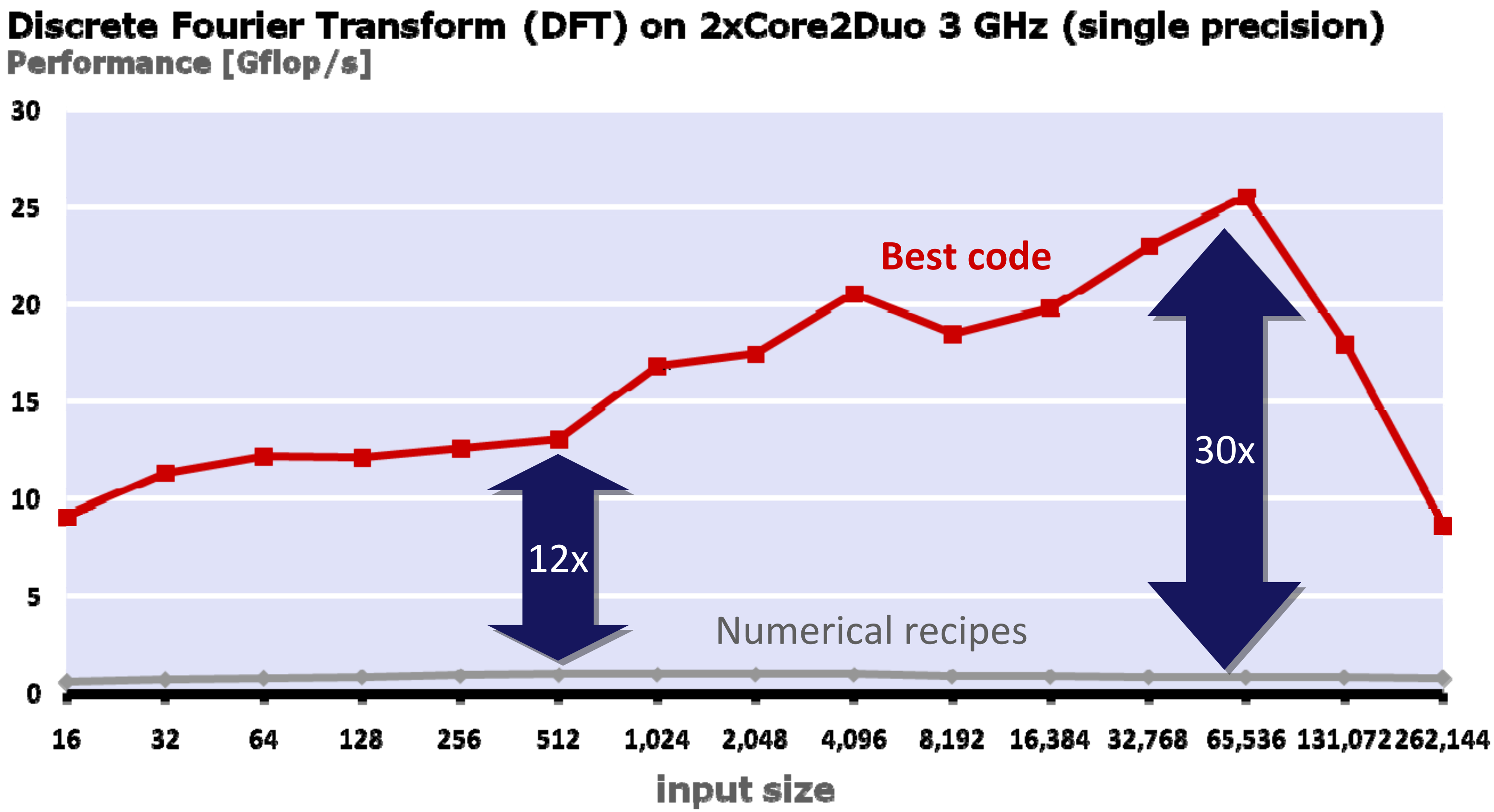
Generating High-Performance General Size Linear Transform Libraries Using Spiral

Yevgen Voronenko
Franz Franchetti
Frédéric de Mesmay
Markus Püschel
Carnegie Mellon University

HPEC, September 2008, Lexington, MA, USA

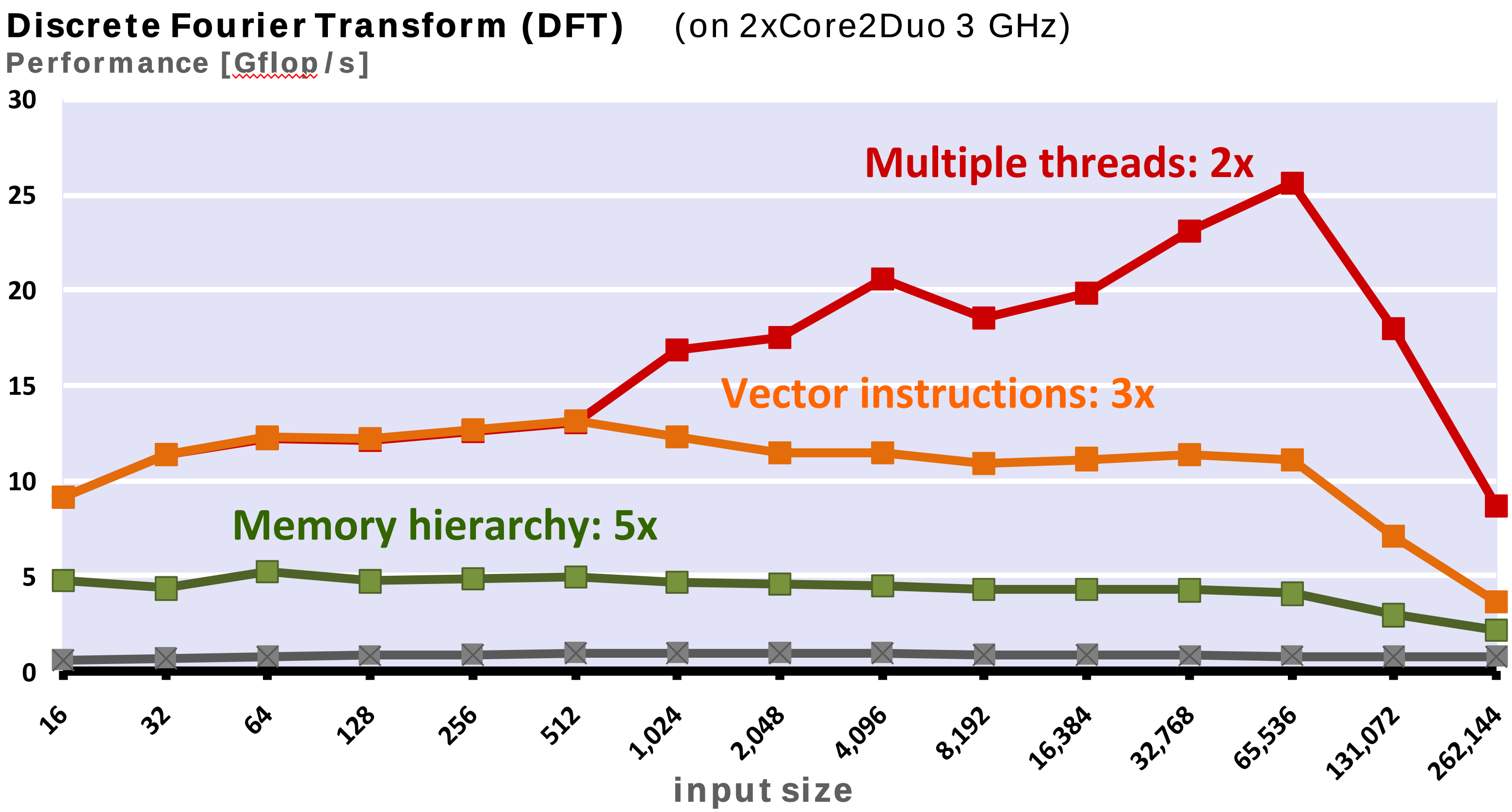
This work was supported by DARPA DESA program, NSF-NGS/ITR, NSF-ACR, and Intel

The Problem: Example DFT



- Standard desktop computer
- Same operations count $\approx 4n\log_2(n)$
- *Similar plots can be shown for all numerical problems*

