceil \frac{l+r}{b} \right\rceil} \mathbf{T}_b\Big(h(z) z^{l-ib}\Big) \oplus^k \mathbf{T}_k \Big(h(z) z^{l - \left\lceil \frac{l+r}{b} \right\rceil b - k} \Big) \right)$$

$$\mathbf{Filt}_n(h(z)) \rightarrow \mathrm{L}^n_{\frac{n}{2}} \, \mathbf{Filt}_{\frac{n}{2}} \left(\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \right) \cdot$$

$$\mathbf{Filt}_{\frac{n}{2}} \left(\begin{bmatrix} h_0(z) & & \\ & h_1(z) & \\ & & h_0(z) + h_1(z) \end{bmatrix} \right) \cdot \mathbf{Filt}_{\frac{n}{2} + \frac{r+l-1}{2}} \left(\begin{bmatrix} 1 & -1 \\ z & -1 \\ 0 & 1 \end{bmatrix} \right) \cdot \mathrm{L}_2^{n+r+l}$$

$$\mathbf{Filt}_n(h(z)) \rightarrow \mathrm{R}^{\text{zero}}_{n,l,r} \cdot \mathbf{C}_{n+l+r}(h(z))$$

$$\begin{aligned} \mathbf{C}_n(h(z)) &\rightarrow \mathbf{RDFT}_n^{-1} \cdot X(\hat{\mathbf{h}}) \cdot \mathbf{RDFT}_n, \\ \hat{\mathbf{h}} &= \mathbf{RDFT}_n \cdot \mathbf{h} \end{aligned}$$

Aca Gacic, **Automatic Implementation and Platform Adaptation of Discrete Filtering and Wavelet Algorithms**, Ph.D. thesis, Electrical and Computer Engineering, Carnegie Mellon University, 2004

Formula- (Algorithm) generation

Transform: $\mathrm{DCT\text{-}2}_4$

Ruletree: (many possibilities)

$$\mathrm{DCT\text{-}2}_4 \to \mathrm{DCT\text{-}4}_2 \quad \mathrm{DCT\text{-}2}_2$$

Formula: $\mathsf{L}_2^4(\mathrm{diag}(1, 1/\sqrt{2})\,\mathsf{F}_2 \oplus \mathsf{J}_2\,\mathsf{R}_{13\pi/8})(\mathsf{F}_2 \otimes \mathsf{I}_2)(\mathsf{I}_2 \oplus \mathsf{J}_2)$

Remaining task

(fast) C Code:

```c
void sub(double *y, double *x) {
  double f0, f1, f2, f3, f4, f7, f8, f10, f11;
  f0 = x[0] - x[3];
  f1 = x[0] + x[3];
  f2 = x[1] - x[2];
  f3 = x[1] + x[2];
  f4 = f1 - f3;
  y[0] = f1 + f3;
  y[2] = 0.7071067811865476 * f4;
  f7 = 0.9238795325112867 * f0;
  f8 = 0.3826834323650898 * f2;
  y[1] = f7 + f8;
  f10 = 0.3826834323650898 * f0;
  f11 = (-0.9238795325112867) * f2;
  y[3] = f10 + f11;
}
```

Set of Algorithms

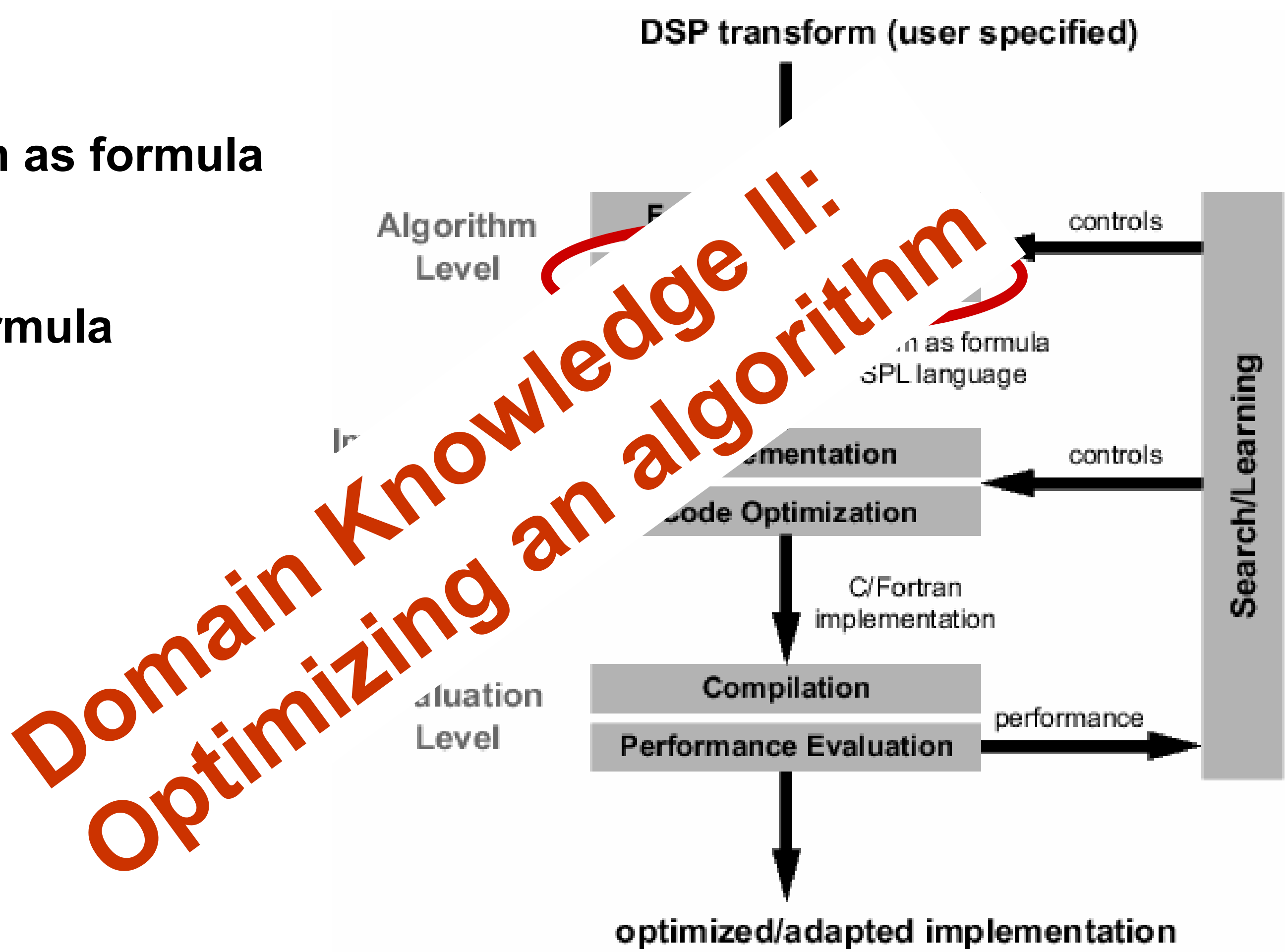
- **Given a transform:**
  - Apply breakdown rules recursively until all occurring transforms are expanded
  - Choice of rules at each step yields (usually) exponentially large algorithms space:
    - about equal in operations count
    - differ in data flow

k	# DFTs, 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

From Algorithm (Formula) to Optimized Algorithm

Input:
 Fast algorithm as formula

Output:
 Optimized formula



Motivation: Loop Fusion

$$(I_4 \otimes F_2)\, L^8_4$$

direct
code generation

```
void I4xF2_L84(double *y, double *x) {
   double t[8];
   for (int i=0; i<8; i++)
      t[i==7 ? 7 : (i*4)%7] = x[i];
   for (int i=0; i<4; i++){
      y[2*i] = t[2*i] + t[2*i+1];
      y[2*i+1] = t[2*i] - t[2*i+1]; } }
```

no compiler does that

Solution: Σ-SPL and
Formula manipulation

$$\sum_{j=0}^{3} S_{(j)_4 \otimes \imath_2} F_2 G_{\imath_2 \otimes (j)_4}$$

```
void I4xF2_L84(double *y, double *x) {
   for (int j=0; j<4; j++){
      y[2*j] = x[j] + x[j+4];
      y[2*j+1] = x[j] - x[j+4]; } }
```

Formula Level Optimization

- **Main goals:**
  - Fusing iterative steps (fusing loops), e.g., permutations with loops
  - Improving structure (data flow) for SIMD instructions
- **Overcomes compiler limitations**
- **Formula manipulation through mathematical rules**
- **Implemented using multiple levels of rewriting systems**
- **Puts math knowledge into the system**

Structure of Loop Optimization

SPL formula

$$\mathrm{T}^{mn}_n (\mathrm{I}_m \otimes C_n)\, \mathrm{L}^{mn}_m$$

To Σ-SPL

Σ -SPL formula

$$\mathrm{diag}\left(\mathrm{t}^{mn}_n\right)\left(\sum_{j=0}^{n-1} \mathrm{S}_{(j)_m\otimes \imath_n}\, C_n\, \mathrm{G}_{(j)_m\otimes \imath_n}\right)\mathrm{perm}\left(\ell^{mn}_m\right)$$

Join permutations

Σ -SPL formula

$$\mathrm{diag}\left(\mathrm{t}^{mn}_n\right)\left(\sum_{j=0}^{n-1} \mathrm{S}_{(j)_m\otimes \imath_n}\, C_n\, \mathrm{G}_{\imath_n\otimes (j)_m}\right)$$

Join diagonals and monomials

Σ -SPL formula

$$\sum_{j=0}^{n-1} \mathrm{S}_{(j)_m\otimes \imath_n}\, \mathrm{diag}\left(\mathrm{t}^{mn}_n \circ \left((j)_m \otimes \imath_n\right)\right) C_n\, \mathrm{G}_{\imath_n\otimes (j)_m}$$

Rules: $\mathrm{G}_r\, \mathrm{perm}\left(\pi\right) = \mathrm{G}_{\pi\circ r}$, $\quad \ell^{mn}_m \circ \left((j)_m \otimes \imath_n\right) = \imath_n \otimes (j)_m$

$$\mathrm{diag}\left(f\right) \mathrm{S}_w = \mathrm{S}_w\, \mathrm{diag}\left(f \circ w\right)$$

Loop Fusion Beyond Cooley-Tukey

rather cheap

Main DFT recursion (breakdown rules):

$$\mathrm{DFT}_{km} \rightarrow (\mathrm{DFT}_k \otimes \mathrm{I}_m)\, \mathrm{T}^n_m (\mathrm{I}_k \otimes \mathrm{DFT}_m)\, \mathrm{L}^n_k$$

expensive

$$\mathrm{DFT}_{km} \rightarrow P_n (\mathrm{DFT}_k \otimes \mathrm{DFT}_m) Q_n, \quad \gcd(k,m) = 1$$

$$\mathrm{DFT}_p \rightarrow R^T_p (\mathrm{I}_1 \oplus \mathrm{DFT}_{p-1}) D_p (\mathrm{I}_1 \oplus \mathrm{DFT}_{p-1}) R_p$$

very expensive

$$\mathrm{DFT}_n \rightarrow D_m\, \mathrm{DFT}_m\, D'_m\, \mathrm{DFT}_m\, D''_m, \quad m > 2n$$

How to fuse permutations from different combinations of rules?