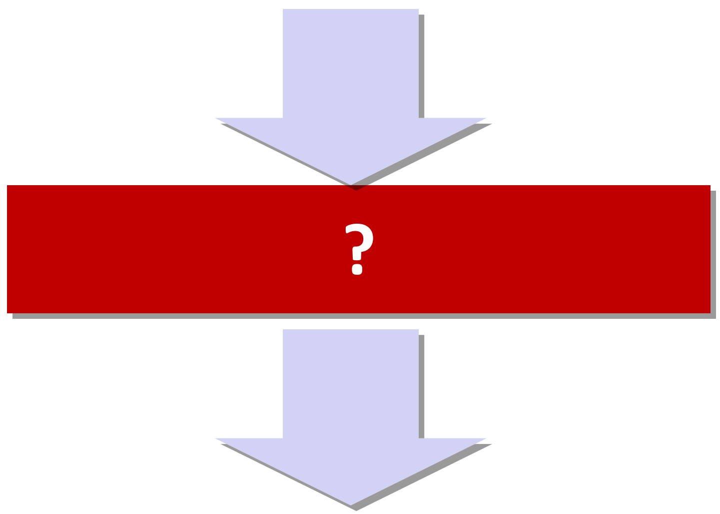
DFT Plot: Analysis



- High performance library development = nightmare
- *Automation?*

Idea: Textbook to Adaptive Library

Textbook FFT

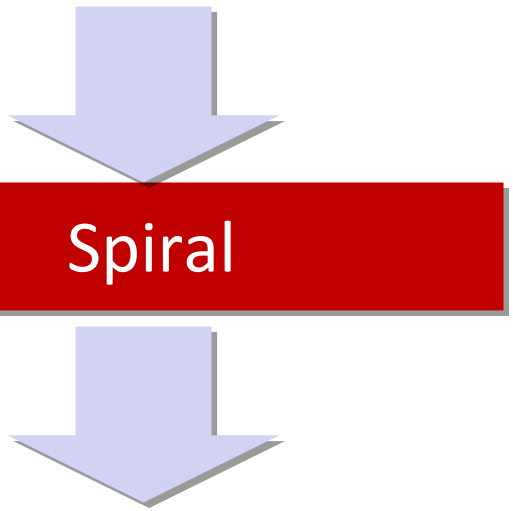


“FFTW”

Goal: Teach Computers to Write Libraries

Input:

- **Transform:** $\mathbf{DFT}_n$
- **Algorithm:** $\mathbf{DFT}_{km} \rightarrow (\mathbf{DFT}_k \otimes I_m) T^{km}_m (I_k \otimes \mathbf{DFT}_m) L^{km}_k$
- **Hardware:** 2-way SIMD + multithreaded



Output:

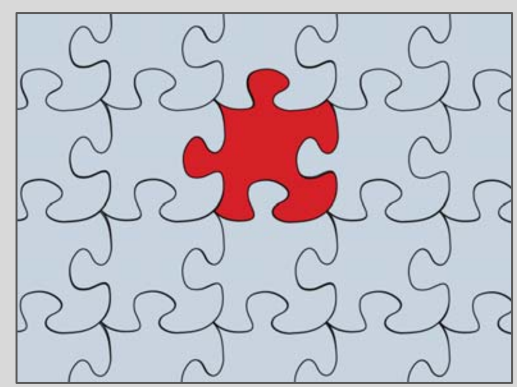
- FFTW equivalent library
- For general input size
- Vectorized and multithreaded
- **Performance competitive**

Key technologies:

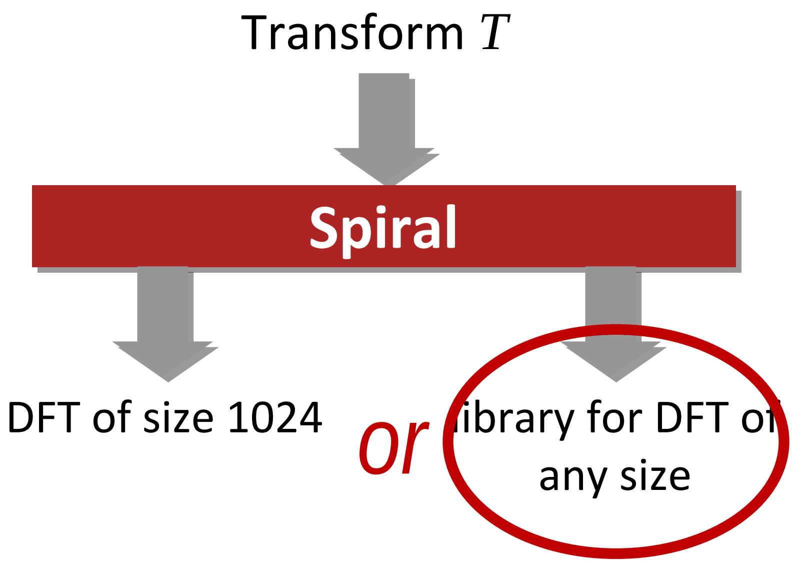
- **Layered domain specific language**
- **Algorithm manipulation via rewriting**
- **Feedback-driven search**

Result:

- **Full automation**



Contribution: General Size Library

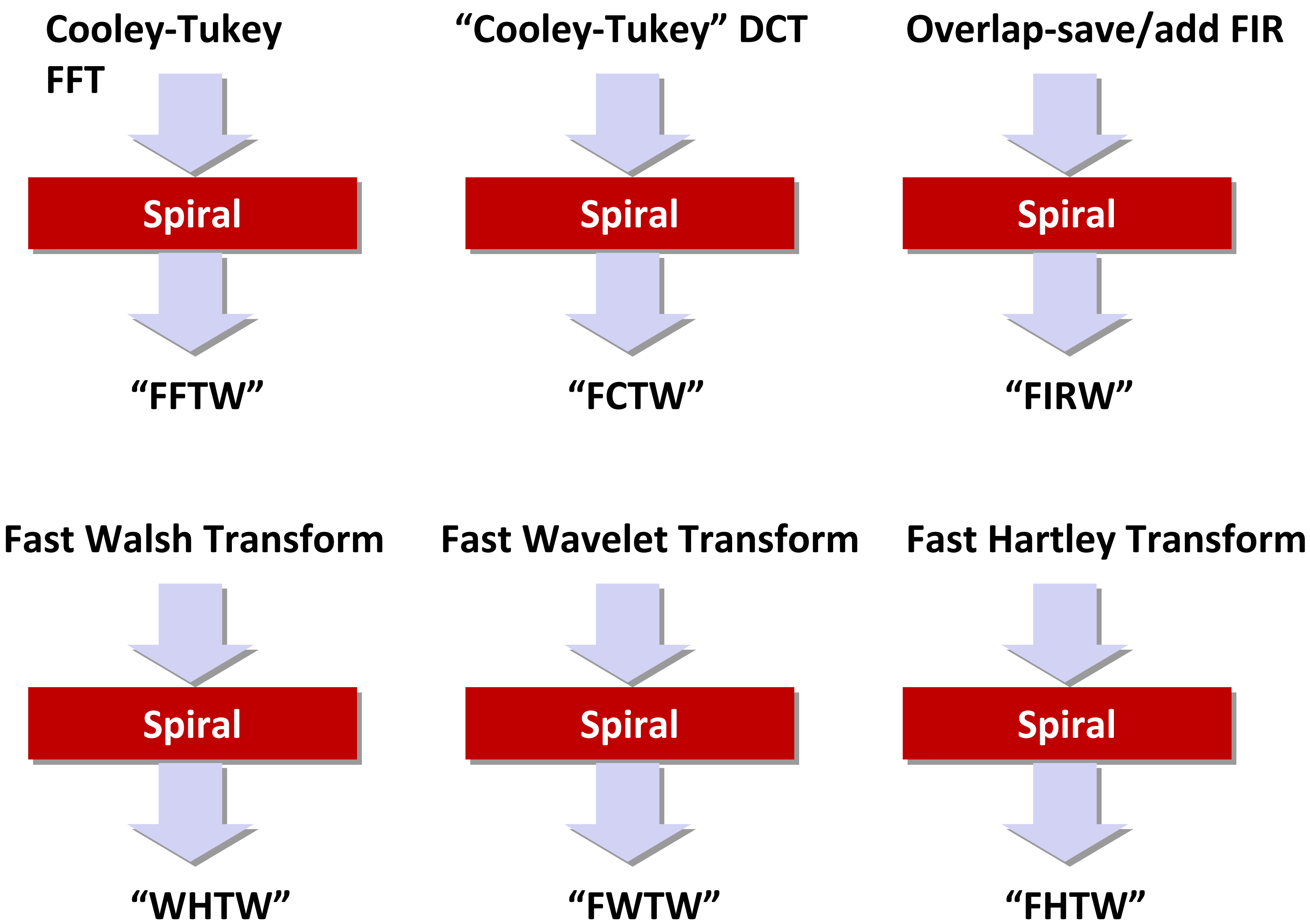



```
dft_1024(X, Y);
```

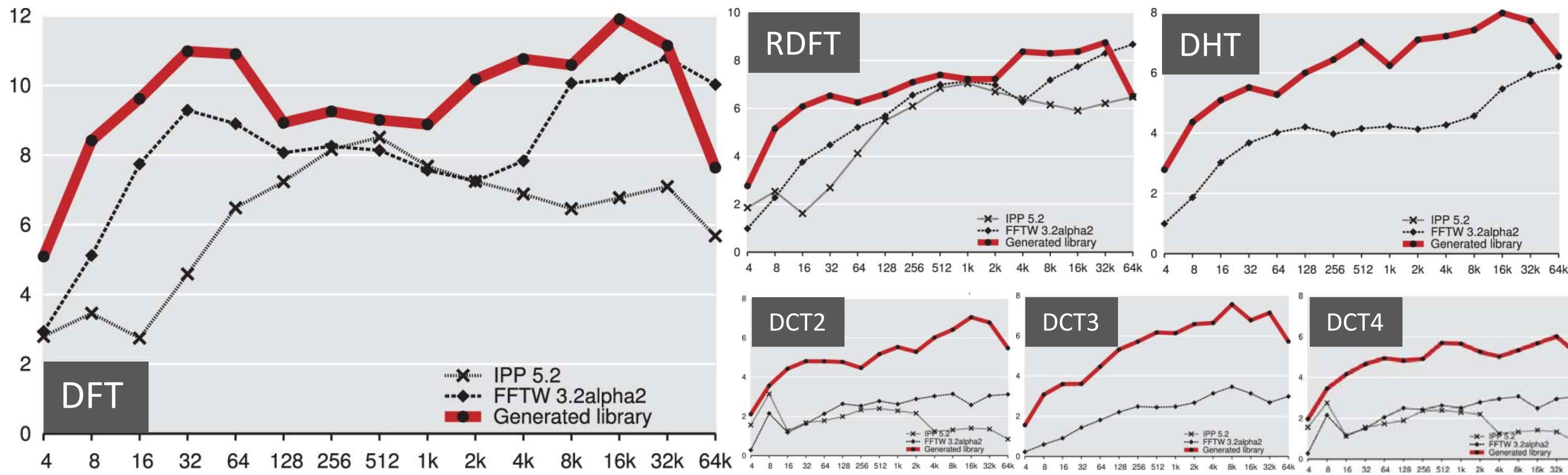
```
Env_1 dft(1024);
dft.compute(X, Y);
```

Fundamentally different problems

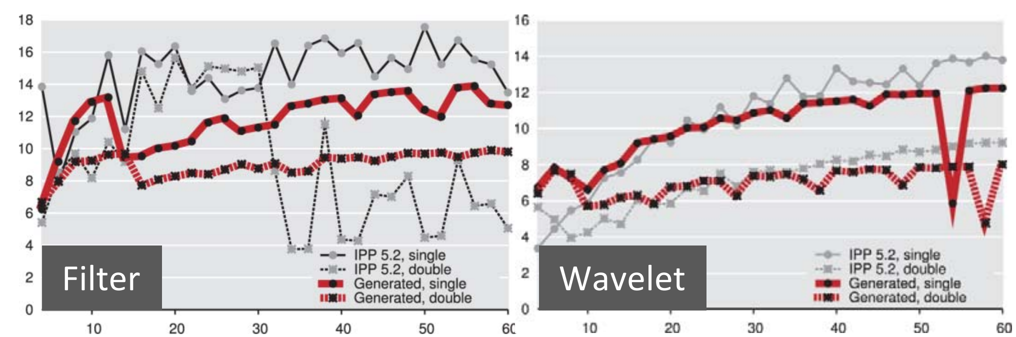
Beyond Fourier Transform and FFTW



Examples of Generated Libraries



- 2-way vectorized, 2-threaded
- Most are faster than hand-written libs
- Code size: 8–120 KLOC or 0.5–5 MB
- Generation time: 1–3 hours



Total: 300 KLOC / 13.3 MB of code generated in < 20 hours
from a few simple algorithm specs

Intel IPP library 6.0 will include Spiral generated code

Linear Transforms

- **Mathematically:** matrix-vector product

$$y = Tx$$

Output vector **Transform matrix** **Input vector**

- **Examples:**

$$\mathbf{DFT}_n = \left[\omega_n^{k\ell}\right]_{0\le k,\ell<n}, \quad \omega_n = e^{-2\pi j/n}$$

$$\mathbf{WHT}_n = \begin{bmatrix} \mathbf{WHT}_{n/2} & \mathbf{WHT}_{n/2} \\ \mathbf{WHT}_{n/2} & -\mathbf{WHT}_{n/2} \end{bmatrix}$$

$$\mathbf{RDFT}_n = \left[\begin{matrix} I''_n \\ \begin{bmatrix} \cos\frac{2\pi k\ell}{n} \\ -\sin\frac{2\pi k\ell}{n} \end{bmatrix} \end{matrix} \right]$$

$$\mathbf{DHT}_n = \left[\begin{matrix} I''_n \\ \begin{bmatrix} \mathrm{cas}\,\frac{2\pi k\ell}{n} \\ \mathrm{cms}\,\frac{2\pi k\ell}{n} \end{bmatrix} \end{matrix} \right]$$

$$\mathbf{DCT\text{-}1}_n = \left[\cos\frac{k\ell\pi}{n-1}\right]$$

$$\mathbf{DST\text{-}1}_n = \left[\sin\frac{(k+1)(\ell+1)\pi}{n+1}\right]$$

$$\mathbf{DCT\text{-}2}_n = \left[\cos\frac{k(2\ell+1)\pi}{2n}\right]$$

$$\mathbf{DST\text{-}2}_n = \left[\sin\frac{(k+1)(2\ell+1)\pi}{2n}\right]$$

$$\mathbf{DCT\text{-}3}_n = \left[\cos\frac{(2k+1)\ell\pi}{2n}\right]$$

$$\mathbf{DST\text{-}3}_n = \left[\sin\frac{(2k+1)(\ell+1)\pi}{2n}\right]$$

$$\mathbf{DCT\text{-}4}_n = \left[\cos\frac{(2k+1)(2\ell+1)\pi}{4n}\right]$$

$$\mathbf{DST\text{-}4}_n = \left[\sin\frac{(2k+1)(2\ell+1)\pi}{4n}\right]$$

$$\mathbf{MDCT}_n = \left[\cos\frac{(2k+1)(2\ell+1+n)\pi}{4n}\right]$$

$$\mathbf{IMDCT}_n = \left[\cos\frac{(2\ell+1)(2k+1+n)\pi}{4n}\right]$$

Fast Algorithms, Example: 4-point FFT

- **Fast algorithms = matrix factorizations**

$$\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & i & -1 & -i\\ 1 & -1 & 1 & -1\\ 1 & -i & -1 & i\end{bmatrix} x \rightarrow \begin{bmatrix}1 & \cdot & 1 & \cdot\\ \cdot & 1 & \cdot & 1\\ 1 & \cdot & -1 & \cdot\\ \cdot & 1 & \cdot & -1\end{bmatrix}\begin{bmatrix}1 & \cdot & \cdot & \cdot\\ \cdot & 1 & \cdot & \cdot\\ \cdot & \cdot & 1 & \cdot\\ \cdot & \cdot & \cdot & i\end{bmatrix}\begin{bmatrix}1 & 1 & \cdot & \cdot\\ 1 & -1 & \cdot & \cdot\\ \cdot & \cdot & 1 & 1\\ \cdot & \cdot & 1 & -1\end{bmatrix}\begin{bmatrix}1 & \cdot & \cdot & \cdot\\ \cdot & \cdot & 1 & \cdot\\ \cdot & 1 & \cdot & \cdot\\ \cdot & \cdot & \cdot & 1\end{bmatrix} x$$

12 adds
4 mults

4 adds

1 mult

4 adds

(when multiplied with input vector {)

$$\mathrm{DFT}_4 \rightarrow (\mathrm{DFT}_2 \otimes \mathrm{I}_2) D (\mathrm{I}_2 \otimes \mathrm{DFT}_2) P$$

Fourier transform Kronecker product Identity Permutation

- **SPL = mathematical, declarative specification**

- **Space of algorithms generated using breakdown rules**

$$\mathrm{DFT}_{mn} \rightarrow (\mathrm{DFT}_m \otimes \mathrm{I}_n) D (\mathrm{I}_m \otimes \mathrm{DFT}_n) P$$

Examples of Breakdown Rules

$$\mathbf{DFT}_n \longrightarrow (\mathbf{DFT}_k \otimes I_m) D_{k,m} (I_k \otimes \mathbf{DFT}_m) L^n_k$$