Example

- Consider the DFT formula fragment

$$(\mathrm{I}_p \otimes (\mathrm{I}_1 \oplus (\mathrm{I}_r \otimes \mathrm{DFT}_s) L^{rs}_r)\, W_p)\, V_{p,q}$$

Cooley-Tukey Rader Good-Thomas

- In Σ-SPL:

$$\sum_{j_1=0}^{p-1} \Big(\mathrm{S}_{((j_1)_p \otimes \imath_q)\circ(0)_+^{1\to q}\circ \imath_1}\, \mathrm{G}_{\mathrm{v}^{p,q}\circ((j_1)_p\otimes\imath_q)\circ\overline{w}^q_{1,g}\circ(0)_+^{1\to q}}$$

$$+ \sum_{j_0=0}^{r-1} \mathrm{S}_{((j_1)_p\otimes\imath_q)\circ(1)_+^{q-1\to q}\circ((j_0)_r\otimes\imath_s)}\, \mathrm{DFT}_s$$

Complicated array access

$$\underline{\mathrm{G}_{\mathrm{v}^{p,q}\circ((j_1)_p\otimes\imath_q)\circ\overline{w}^q_{1,g}\circ(1)_+^{q-1\to q}\circ\ell^{rs}_r\circ((j_0)_r\otimes\imath_s)}}\Big).$$

Example (cont'd)

- **After index function simplification:**

$$\sum_{\substack{j_1=0\\ b_1=qj_1}}^{p-1} \left(\mathrm{S}_{h_{0,q}^{p\rightarrow pq}\circ(j_1)_p} \, \mathrm{G}_{\hbar_{0,q}^{p\rightarrow pq}\circ(j_1)_p} + \right.$$

$$\left. \sum_{\substack{j_0=0\\ \phi_1=g^{j_0}}}^{r-1} \mathrm{S}_{h_{qj_1+sj_0+1,1}^{s\rightarrow pq}} \, \mathrm{DFT}_s \, \mathrm{G}_{\underline{\hbar_{b_1,p}^{q\rightarrow pq}\circ w_{\phi_1,g^s}^{s\rightarrow q}}} \right)$$

Simplified array access

Example (cont'd)

- **Generated Code**

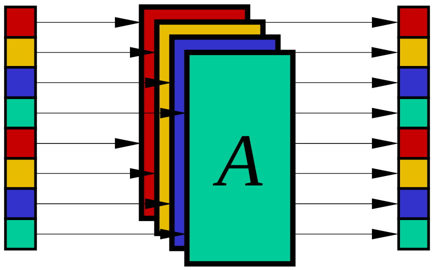
```
// Input: _Complex double x[28], output: y[28]
int p1, b1;
for(int j1 = 0; j1 <= 3; j1++) {
   y[7*j1] = x[(7*j1%28)];
   p1 = 1; b1 = 7*j1;
   for(int j0 = 0; j0 <= 2; j0++) {
      y[b1 + 2*j0 + 1] =
         x[(b1 + 4*p1)%28] + x[(b1 + 24*p1)%28];
      y[b1 + 2*j0 + 2] =
         x[(b1 + 4*p1)%28] - x[(b1 + 24*p1)%28];
      p1 = (p1*3%7);
   }
}
```

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)

Formula manipulation overcomes compiler limitations

Example DFT

Standard FFT

$$(\mathbf{DFT}_k \otimes \mathrm{I}_m)\, \mathsf{T}^n_m (\mathrm{I}_k \otimes \mathbf{DFT}_m)\, \mathsf{L}^n_k$$

Formula manipulation

$$\left(\mathrm{I}_{\frac{mn}{\nu}} \otimes \mathsf{L}^{2\nu}_{\nu}\right) \left(\overline{\mathbf{DFT}_m \otimes \mathrm{I}_{\frac{n}{\nu}}} \otimes \mathrm{I}_\nu\right) \overline{\overline{\mathsf{T}}}'^{mn}_{n}$$
$$\left(\mathrm{I}_{\frac{m}{\nu}} \otimes \left(\mathsf{L}^{2n}_{\nu} \otimes \mathrm{I}_\nu\right) \left(\mathrm{I}_{\frac{2n}{\nu}} \otimes \mathsf{L}^{\nu^2}_{\nu}\right) \left(\overline{\mathbf{DFT}_n} \otimes \mathrm{I}_\nu\right)\right) \left(\mathsf{L}^{\frac{mn}{\nu}}_{\frac{m}{\nu}} \otimes \mathsf{L}^{2\nu}_{2}\right)$$

Vector FFT for ν-way vector instructions

Implementation of Formula Generation and Manipulation

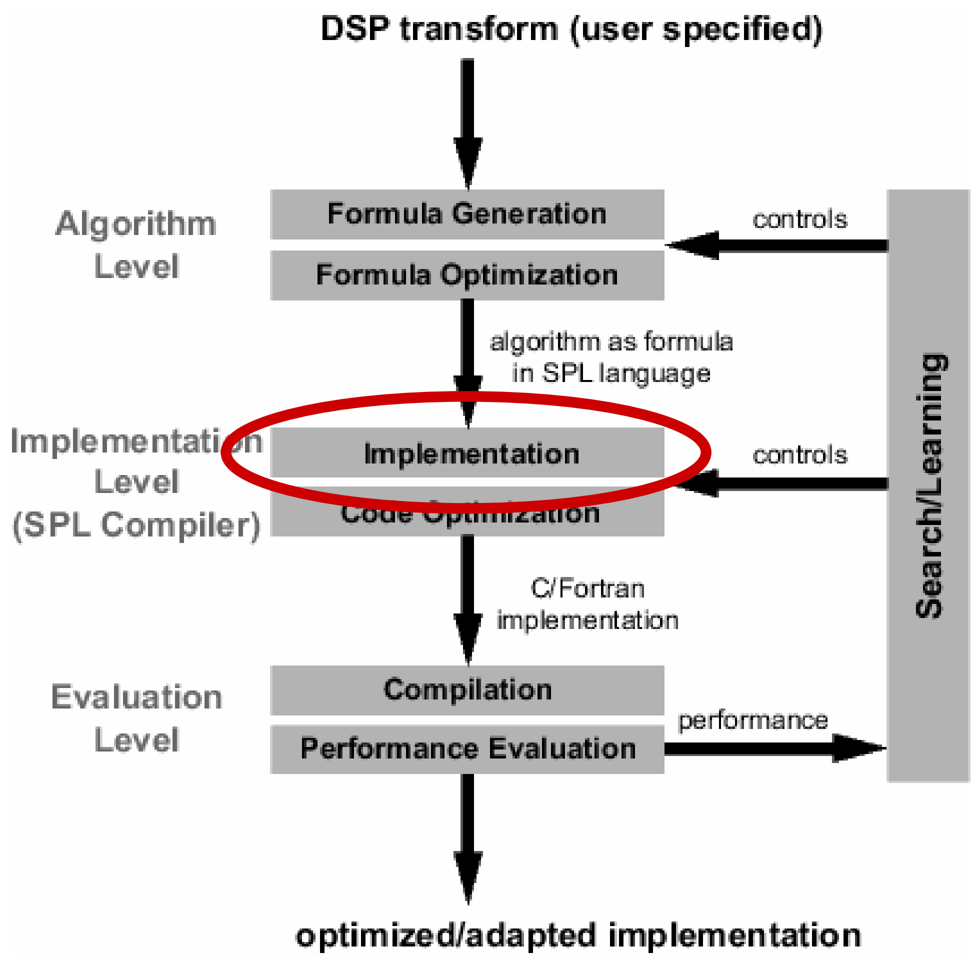
- Implementation using a computer algebra system (GAP)
- SPL/Σ-SPL implemented as recursive data types
- Exact representation of sin(), cos(), etc.
- Symbolic computation enables exact verification of rules

From Optimized Algorithm (Formula) to Code

Input:
Optimized formula

Output:
Intermediate Code

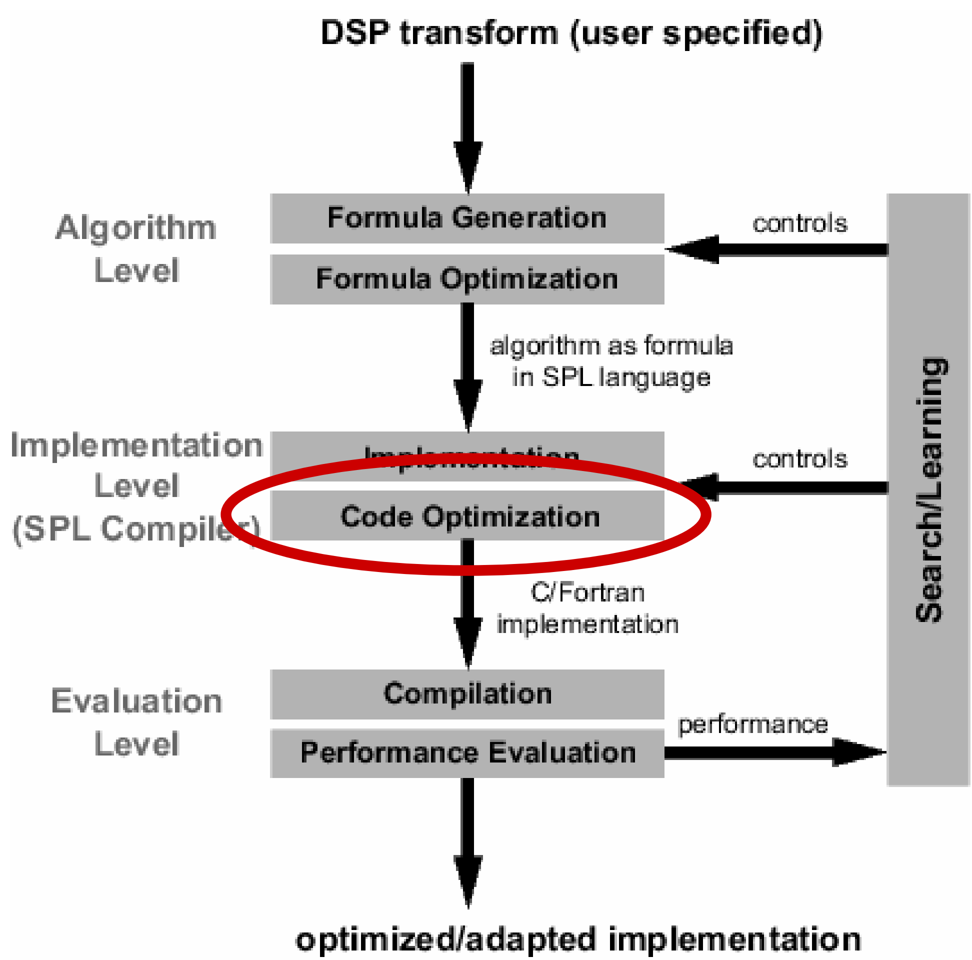
Straightforward



From Code to Optimized Code

Input:
Intermediate Code

Output:
Optimized C code



Code Level Optimizations

- Precomputation of constants
- Loop unrolling (controlled by search module)
- Constant inlining
- SSA code, scalar replacement, algebraic simplifications, CSE
- Code reordering for locality (optional)
- Conversion to FMA code (optional)
- Conversion to fixed point code (optional)
- Conversion to multiplierless code (optional)

- Finally: Unparsing to C (or Fortran)

Conversion to FMA code

- FMA (fused multiply-add) or MAC (multiply accumulate) instructions: $y = \pm ax \pm b$
- Extension of the instruction set + specialized execution units
- As fast as a single add or multiply
- Conversion of linear algorithms to FMA code: blackboard
- Paper: Yevgen Voronenko and Markus Püschel
  Automatic Generation of Implementations for DSP Transforms on Fused Multiply-Add Architectures
  Proc. (ICASSP) 2004