DFT Cooley-Tukey

$$\mathbf{DFT}_n \longrightarrow V^{-1}_{m,k} (\mathbf{DFT}_k \otimes I_m)(I_k \otimes \mathbf{DFT}_m) V_{m,k} \qquad (2.2)$$

$$\mathbf{DFT}_n \longrightarrow W^{-1}_n (I_1 \oplus \mathbf{DFT}_{n-1}) E_n (I_1 \oplus \mathbf{DFT}_{n-1}) W_n \qquad (2.3)$$

$$\mathbf{DFT}_n \longrightarrow B^\top_{n,m} D_m \, \mathbf{DFT}_m \, D'_m \, \mathbf{DFT}_m \, D''_m B_{n,m}, \quad m \geq 2n-1 \qquad (2.4)$$

$$\mathbf{DFT}_n \longrightarrow P^\top_{k/2,2m} \left(\mathbf{DFT}_{2m} \oplus \left(I_{k/2-1} \otimes_i C_{2m} \, \mathbf{rDFT}_{2m}((i+1)/k)\right)\right) \left(\mathbf{RDFT}'_k \otimes I_m\right) \qquad (2.5)$$

$$\begin{vmatrix} \mathbf{RDFT}_n \\ \mathbf{RDFT}'_n \\ \mathbf{DHT}_n \\ \mathbf{DHT}'_n \end{vmatrix} \longrightarrow (P^\top_{k/2,m} \otimes I_2) \left(\begin{vmatrix} \mathbf{RDFT}_{2m} \\ \mathbf{RDFT}'_{2m} \\ \mathbf{DHT}_{2m} \\ \mathbf{DHT}'_{2m} \end{vmatrix} \oplus \left(I_{k/2-1} \otimes_i M_{2m} \begin{vmatrix} \mathbf{rDFT}_{2m}((i+1)/k) \\ \mathbf{rDFT}_{2m}((i+1)/k) \\ \mathbf{rDHT}_{2m}((i+1)/k) \\ \mathbf{rDHT}_{2m}((i+1)/k) \end{vmatrix} \right) \right)$$

$$\cdot \left(\begin{vmatrix} \mathbf{RDFT}'_k \\ \mathbf{RDFT}'_k \\ \mathbf{DHT}'_k \\ \mathbf{DHT}'_k \end{vmatrix} \otimes I_m \right) \qquad (2.6)$$

$$\mathbf{RDFT}_n \longrightarrow D_n \cdot \mathbf{DCT\text{-}2}_n \cdot P_n, \quad n \text{ odd} \qquad (2.7)$$

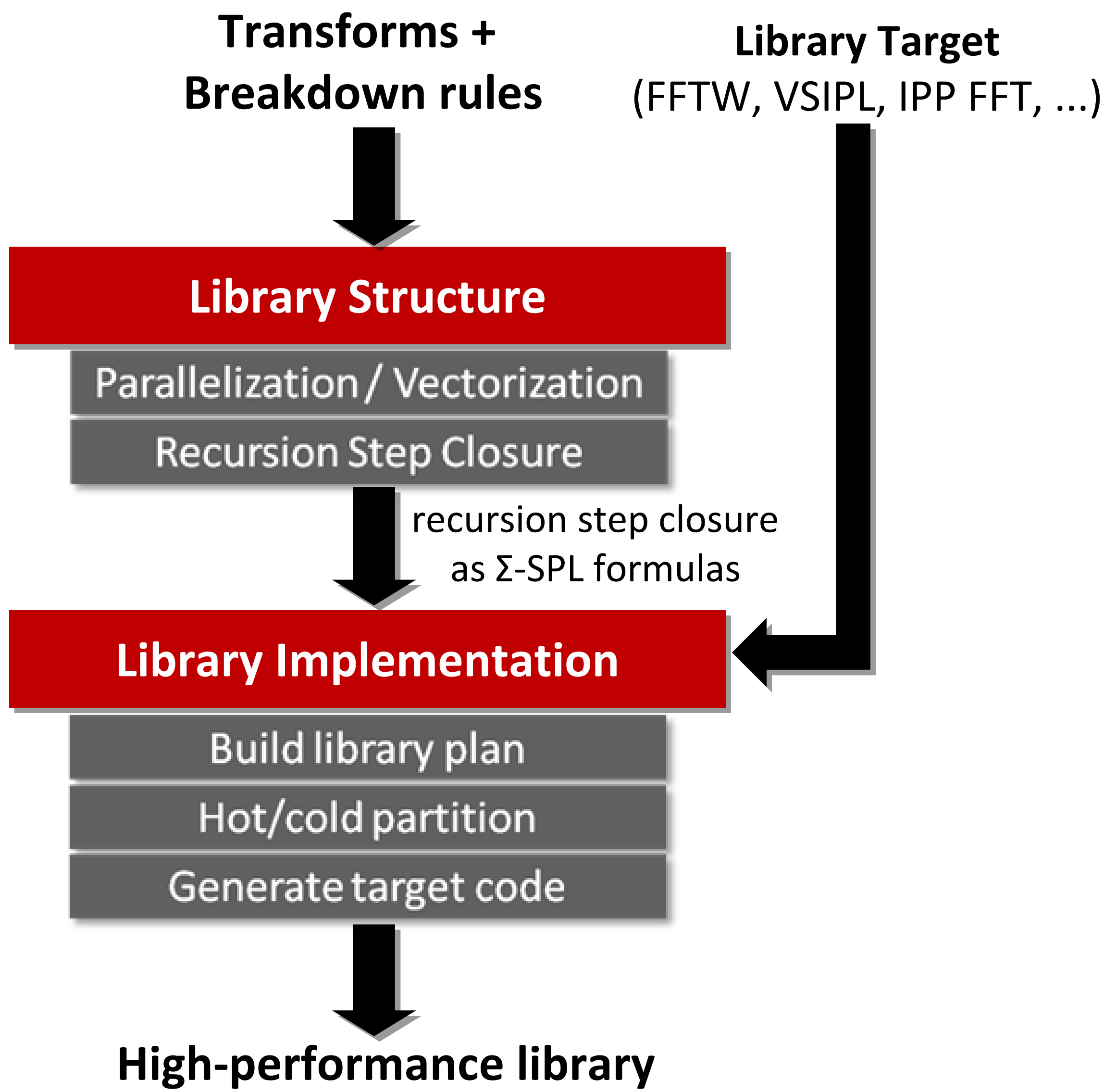
$$\mathbf{DCT\text{-}2}_n \longrightarrow P^\top_{k/2,2m} \left(\mathbf{DCT\text{-}2}_{2m} K^{2m}_2 \oplus \left(I_{k/2-1} \otimes N_{2m} \, \mathbf{RDFT\text{-}3}^\top_{2m}\right)\right) G_n (L^{n/2}_{k/2} \otimes I_2)$$

$$\cdot (I_m \otimes \mathbf{RDFT}'_k) Q_{m/2,k} \qquad (2.8)$$

DCT “Cooley-Tukey”

- “Teach” Spiral domain knowledge of algorithms. Never obsolete.
- Each rule leads to a library

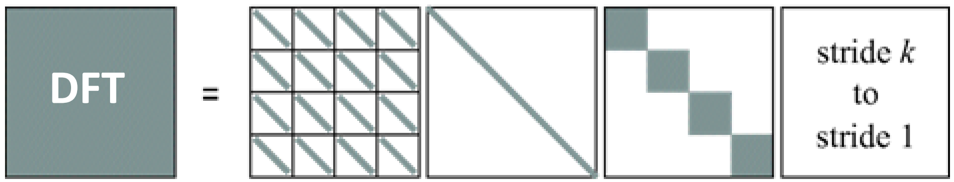
How Library Generation Works



Breakdown Rules to Library Code

- **Cooley-Tukey Fast Fourier Transform (FFT)**

$$\mathrm{DFT}_{km}\, x = (\mathrm{DFT}_k \otimes I_m)\, T_m^{km}\, (I_k \otimes \mathrm{DFT}_m)\, L_k^{km}\, x$$



k=4

- **Naive implementation**

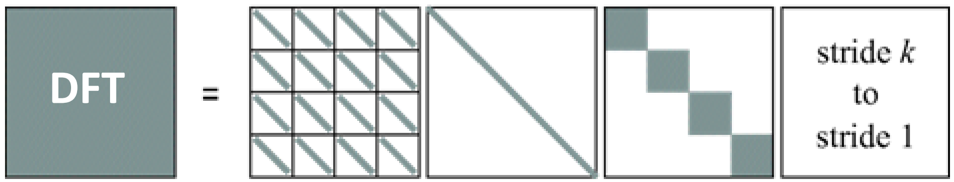
```
void dft(int n, cplx X[], cplx Y[]) {
  k = choose_factor(n); m = n/k;
  Z = permute(X)

  for i=0 to k-1
    dft_subvec(m, Z, Y, …)
  for i=0 to n-1
    Y[i] = Y[i]*T[i];
  for i=0 to m-1
    dft_strided(k, Y, Y, …)
}
```

2 extra functions needed

Breakdown Rules to Library Code

- **Cooley-Tukey Fast Fourier Transform (FFT)**

$$\mathrm{DFT}_{km}\, x = (\mathrm{DFT}_k \otimes I_m) T_m^{km} (I_k \otimes \mathrm{DFT}_m) L_k^{km} x$$



- **Naive implementation**

```
void dft(int n, cplx X[], cplx Y[]) {
  k = choose_factor(n); m = n/k;
  Z = permute(X)

  for i=0 to k-1
    dft_subvec(m, Z, Y, …)
  for i=0 to n-1
    Y[i] = Y[i]*T[i];
  for i=0 to m-1
    dft_strided(k, Y, Y, …)
}
```

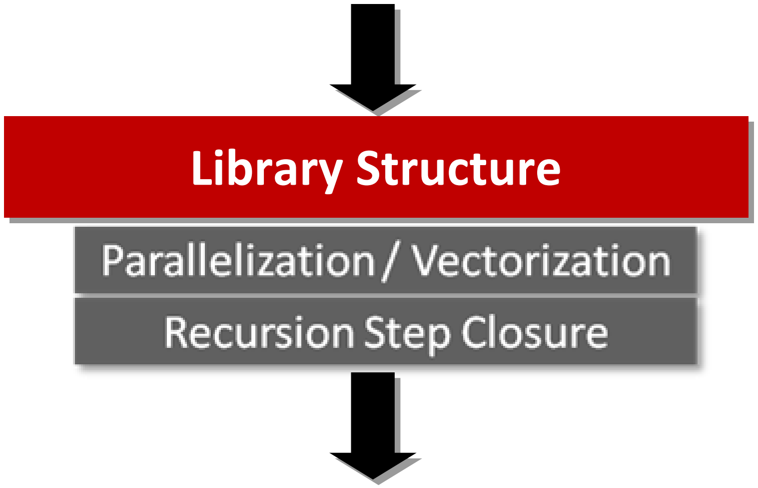
- **Optimized implementation**

```
void dft(int n, cplx X[], cplx Y[]) {
  k = choose_factor(n); m = n/k;

  for i=0 to k-1
    dft_strided2(m, X, Y, …)

  for i=0 to m-1
    dft_strided3_scaled(k, Y, Y, T, …)
}
```

How to discover these specialized variants automatically?

Library Structure

$$V_{k,m}^T(\mathbf{DFT}_k \otimes I_m)(I_k \otimes \mathbf{DFT}_m)\, V_{k,m}$$
$$(\mathbf{DFT}_k \otimes I_m)\, T_m^{km}(I_k \otimes \mathbf{DFT}_m)\, L_k^{km}$$



$$\mathbf{DFT}$$
$$S_z\, \mathbf{DFT}\, G_h$$
$$S_h\, \mathbf{DFT}\, G_z$$
$$S_h\, \mathbf{DFT}\, \mathrm{diag}\,(\mathrm{Dat})\, G_h$$
$$S_h\, \mathbf{DFT}\, G_h$$
$$S_h\, \mathbf{DFT}\, G_{h \circ z}$$
$$S_{h \circ z}\, \mathbf{DFT}\, G_h$$
$$S_h\, \mathbf{DFT}\, \mathrm{diag}\,(\mathrm{Dat})\, G_{h \circ z}$$
$$S_z\, \mathbf{DFT}\, \mathrm{diag}\,(\mathrm{Dat})\, G_h$$
$$S_{h \circ z}\, \mathbf{DFT}\, \mathrm{diag}\,(\mathrm{Dat})\, G_h$$

- **Input:**
  - Breakdown rules
- **Output:**
  - Recursion step closure
  - Σ-SPL Implementation of each recursion step
- **Parallelization/Vectorization**
  - Adds additional breakdown rules
  - Orthogonal to the closure generation

Computing Recursion Step Closure

- **Input:** transform T and a breakdown rule
- **Output:** spawned recursion steps + Σ-SPL implementation
- **Algorithm:**
  1. Apply the breakdown rule
  2. Convert to Σ-SPL
  3. Apply loop merging + index simplification rules.
  4. Extract recursion steps
  5. Repeat until closure is reached

$$\{\mathbf{DFT}_n\}$$

$$\downarrow$$

$$(\{\mathbf{DFT}_{n/k}\} \otimes I_k) T_k^n (I_{n/k} \otimes \{\mathbf{DFT}_k\}) L_{n/k}^n$$

$$\downarrow$$

$$\left(\sum_{i=0}^{k-1} \mathsf{S}_{h_{i,k}} \{\mathbf{DFT}_{n/k}\}\, \mathsf{G}_{h_{i,k}}\right) \mathsf{diag}(f) \left(\sum_{j=0}^{n/k-1} \mathsf{S}_{h_{jk,1}} \{\mathbf{DFT}_k\}\, \mathsf{G}_{h_{jk,1}}\right) \mathsf{perm}(\ell_{n/k}^n)$$

$$\downarrow$$

$$\sum_{i=0}^{k-1} \mathsf{S}_{h_{i,k}} \{\mathbf{DFT}_{n/k}\}\, \mathsf{diag}(f \circ h_{i,k})\, \mathsf{G}_{h_{i,k}} \sum_{j=0}^{n/k-1} \mathsf{S}_{h_{jk,1}} \{\mathbf{DFT}_k\}\, \mathsf{G}_{h_{j,n/k}}$$

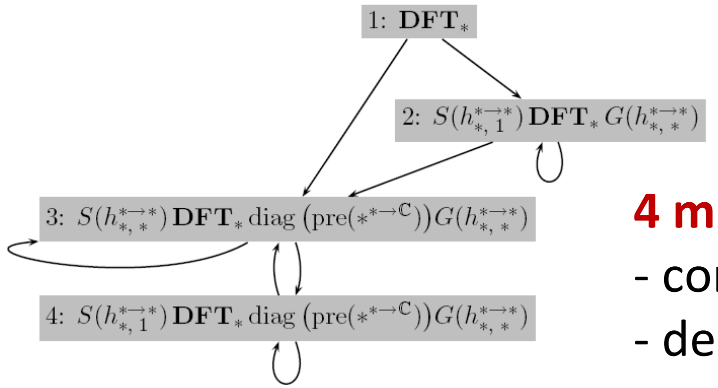
$$\downarrow$$

$$\sum_{i=0}^{k-1} \boxed{\left\{ \mathsf{S}_{h_{i,k}}\, \mathbf{DFT}_{n/k}\, \mathsf{diag}(f \circ h_{i,k})\, \mathsf{G}_{h_{i,k}} \right\}} \sum_{j=0}^{n/k-1} \boxed{\left\{ \mathsf{S}_{h_{jk,1}}\, \mathbf{DFT}_k\, \mathsf{G}_{h_{j,n/k}} \right\}}$$

Parametrization (not shown) derives the independent parameter set for each recursion step

Recursion Step Closure Examples

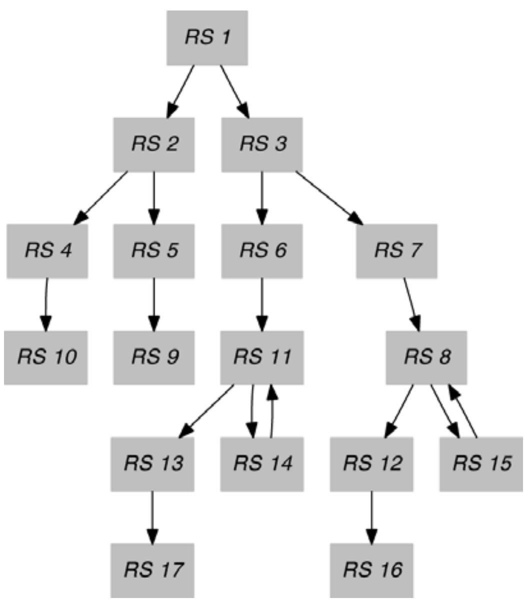
DFT (scalar)



4 mutually recursive functions

- computed automatically
- described using Σ-SPL formulas

DCT4 (vectorized)



1 : $\mathrm{Vec}_2(\mathbf{DCT\text{-}4}_{u_1})$

2 : $\mathrm{Vec}_2(\mathrm{GT}(\mathrm{diag}\,(N_{2u_8})\,\mathbf{RDFT\text{-}3}^{\top}_{2u_8}\,\mathrm{rcdiag}(\mathrm{pre}({u_4}^{\mathbb{Z}\times 2u_8\to\mathbb{R}})),\ h^{2u_8\to u_6}_{0,\,1,u_7}\circ \ell^{2u_8}_{u_8},\ r^{2u_8\to u_9}_{0,\,u_{11},1,\,u_{12}},\ \{u_{13}\}))$