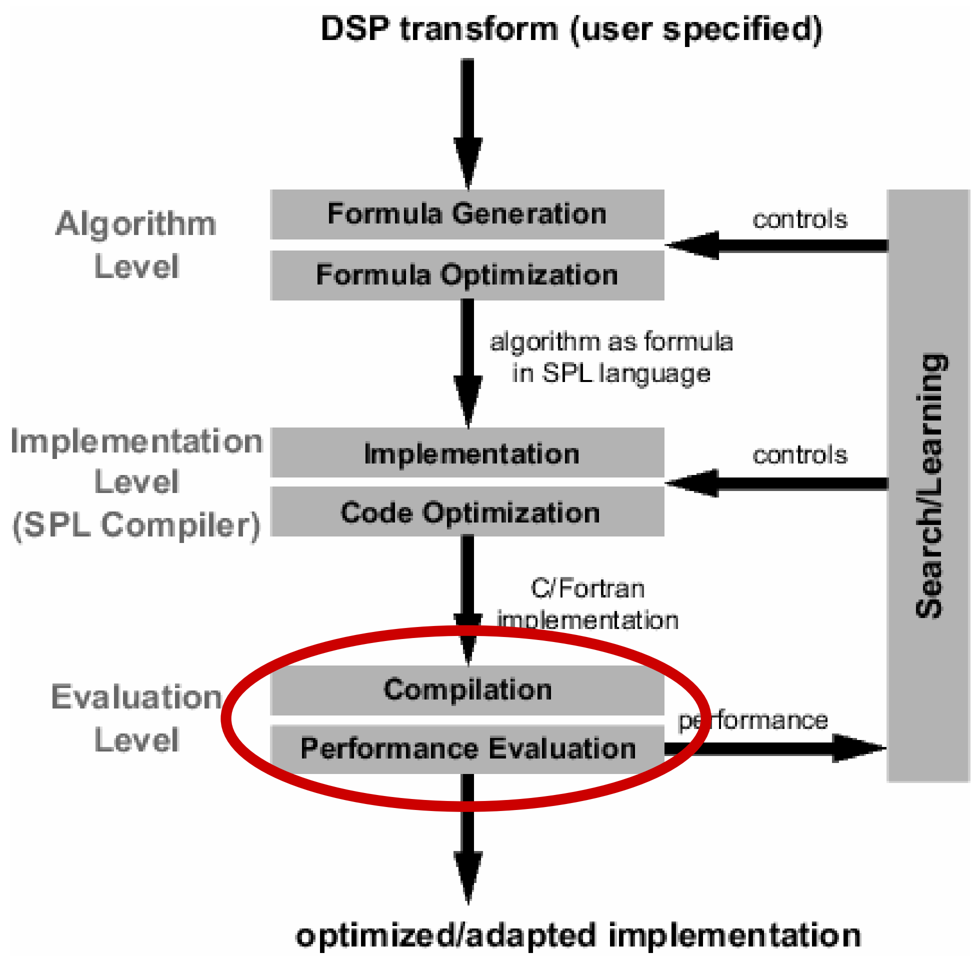
Evaluating Code

Input:
 Optimized C code

Output:
 Performance Number

Straightforward

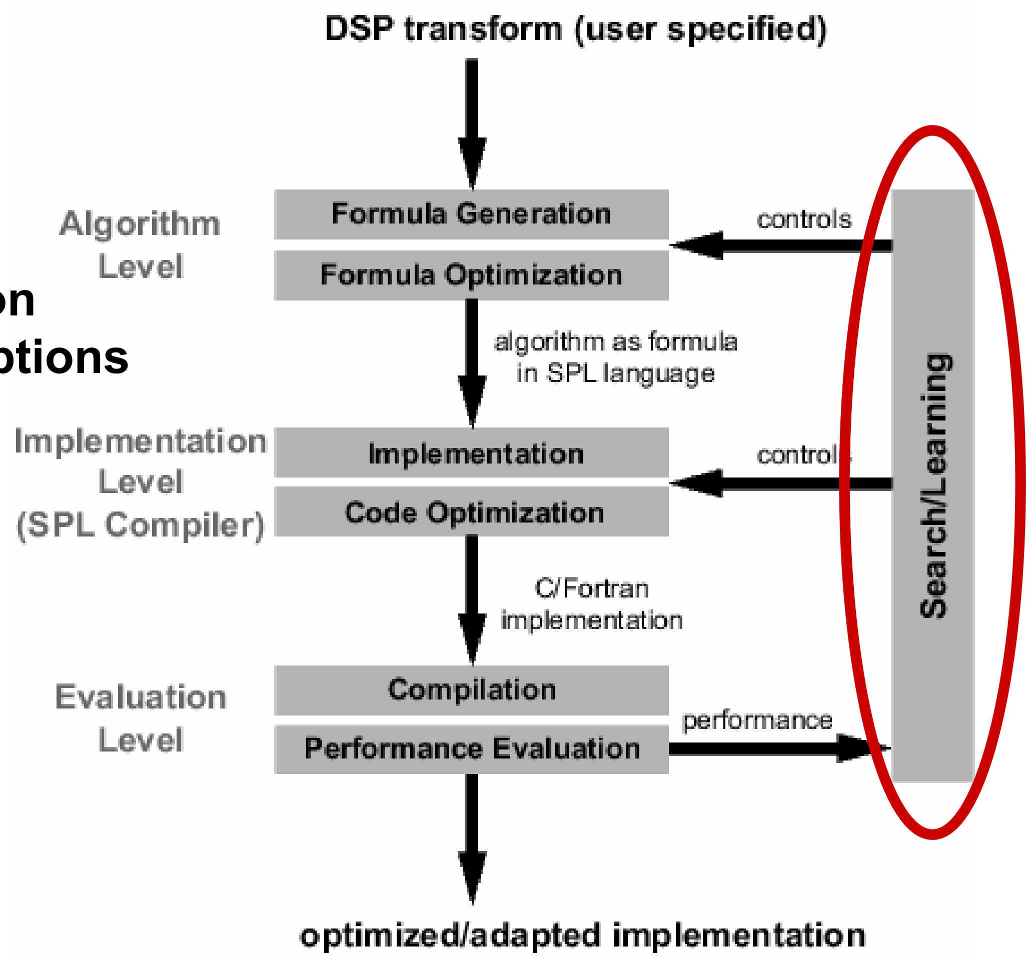
Examples:
- **runtime**
- **accuracy**
- **operations count**



Search (Learning) for the Best

Input:
Performance Number

Output:
Controls Formula Generation
Controls Implementation Options



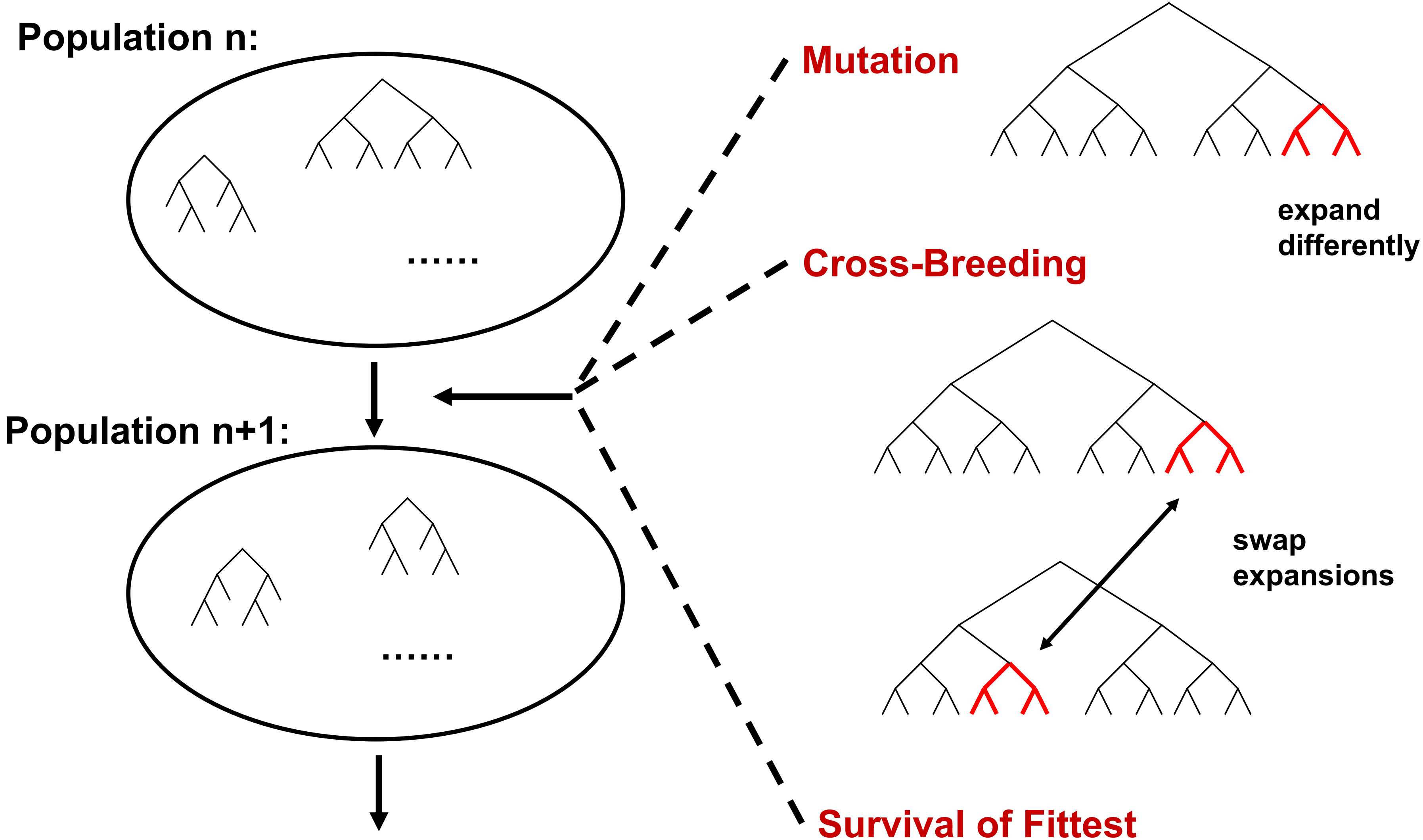
Search Methods

- **Search over:**
  - Algorithmic degrees of freedom
    (choice of breakdown rules)
  - Implementation degrees of freedom
    (degree of unrolling)

- **Operates with the ruletree representation of an algorithm**
  - transform independent
  - efficient

- **Search Methods**
  - Exhaustive Search
  - Dynamic Programming (DP)
  - Random Search
  - Hill Climbing
  - STEER (an evolutionary algorithm)

STEER: Evolutionary Search



Learning

- **Procedure:**
  - Generate a set of (1000 say) algorithms and their runtimes (one transform, one size); represent algorithms by features
  - From this data (pairs of features and runtimes), learn a set of algorithm design rules
  - From this set, generate best algorithms (theory of Markov decision processes)

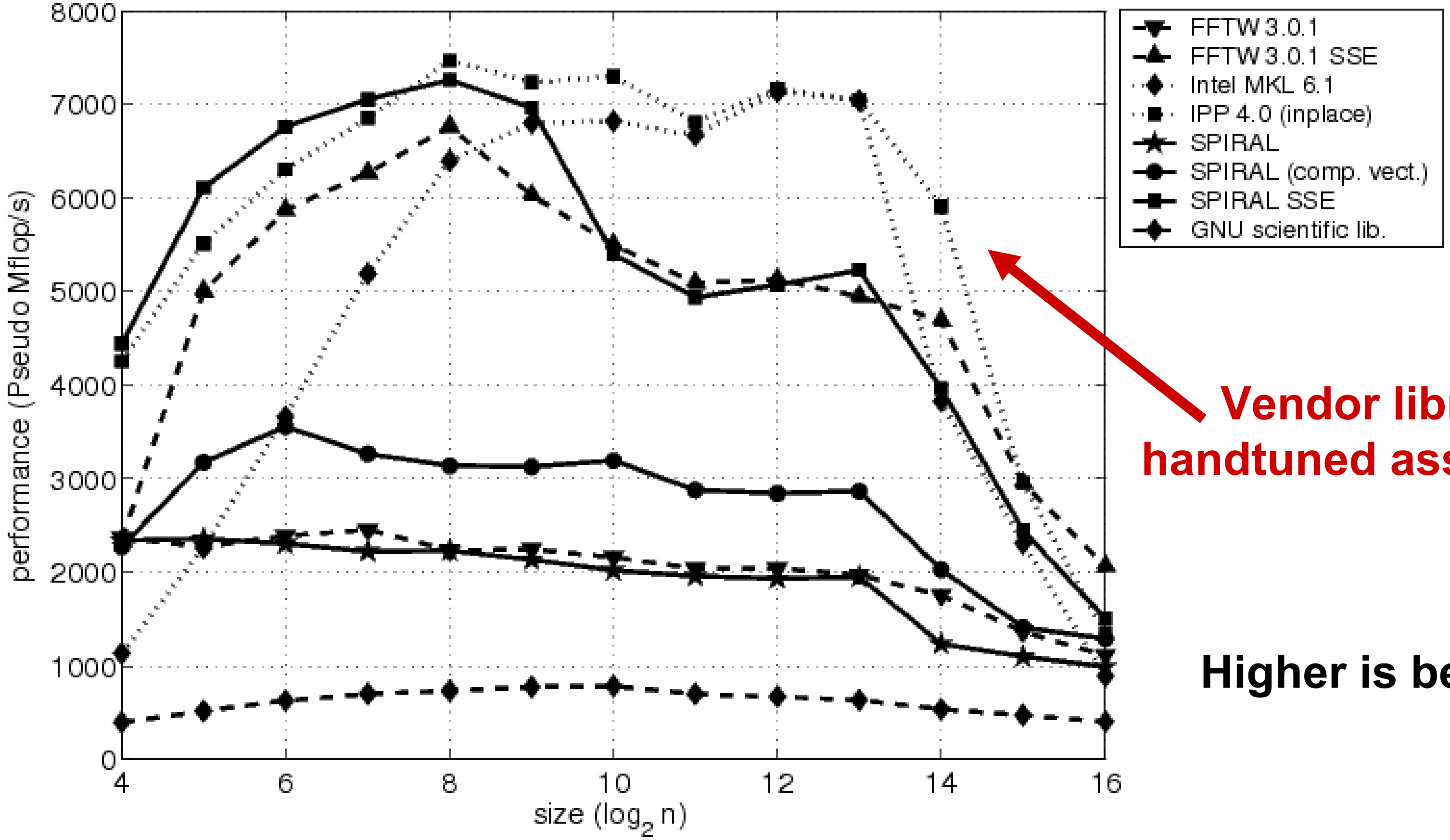
- **Evaluation:**
  - Tested for WHT and DFT
  - From data generated for one size (2^15) could construct best algorithms across sizes (2^12-2^18)