3 : $\mathrm{Vec}_2(\mathrm{GT}(\mathbf{RDFT\text{-}3}_{u_1}\,\mathrm{diag}\,(N_{u_1}),\ r^{u_1\to u_3}_{0,\,u_5,1,\,u_6},\ h^{u_1\to u_8}_{0,\,u_9,1},\ \{u_{10}\}))$

4 : $\mathrm{VJam}_2(\mathrm{GT}(\mathrm{diag}\,(N_{2u_9})\,\mathbf{RDFT\text{-}3}^{\top}_{2u_9}\,\mathrm{rcdiag}(\mathrm{pre}({u_4}^{\mathbb{Z}\times\mathbb{Z}\times 2u_9\to\mathbb{R}})),\ h^{2u_9\to u_6}_{0,\,1,u_7,u_8}\circ \ell^{2u_9}_{u_9},\ r^{2u_9\to u_{10}}_{0,\,u_{12},1,2,\,u_{13}},\ \{2,u_{14}\}))$

5 : $\mathrm{GT}(\mathrm{diag}\,(N_{2u_9})\,\mathbf{RDFT\text{-}3}^{\top}_{2u_9}\,\mathrm{rcdiag}(\mathrm{pre}({u_4}^{\mathbb{Z}\times 2u_9\to\mathbb{R}})),\ h^{2u_9\to u_6}_{u_7,\,1,u_8}\circ \ell^{2u_9}_{u_9},\ r^{2u_9\to u_{10}}_{u_{12},\,u_{13},1,\,u_{14}},\ \{u_{15}\})$

6 : $\mathrm{VJam}_2(\mathrm{GT}(\mathbf{RDFT\text{-}3}_{u_1}\,\mathrm{diag}\,(N_{u_1}),\ r^{u_1\to u_3}_{0,\,u_5,1,2,\,u_6},\ h^{u_1\to u_8}_{0,\,u_9,1,2},\ \{2,u_{10}\}))$

7 : $\mathrm{GT}(\mathbf{RDFT\text{-}3}_{u_1}\,\mathrm{diag}\,(N_{u_1}),\ r^{u_1\to u_3}_{u_5,\,u_6,1,\,u_7},\ h^{u_1\to u_9}_{u_{10},\,u_{11},1},\ \{u_{12}\})$

8 : $\mathrm{S}(h^{u_1\to u_2}_{u_3,\,u_4})\,\mathbf{RDFT\text{-}3}_{u_1}\,\mathrm{diag}\,(N_{u_1})\mathrm{G}(r^{u_1\to u_7}_{u_9,\,u_{10},\,u_{11}})$

9 : $\mathrm{S}(r^{2u_{13}\to u_1}_{u_3,\,u_4,\,u_5})\mathrm{diag}\,(N_{2u_{13}})\,\mathbf{RDFT\text{-}3}^{\top}_{2u_{13}}\,\mathrm{rcdiag}(\mathrm{pre}({u_9}^{2u_{13}\to\mathbb{R}}))\mathrm{G}(h^{2u_{13}\to u_{11}}_{u_{12},\,1}\circ \ell^{2u_{13}}_{u_{13}})$

10 : $\mathrm{VJam}_2(\mathrm{GT}(\mathrm{diag}\,(N_{2u_9})\,\mathbf{RDFT\text{-}3}^{\top}_{2u_9}\,\mathrm{rcdiag}(\mathrm{pre}({u_4}^{\mathbb{Z}\times 2u_9\to\mathbb{R}})),\ h^{2u_9\to u_6}_{u_7,\,1,u_8}\circ \ell^{2u_9}_{u_9},\ r^{2u_9\to u_{10}}_{u_{12},\,u_{13},1,\,u_{14}},\ \{2\}))$

11 : $\mathrm{VJam}_2(\mathrm{GT}(\mathbf{RDFT\text{-}3}_{u_1}\,\mathrm{diag}\,(N_{u_1}),\ r^{u_1\to u_3}_{u_5,\,u_6,1,\,u_7},\ h^{u_1\to u_9}_{u_{10},\,u_{11},1},\ \{2\}))$

12 : $\mathrm{GT}(\mathrm{diag}\,(C_{u_1})\,\mathbf{rDFT}_{2u_1}(\lambda\text{-wrap}({\lambda_1}^{\mathbb{Z}\times\mathbb{Z}\to\mathbb{R}})),\ h^{2u_1\to u_4}_{0,\,1,u_5},\ h^{2u_{10}\to u_7}_{u_8,\,u_9}\circ(r^{u_1\to u_{10}}_{0,\,u_{12},1,\,u_{13}}\otimes \imath_2),\ \{u_{14}\})$

13 : $\mathrm{VJam}_2(\mathrm{GT}(\mathrm{diag}\,(C_{u_1})\,\mathbf{rDFT}_{2u_1}(\lambda\text{-wrap}({\lambda_1}^{\mathbb{Z}\times\mathbb{Z}\times\mathbb{Z}\to\mathbb{R}})),\ h^{2u_1\to u_4}_{u_5,\,u_6,1,u_7},\ h^{2u_{12}\to u_9}_{u_{10},\,u_{11},1}\circ(r^{u_1\to u_{12}}_{0,\,u_{14},0,1,\,u_{15}}\otimes \imath_2),\ \{2,u_{16}\}))$

14 : $\mathrm{VJam}_2(\mathrm{GT}(\mathbf{RDFT\text{-}3}_{u_1}\,\mathrm{diag}\,(N_{u_1}),\ r^{u_1\to u_3}_{u_5,\,u_6,1,u_7,\,u_8},\ h^{u_1\to u_{10}}_{u_{11},\,u_{12},1,u_{13}},\ \{2,u_{14}\}))$

15 : $\mathrm{GT}(\mathbf{RDFT\text{-}3}_{u_1}\,\mathrm{diag}\,(N_{u_1}),\ r^{u_1\to u_3}_{u_5,\,u_6,u_7,\,u_8},\ h^{u_1\to u_{10}}_{0,\,u_{11},1},\ \{u_{12}\})$

16 : $\mathrm{S}(h^{2u_5\to u_2}_{u_3,\,u_4}\circ(r^{u_6\to u_5}_{u_7,\,u_8,\,u_9}\otimes \imath_2))\mathrm{diag}\,(C_{u_6})\,\mathbf{rDFT}_{2u_6}(\lambda\text{-wrap}({\lambda_1}^{\mathbb{Z}\to\mathbb{R}}))\mathrm{G}(h^{2u_6\to u_{13}}_{u_{14},\,1})$

17 : $\mathrm{VJam}_2(\mathrm{GT}(\mathrm{diag}\,(C_{u_1})\,\mathbf{rDFT}_{2u_1}(\lambda\text{-wrap}({\lambda_1}^{\mathbb{Z}\times\mathbb{Z}\to\mathbb{R}})),\ h^{2u_1\to u_4}_{u_5,\,u_6,1},\ h^{2u_{11}\to u_8}_{u_9,\,u_{10},1}\circ(r^{u_1\to u_{11}}_{u_{13},\,u_{14},\,u_{15}}\otimes \imath_2),\ \{2\}))$

17 mutually recursive functions

Base Cases

- **Base cases are called "codelets" in FFTW**
- **Why needed:**
  - Closure is converted into mutually recursive functions
  - Recursion must be terminated
  - Larger base cases eliminate overhead from recursion
- **How many:**
  - **In FFTW 3.2:** 183 codelets for complex DFT (21 types)
    147 codelets for real DFT (18 types)
  - **In our generator:** # codelet types ‰ # recursion steps
- **Obtained by using standard Spiral to generate fixed size code**

$$\left\{\mathrm{S}(h_{u_3,1}^{2\rightarrow u_2})\,\mathbf{DFT}_2\,\mathrm{G}(h_{u_7,u_8}^{2\rightarrow u_6})\right\}$$
$$\left\{\mathrm{S}(h_{u_3,1}^{3\rightarrow u_2})\,\mathbf{DFT}_3\,\mathrm{G}(h_{u_7,u_8}^{3\rightarrow u_6})\right\}$$
$$\ldots$$

Library Implementation

$$\begin{array}{c}
\mathbf{DFT} \\
S_z\, \mathbf{DFT}\, G_h \\
S_h\, \mathbf{DFT}\, G_z \\
S_h\, \mathbf{DFT}\, \mathrm{diag}\,(\mathrm{Dat})\, G_h \\
S_h\, \mathbf{DFT}\, G_h \\
S_h\, \mathbf{DFT}\, G_{h\circ z} \\
S_{h\circ z}\, \mathbf{DFT}\, G_h \\
S_h\, \mathbf{DFT}\, \mathrm{diag}\,(\mathrm{Dat})\, G_{h\circ z} \\
S_z\, \mathbf{DFT}\, \mathrm{diag}\,(\mathrm{Dat})\, G_h \\
S_{h\circ z}\, \mathbf{DFT}\, \mathrm{diag}\,(\mathrm{Dat})\, G_h
\end{array}$$

Library Implementation

Build library plan

Hot/cold partition

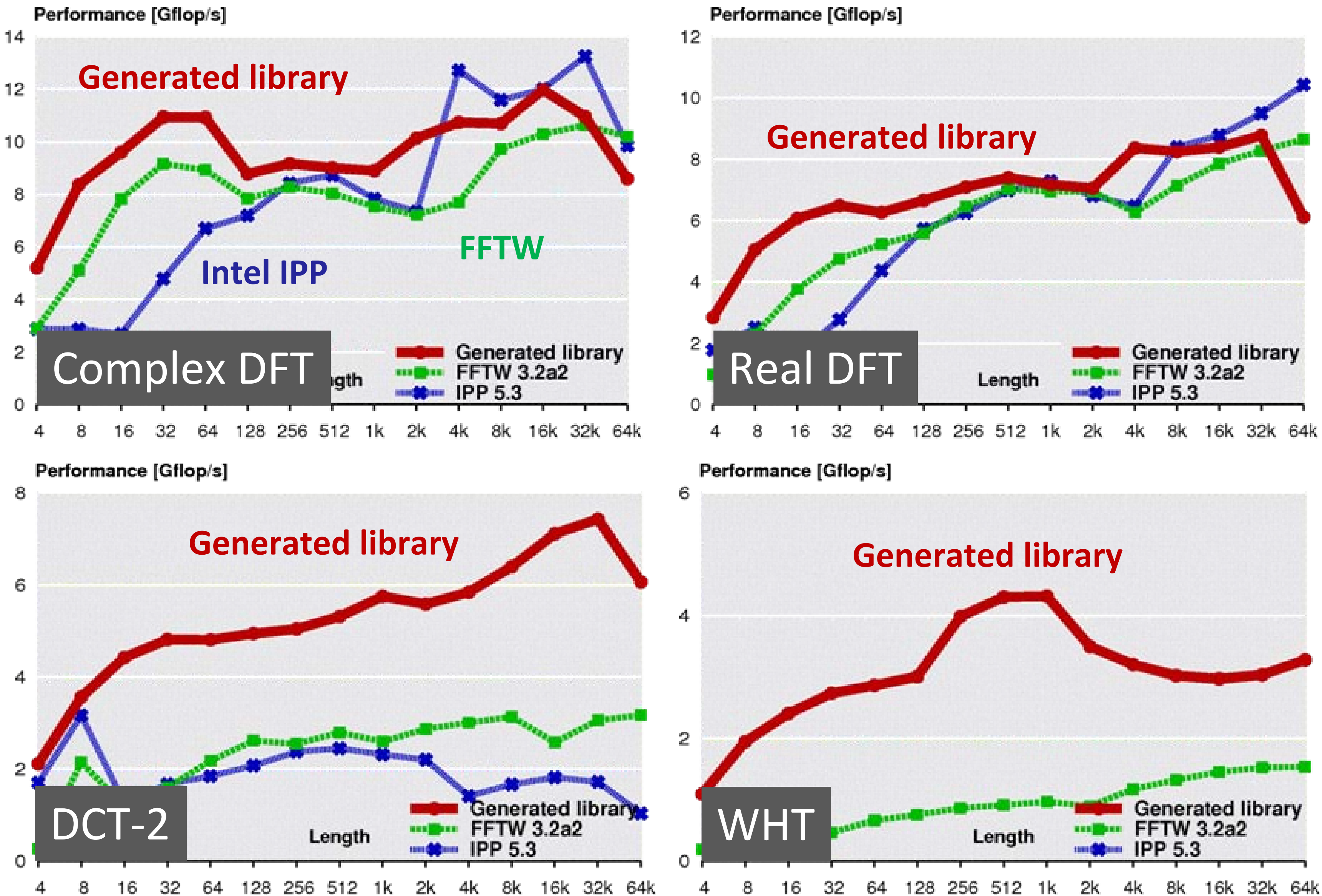
Generate target code

High-performance library

- **Input:**
  - Recursion step closure
  - Σ-SPL implementation of each recursion step
    (base cases + recursions)
- **Output:**
  - High-performance library
  - Target language: C++, Java, etc.
- **Process:**
  - Build library plan
  - Perform hot/cold partitioning
  - Generate target language code

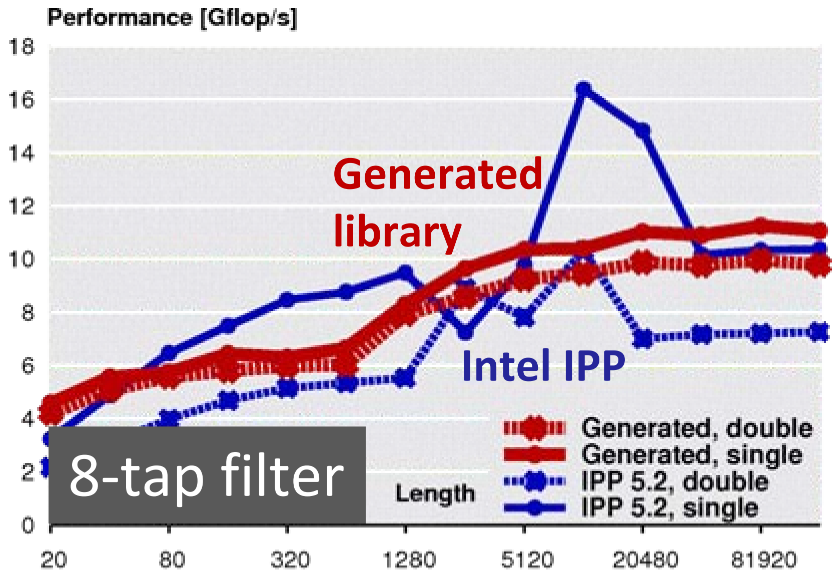
Double Precision Performance: Intel Xeon 5160