Bryan Singer and Manuela Veloso
Learning to Construct Fast Signal Processing Implementations
Journal of Machine Learning Research, 2002, Vol. 3, pp. 887-919

Benchmarks

Benchmark: DFT, 2-powers

P4, 3.2 GHz, icc 8.0



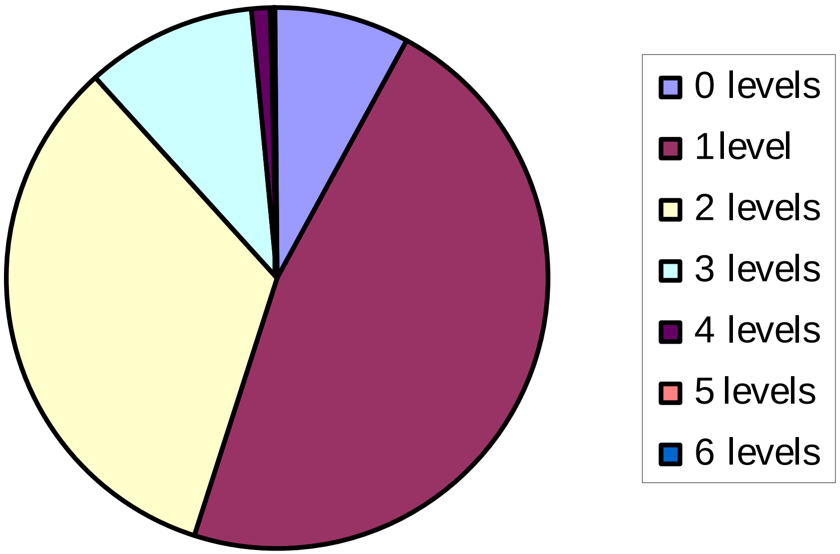
Vendor library: handtuned assembly?

Higher is better

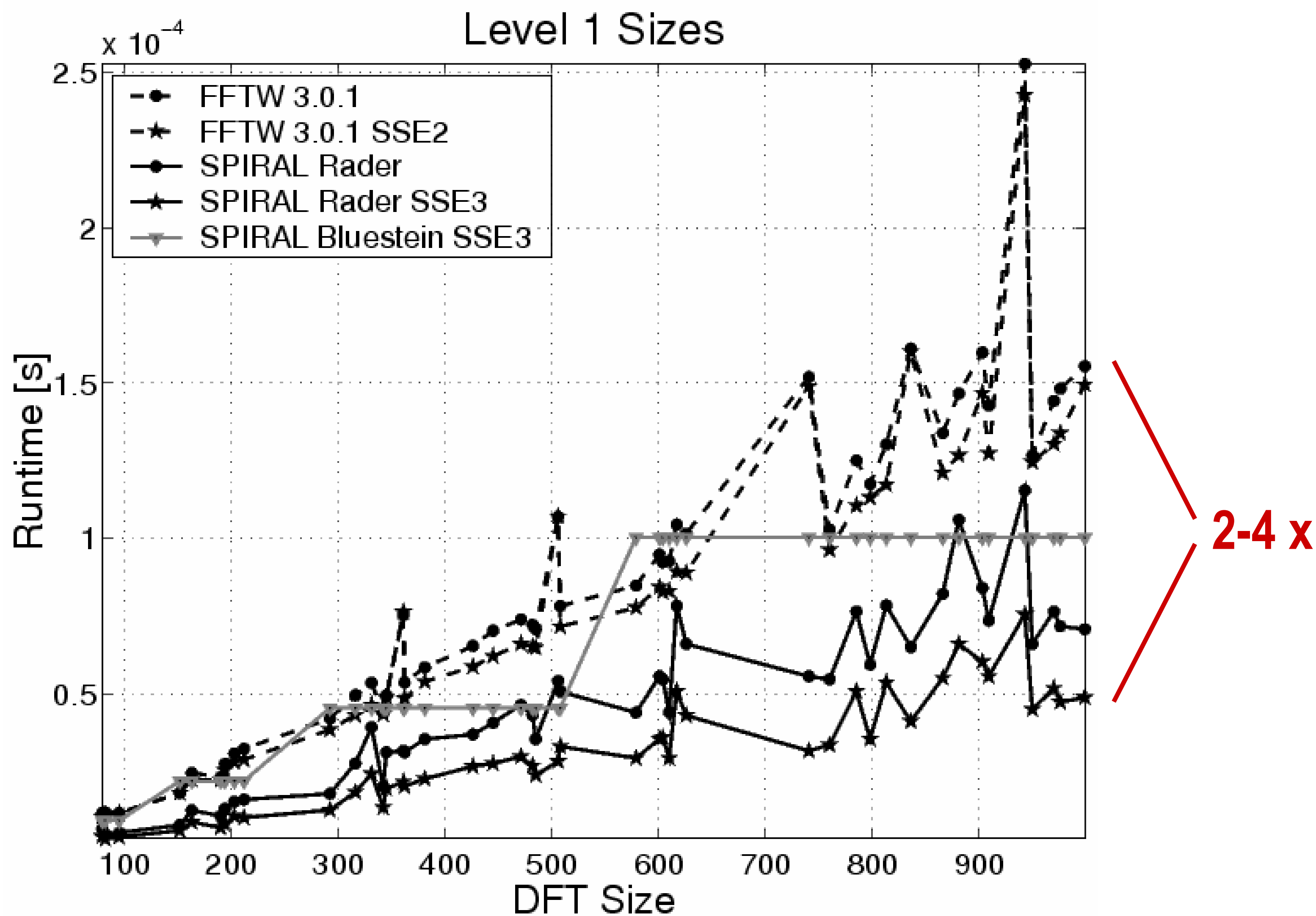
Single precision

Benchmark: DFT, Other Sizes

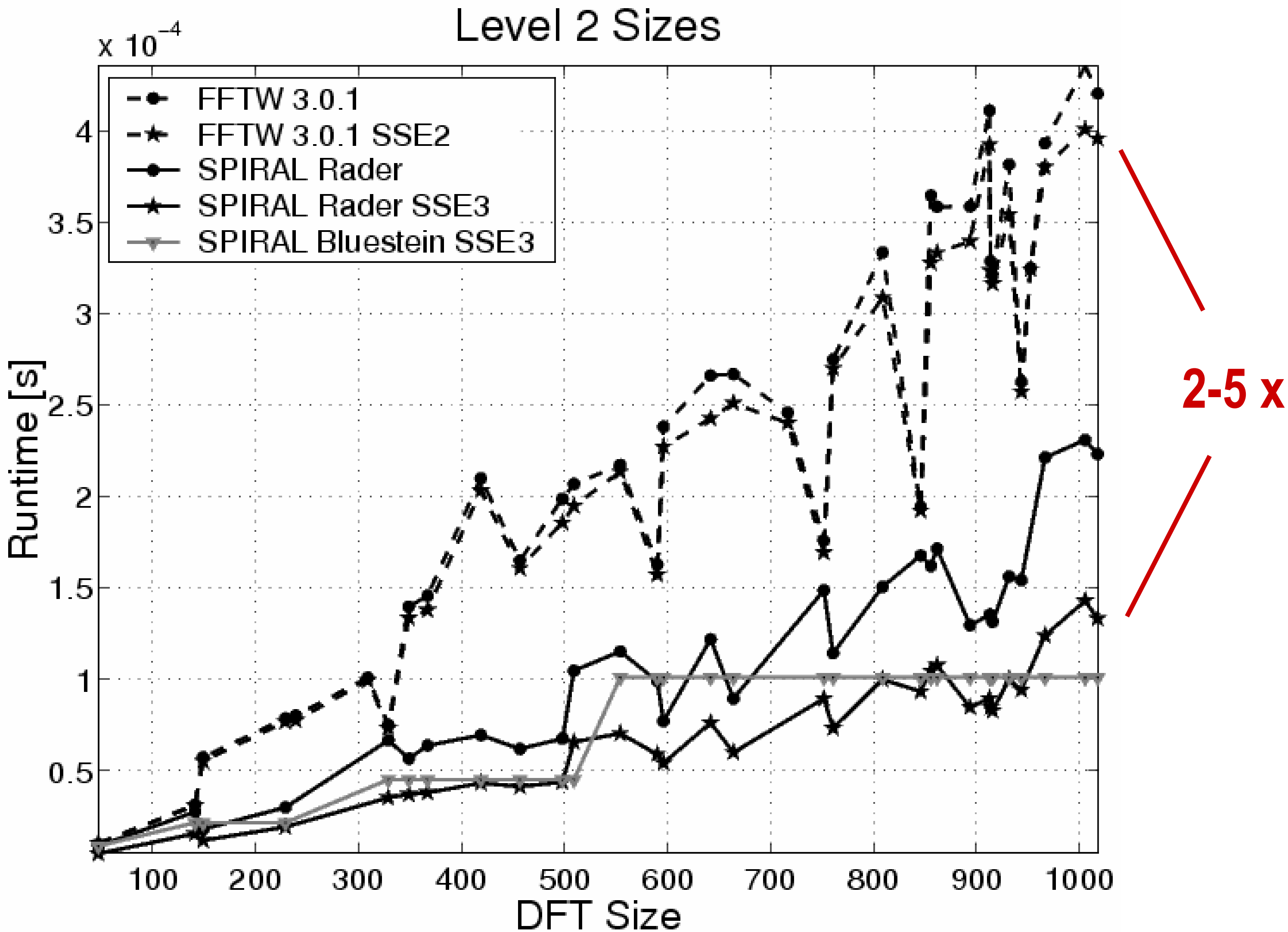
- Divide sizes into levels by number of necessary Rader steps
- n < 8192



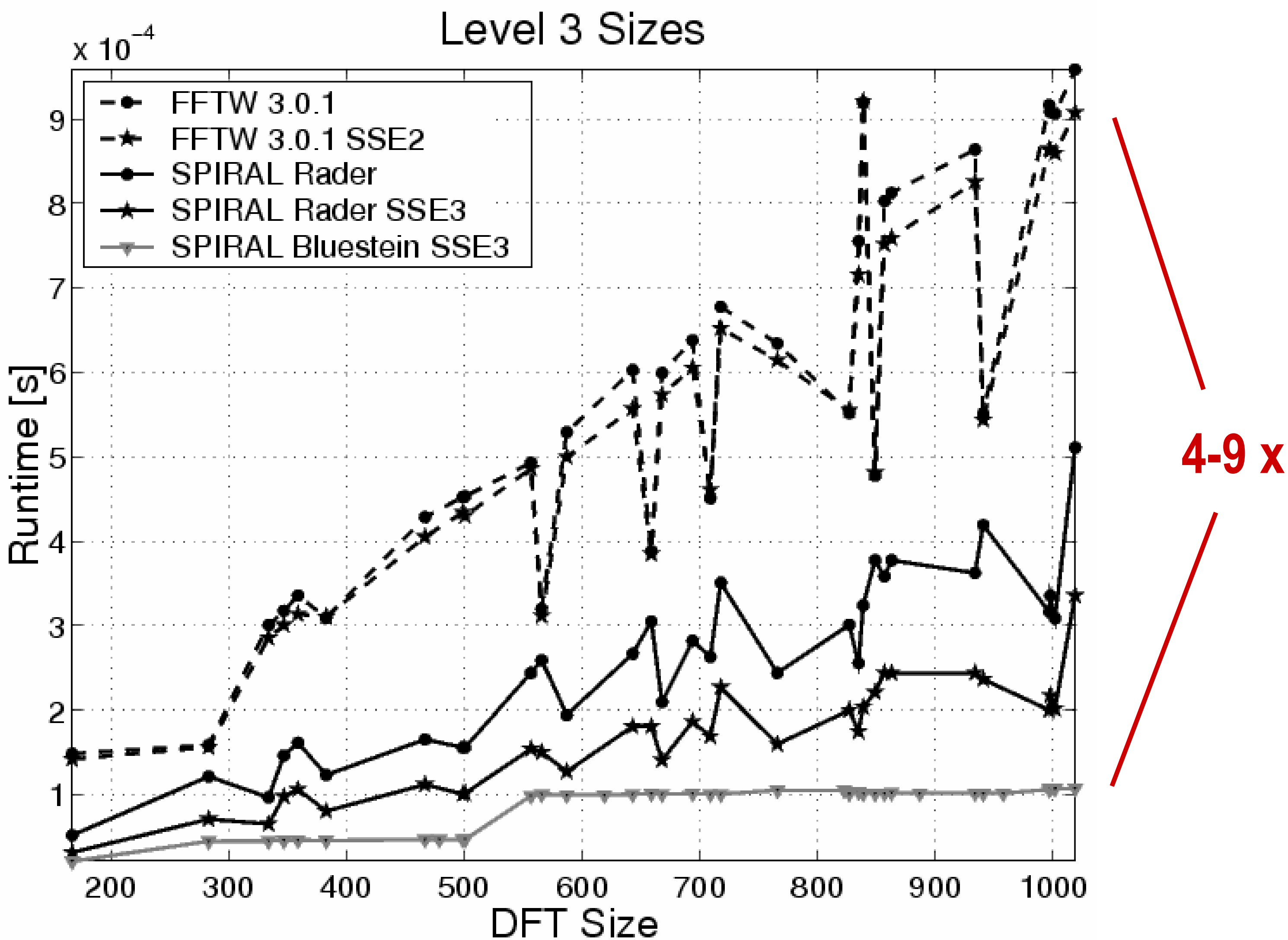
Benchmark: DFT, Level 1 Sizes



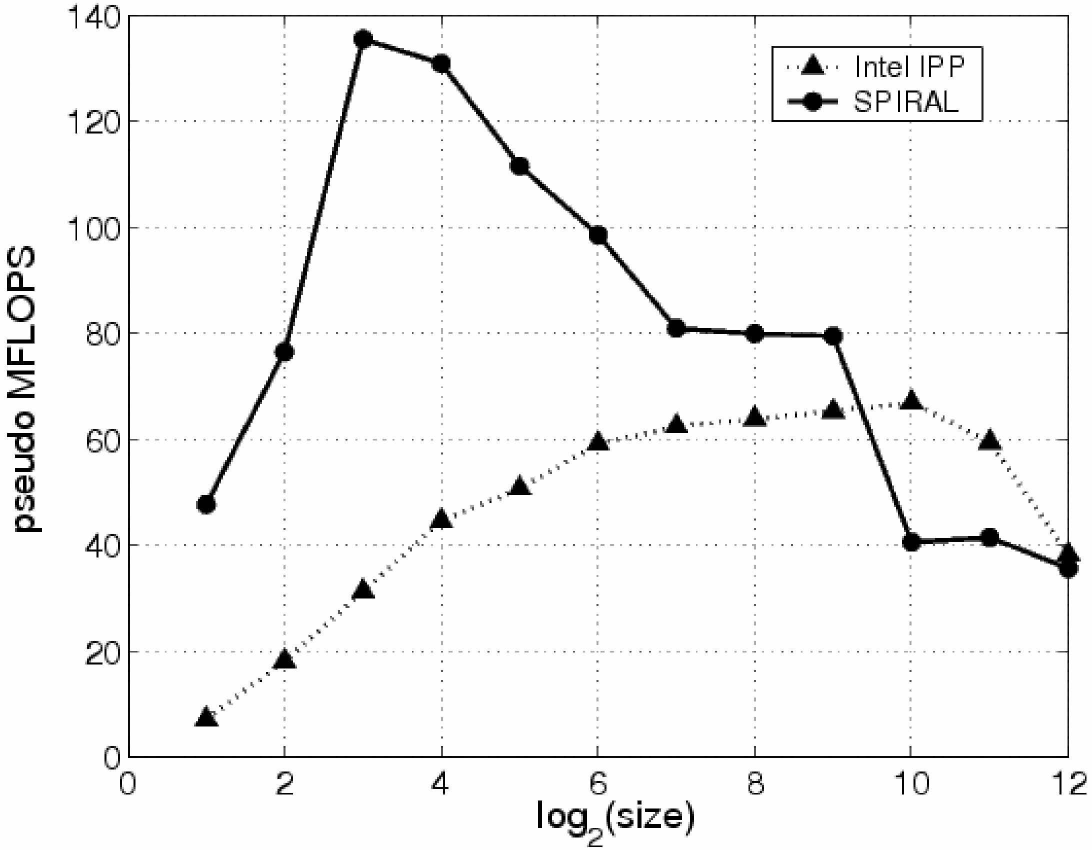
Benchmark: DFT, Level 2 Sizes



Benchmark: DFT, Level 3 Sizes



Benchmark: Fixed Point DFT, IPAQ



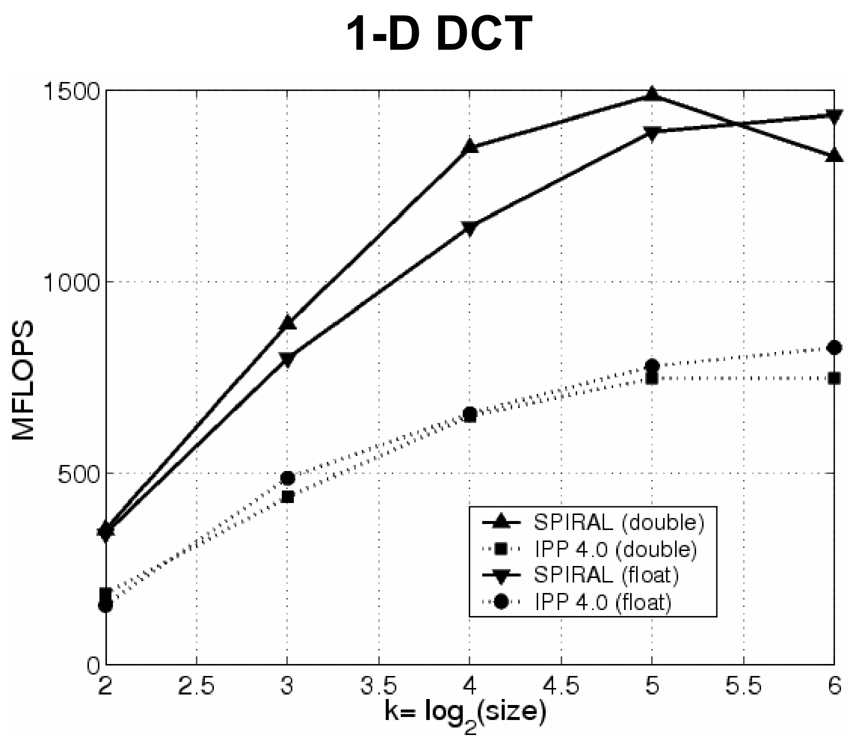
IPAQ
Xscale arch.
400 MHz
Has only fixed point

Higher is better

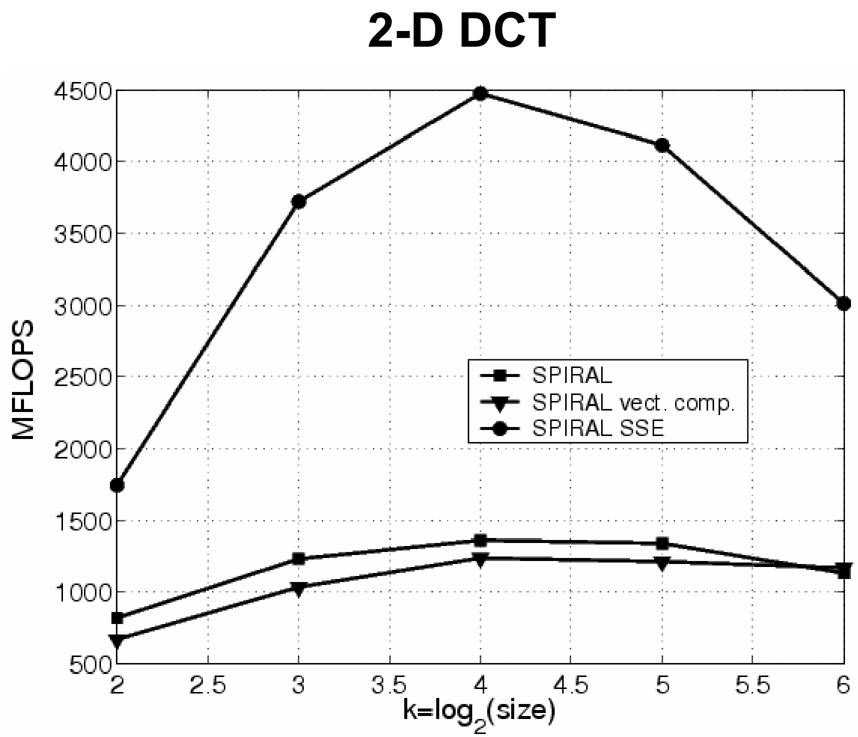
Intel spent less effort?

Benchmark: DCT

P4, 3.2 GHz,
icc 8.0



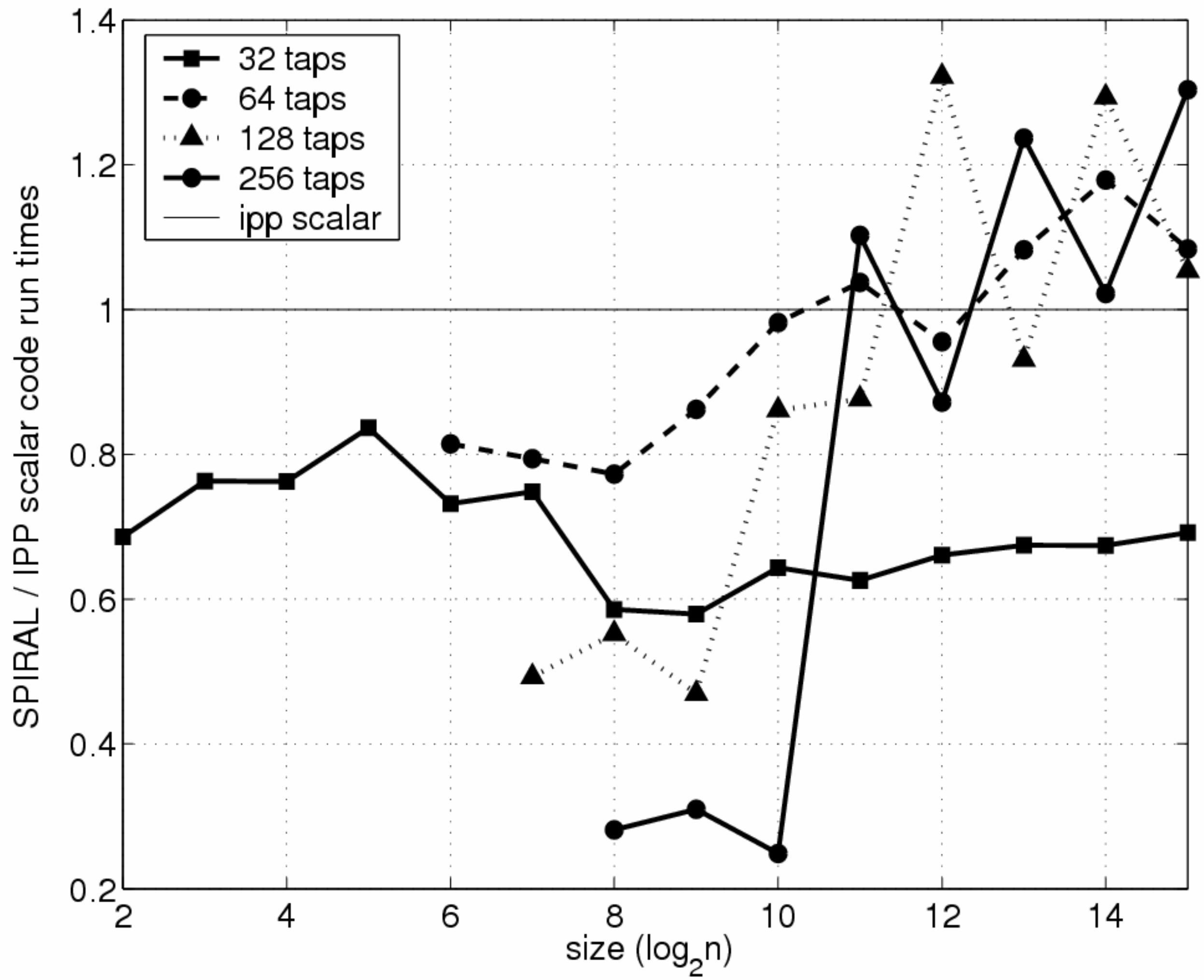
Scalar code



Scalar vs. SSE code

- This is not the latest IPP
- Spiral gains a factor of 2 to vendor library
- Another factor of 3 with 2D and vector instructions

Benchmark: Filter (Relative to IPP)



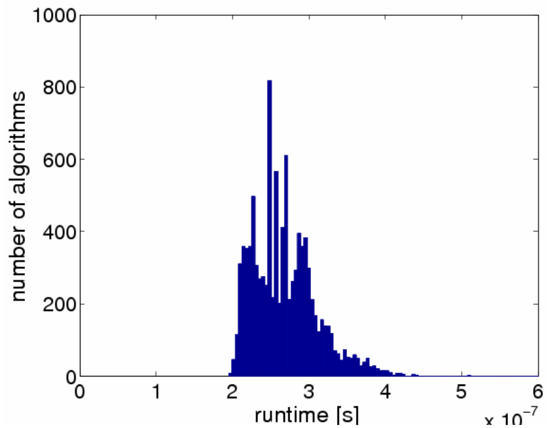
IPP

Lower = better

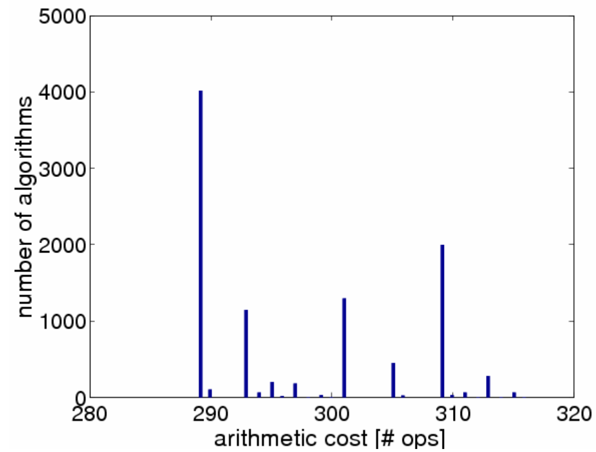
Instructive Experiments

Performance Spread: DCT, size 32
Histograms, 10,000 algorithms

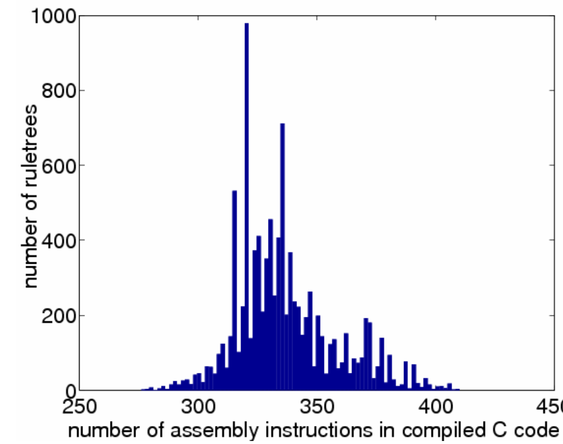
P4, 3.2 GHz, gcc



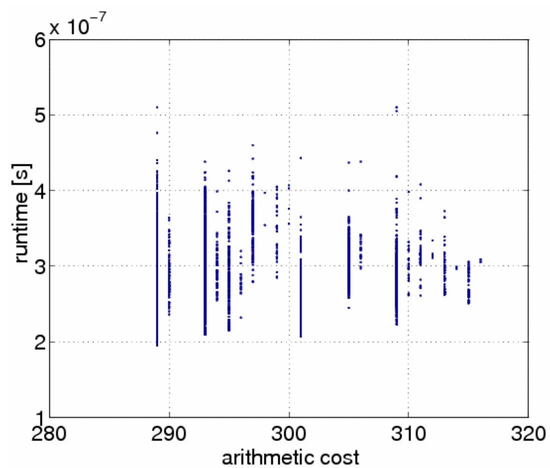
runtime: x2



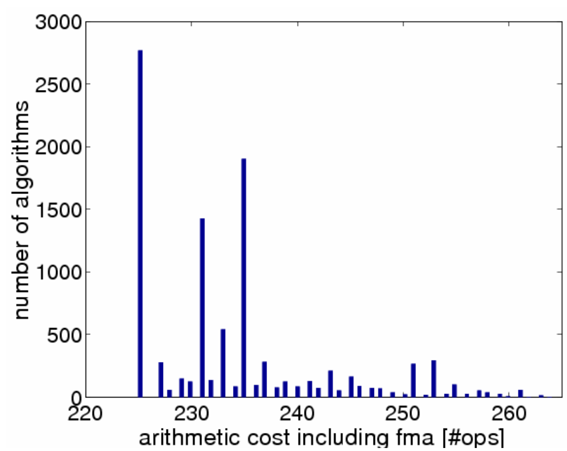
#ops: x1.08



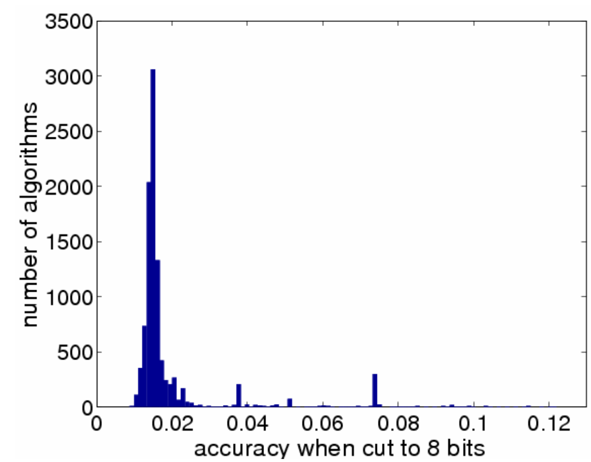
#assembly instr: x1.5



#ops vs. runtime: no correlation



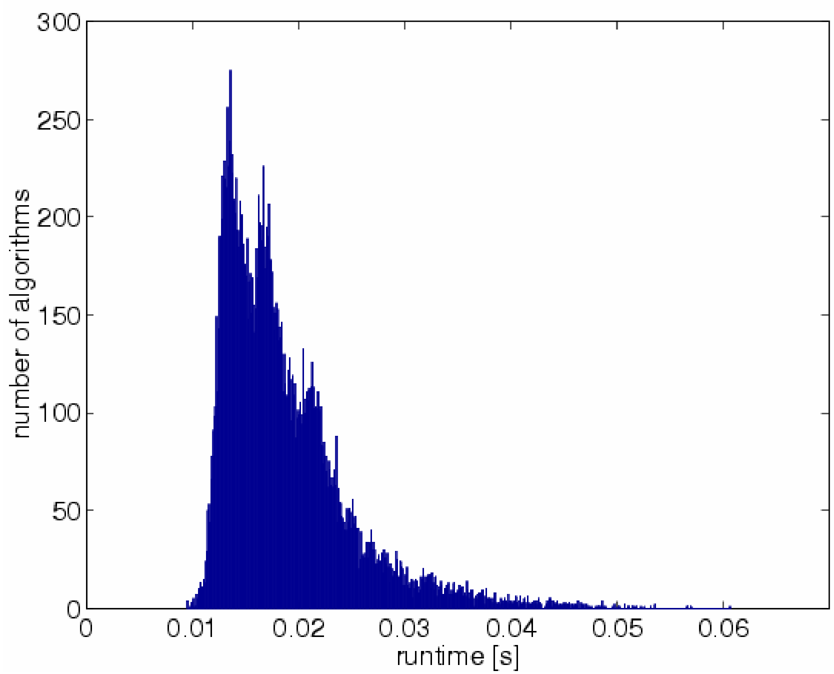
#fma ops: x1.2



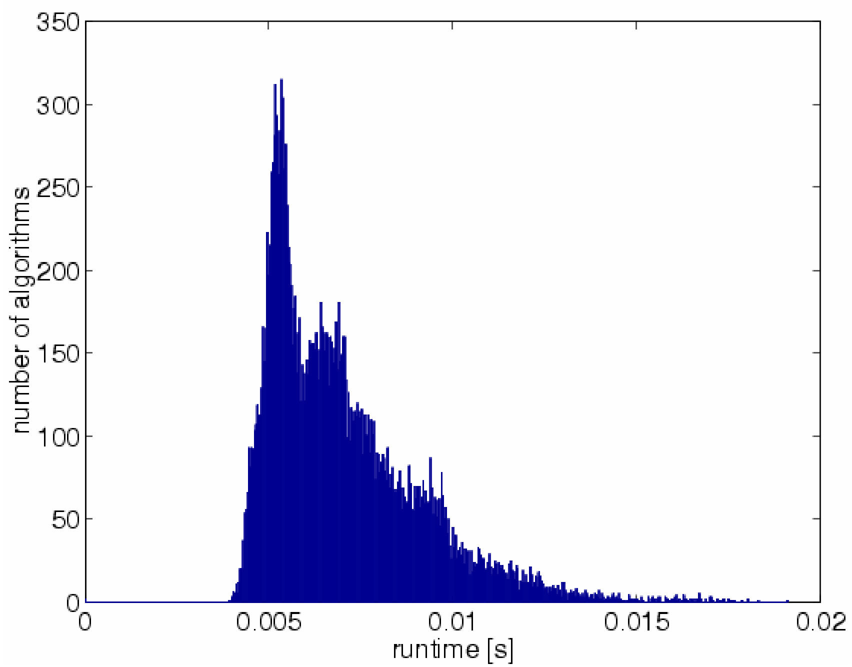
accuracy: x10, most x2

Performance Spread: DFT 2^16 Histograms, 20,000 Algorithms

P4, 3.2 GHz, icc 8.0



Generated scalar code

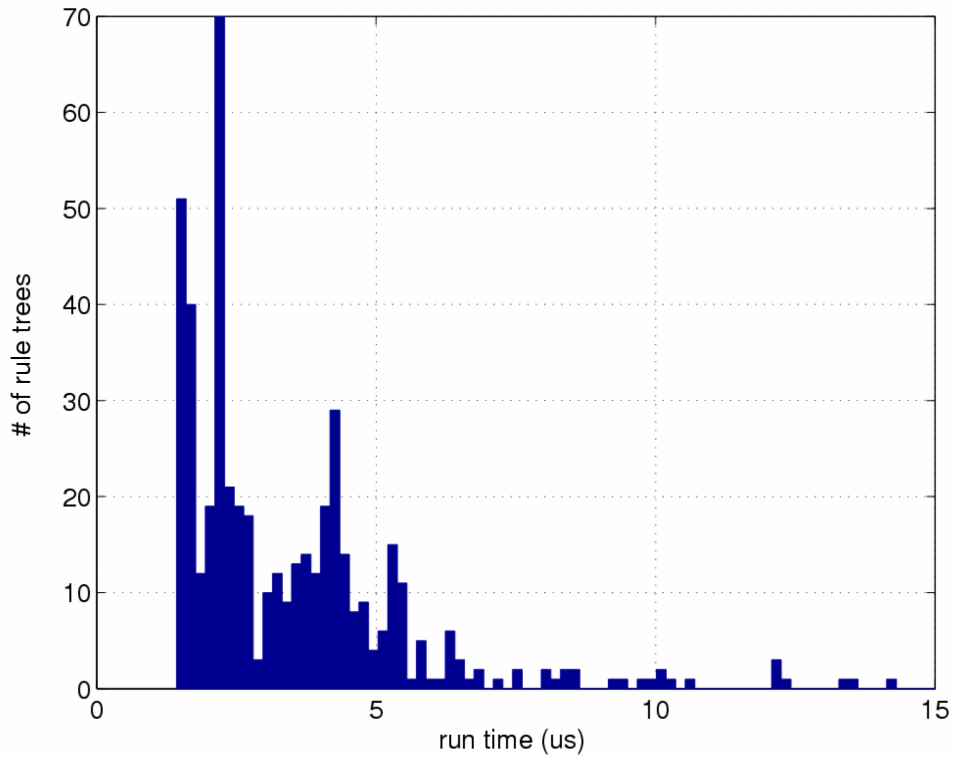


Generated SSE code (4-way vector single precision)

- **Generality of vectorization (all algorithms improve)**
- **Efficiency of vectorization (x 2.5 gain)**

Performance Spread: Filter(128, 16)

Pentium 4 – 3.2

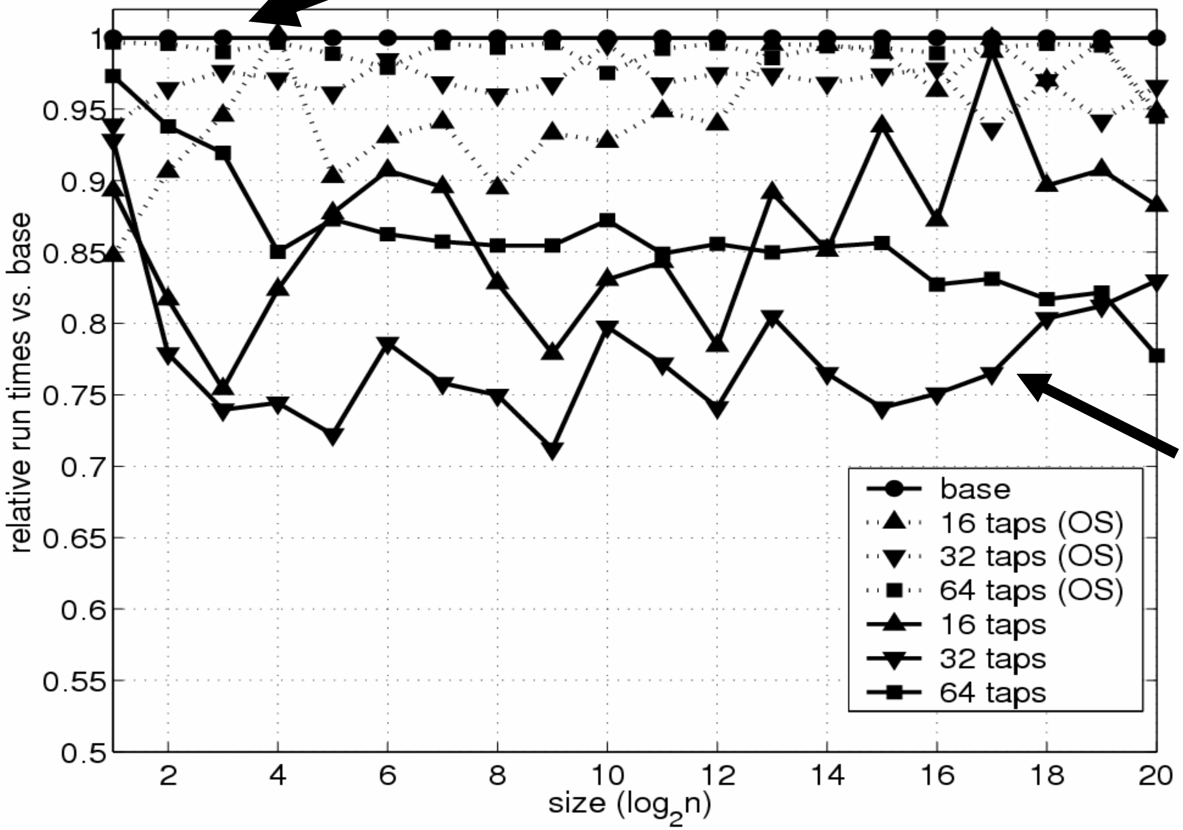


Filter: Time Domain Methods

$$\mathrm{Filt}_n(h(z)) \rightarrow \mathrm{I}_n \otimes_{k-1} (h_0, \ldots, h_{k-1})$$

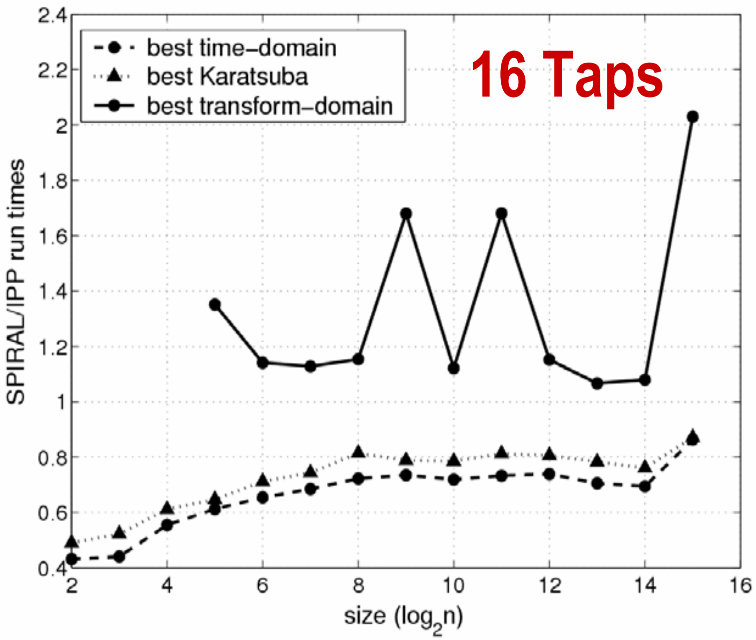
```
for i = 0..n-1
   y[i] = h[0]*x[0+i]+h[1]*x[1+i]+...+h[n-1]*x[n-1+i]
end
```

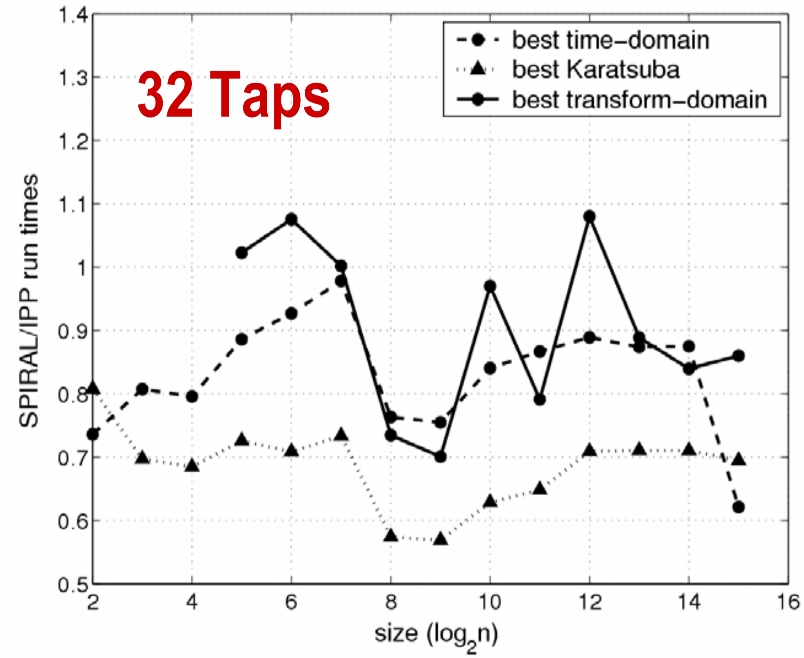
Xeon-1.7



best blocking strategy

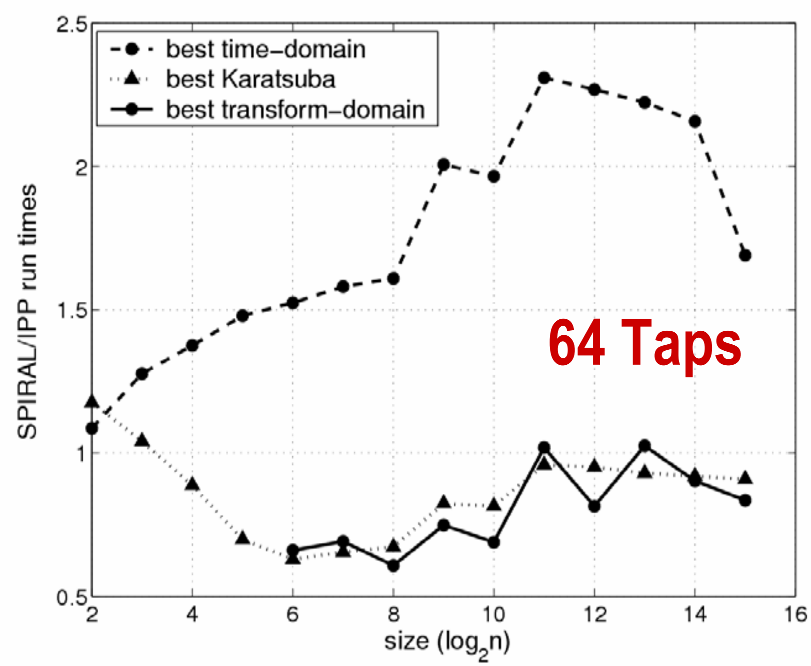
Filter: All Methods



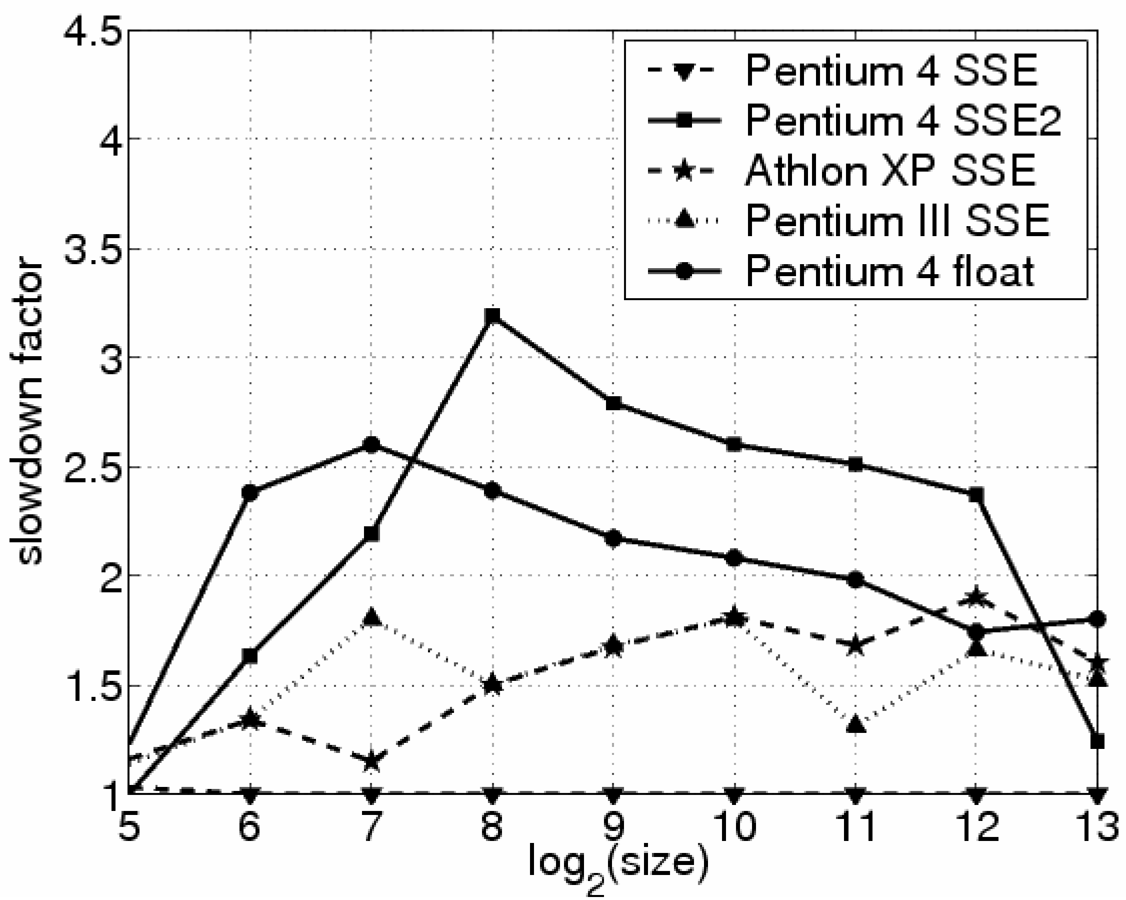


Athlon XP 1.73

- 16: Time domain wins
- 32: Karatsuba wins
- 64: Karatsuba/DFT ~equal



Platform Dependency: DFT



50% Loss by porting from PIII to P4

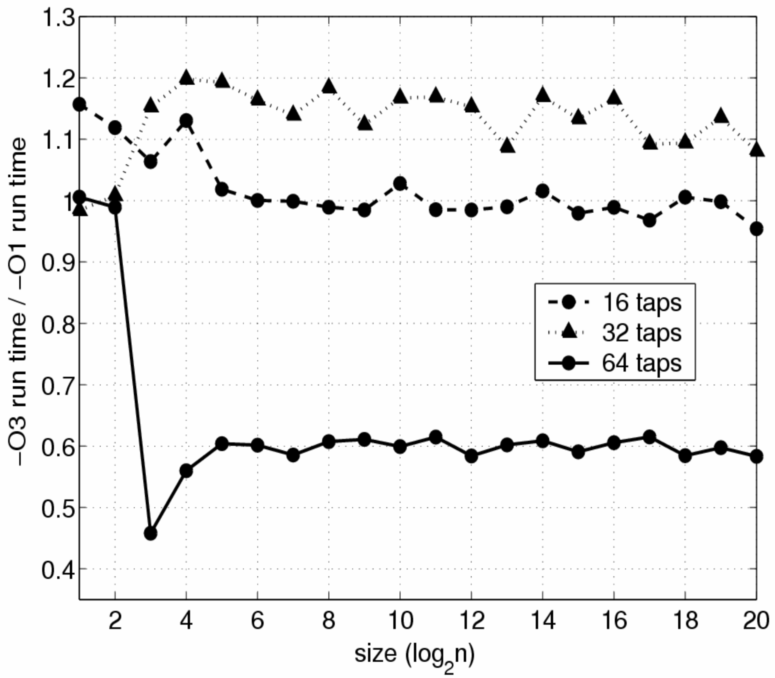
Platform Dependency: Filter

	16-tap	32-tap	64-tap	128-tap
Pentium 4 3.0GHz Northwood	Blocking	Karatsuba	RDFT	RDFT
Pentium 4 3.6GHz Prescott	Blocking	Karatsuba	Karatsuba	RDFT
Macintosh	Karatsuba	Karatsuba	RDFT	RDFT
Xeon 1.7 GHz	Blocking	Blocking	Blocking	RDFT
Athlon 1.73GHz	Karatsuba/ Blocking	Karatsuba	Karatsuba/ RDFT	RDFT

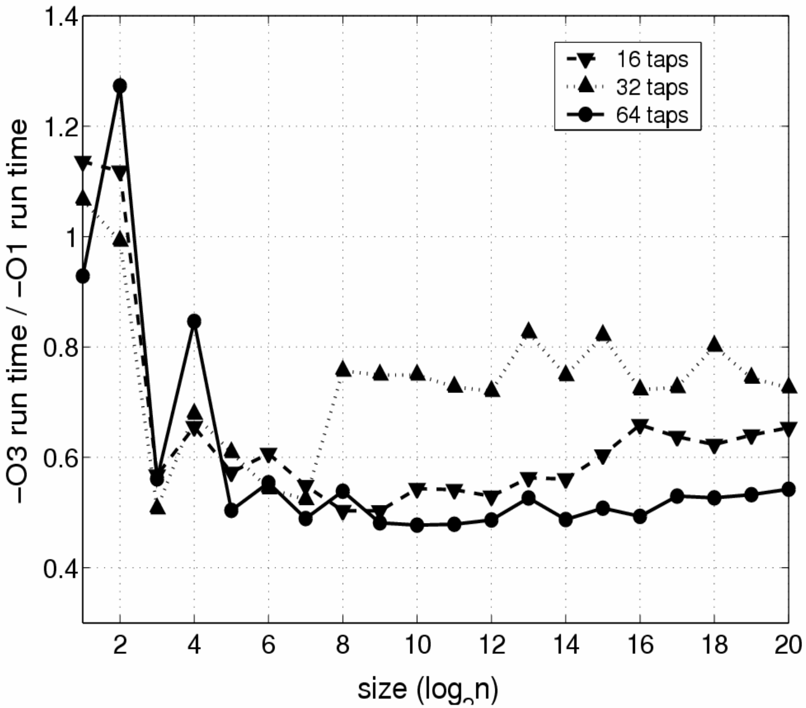
Compileroptions: Filter

Macintosh - GNU C 3.3 (Apple)

Blocking/nesting



+ Karatsuba

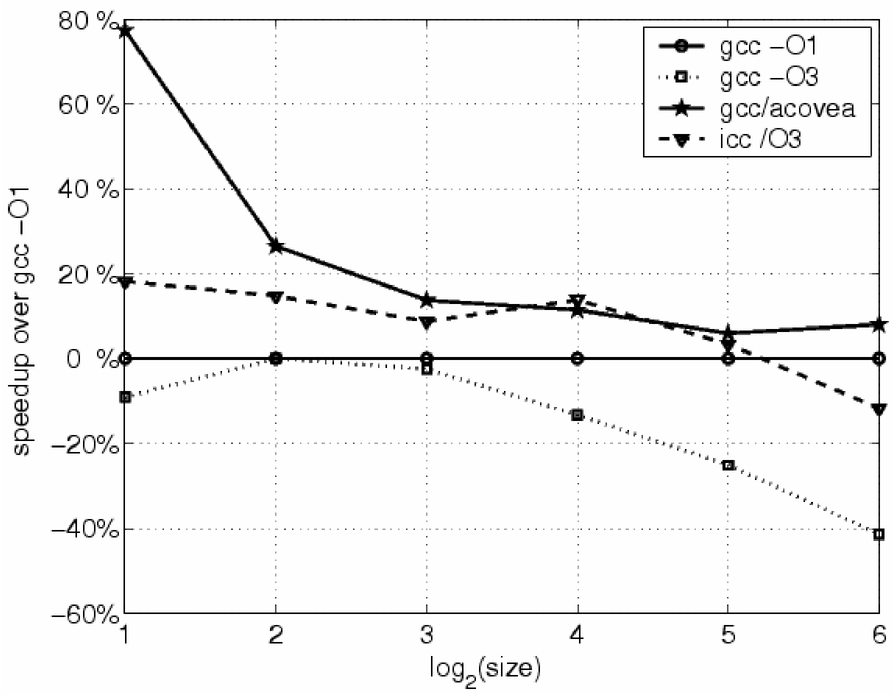


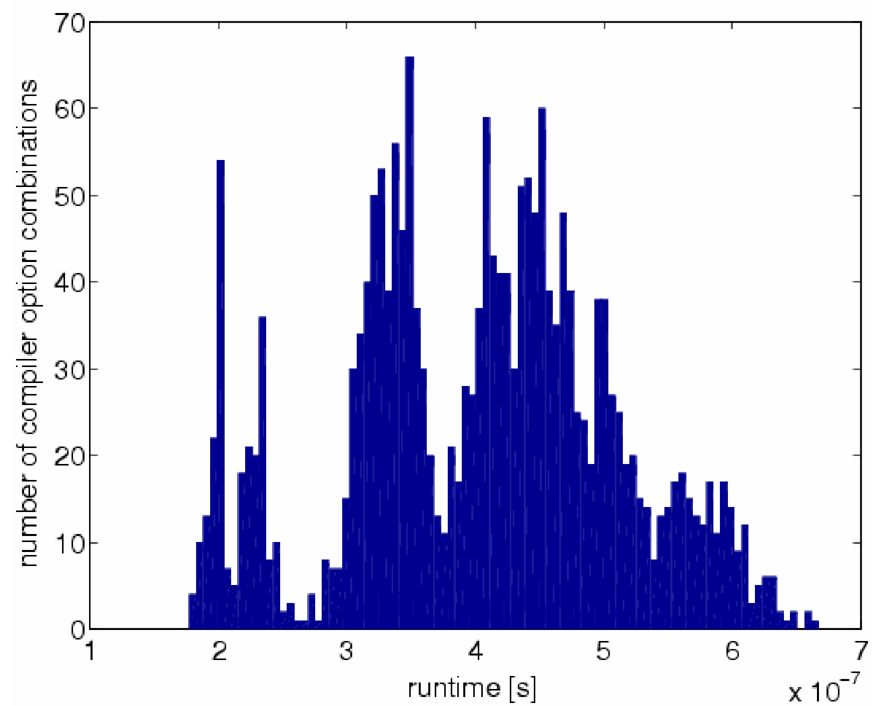

```
gcc {-O1/-O3} -fomit-frame-pointer -std=c99 -fast -mcpu=7450
```

Compileroptions DCT

P4, 3.2 GHz, gcc

ACOVEA: Evolutionary search for best compiler flags (gcc has ~500)

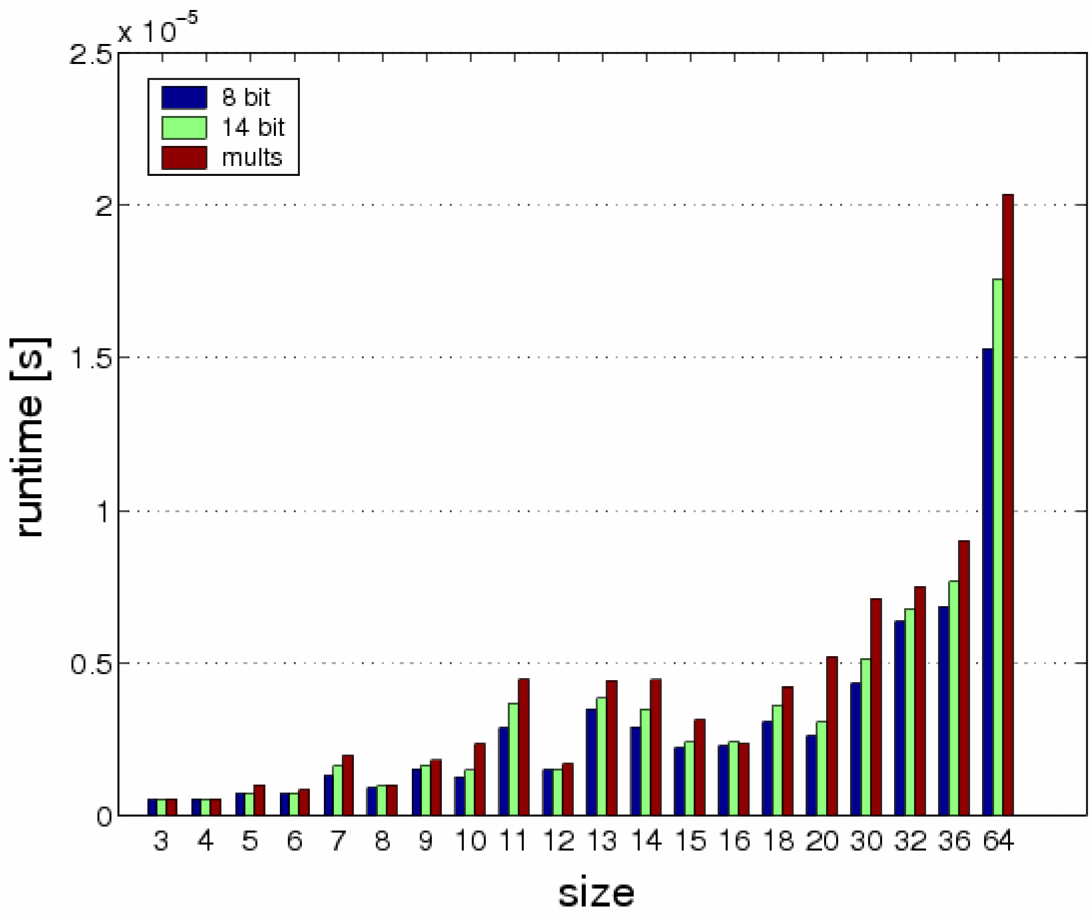




Runtime histogram
Random compiler flags
incl. -O1 –march=pentium4

10% improvement of best Spiral generated code

Multiplierless DFT, IPAQ



IPAQ
Xscale arch.
400 MHz
Fixed-point only

- **fixed-point constant multiplications replaced by adds and shifts**
- **trade-off runtime and precision**

Summary

- Code generation and tuning as optimization problem over the algorithm and implementation space

  Exploit the structure of the domain to solve it

- Declarative framework for computer representation of the domain-knowledge

  Enables algorithm generation and optimization

  (teaches the system the math; does not become obsolete?)

- Compiler to translate math description into code
- Search and learning to explore implementation space

  Closes the feedback loop

  gives the system "intelligence"

- Extensible, versatile

  Every step in the code generation is exposed

www.spiral.net