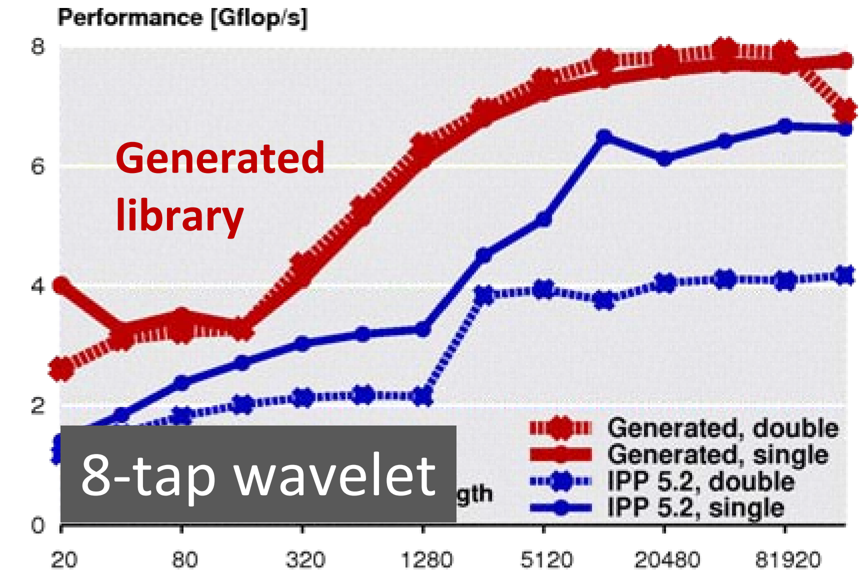
2-way vectorization, up to 2 threads

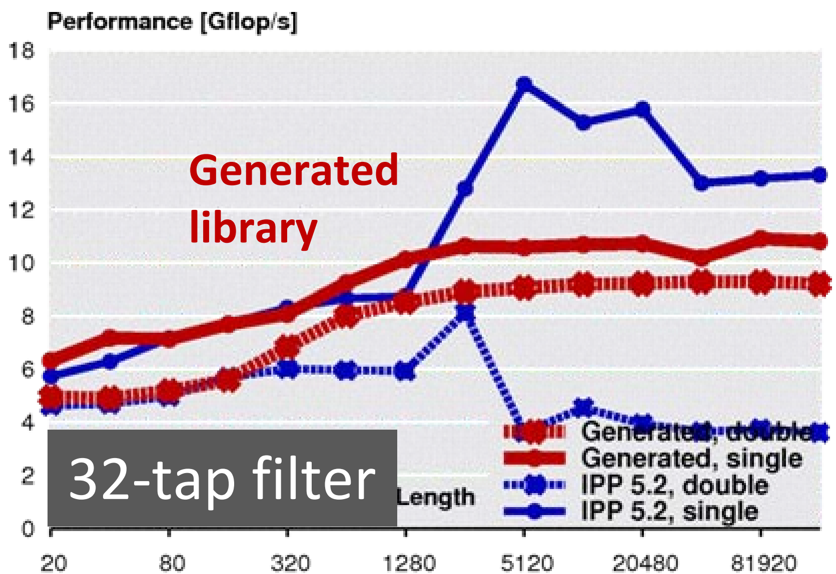


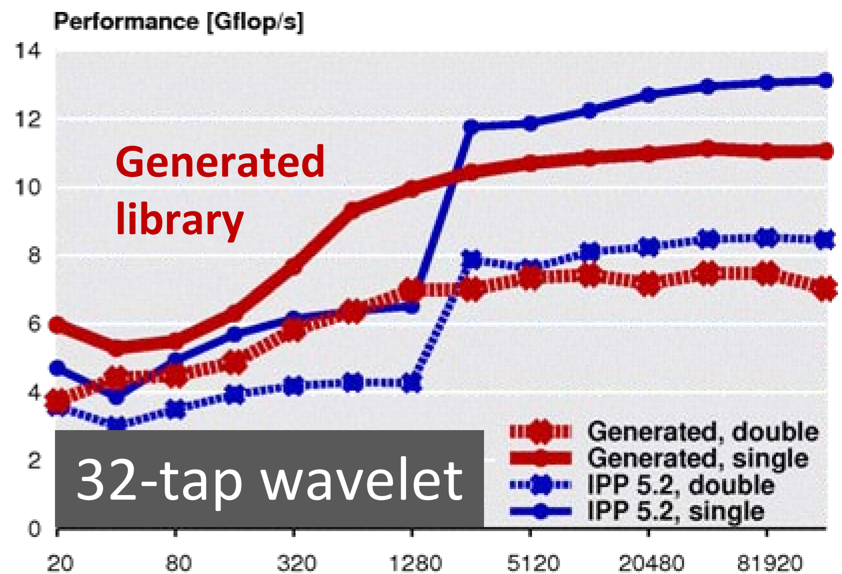
FIR Filter Performance

2- and 4-way vectorization, up to 2 threads



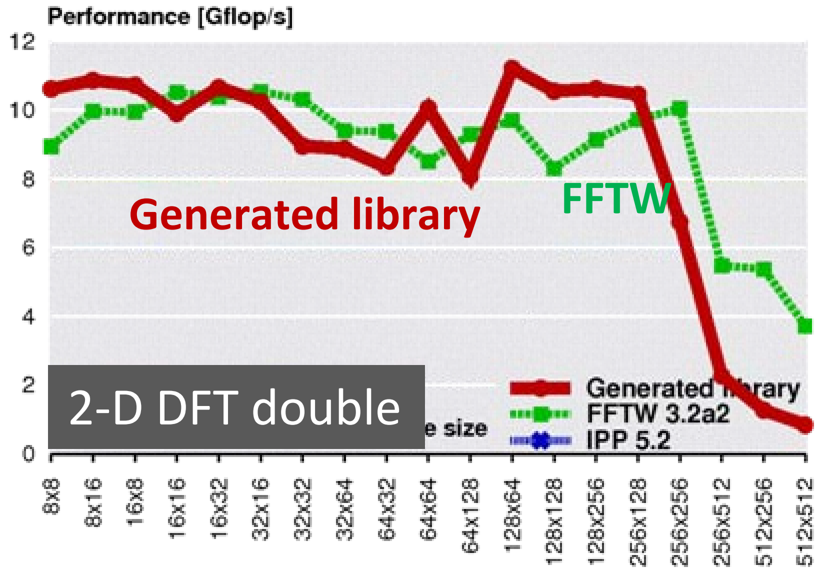


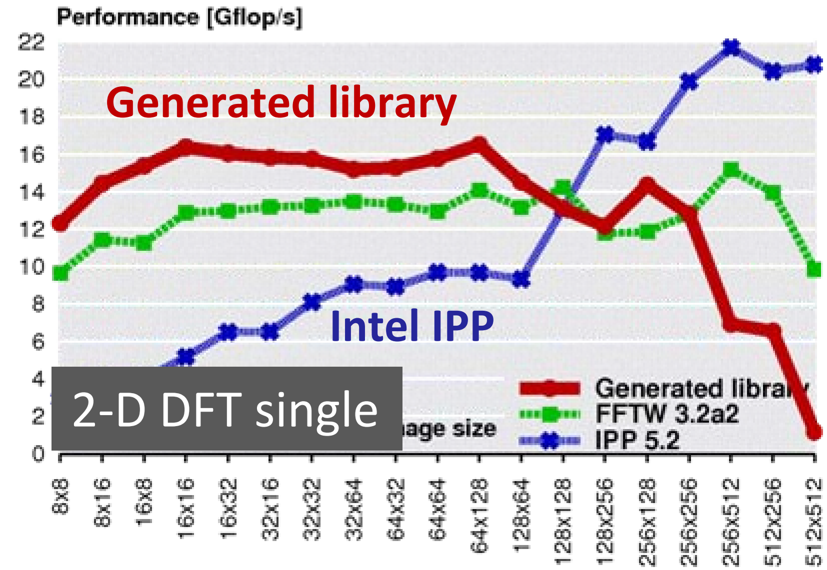


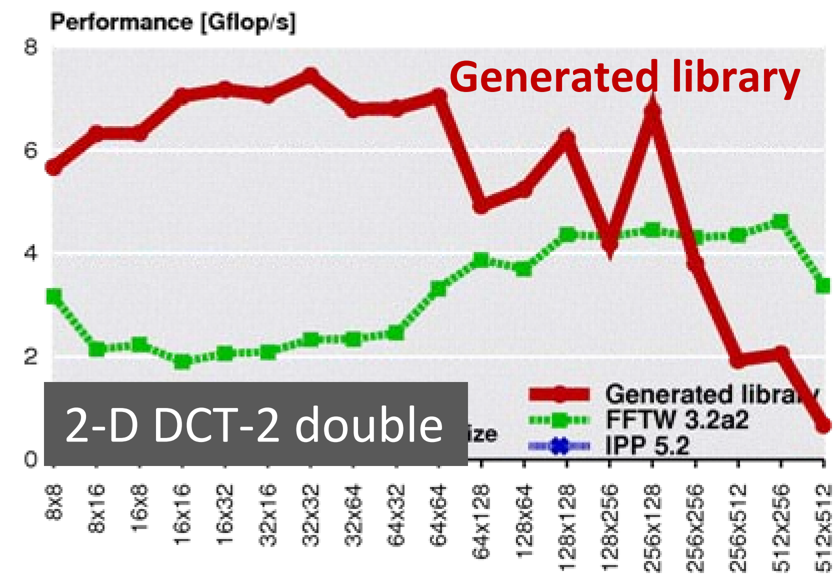


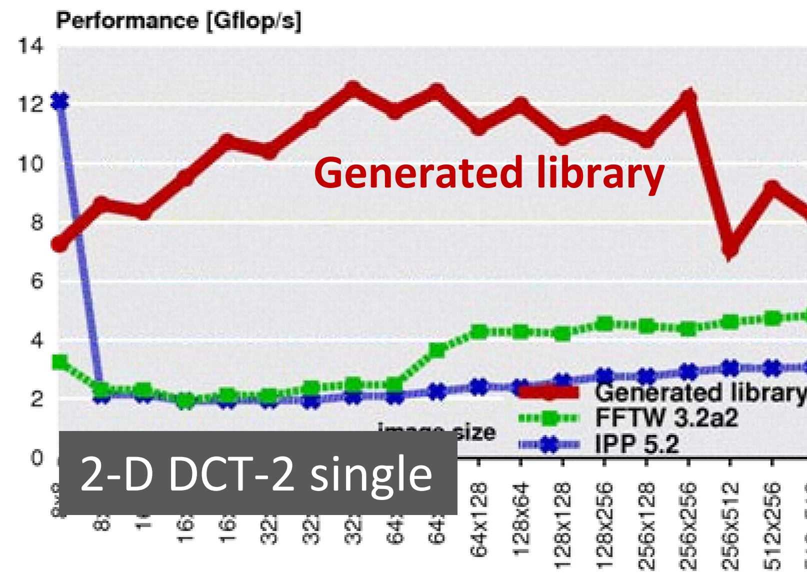
2-D Transforms Performance

2- or 4-way vectorization, up to 2 threads

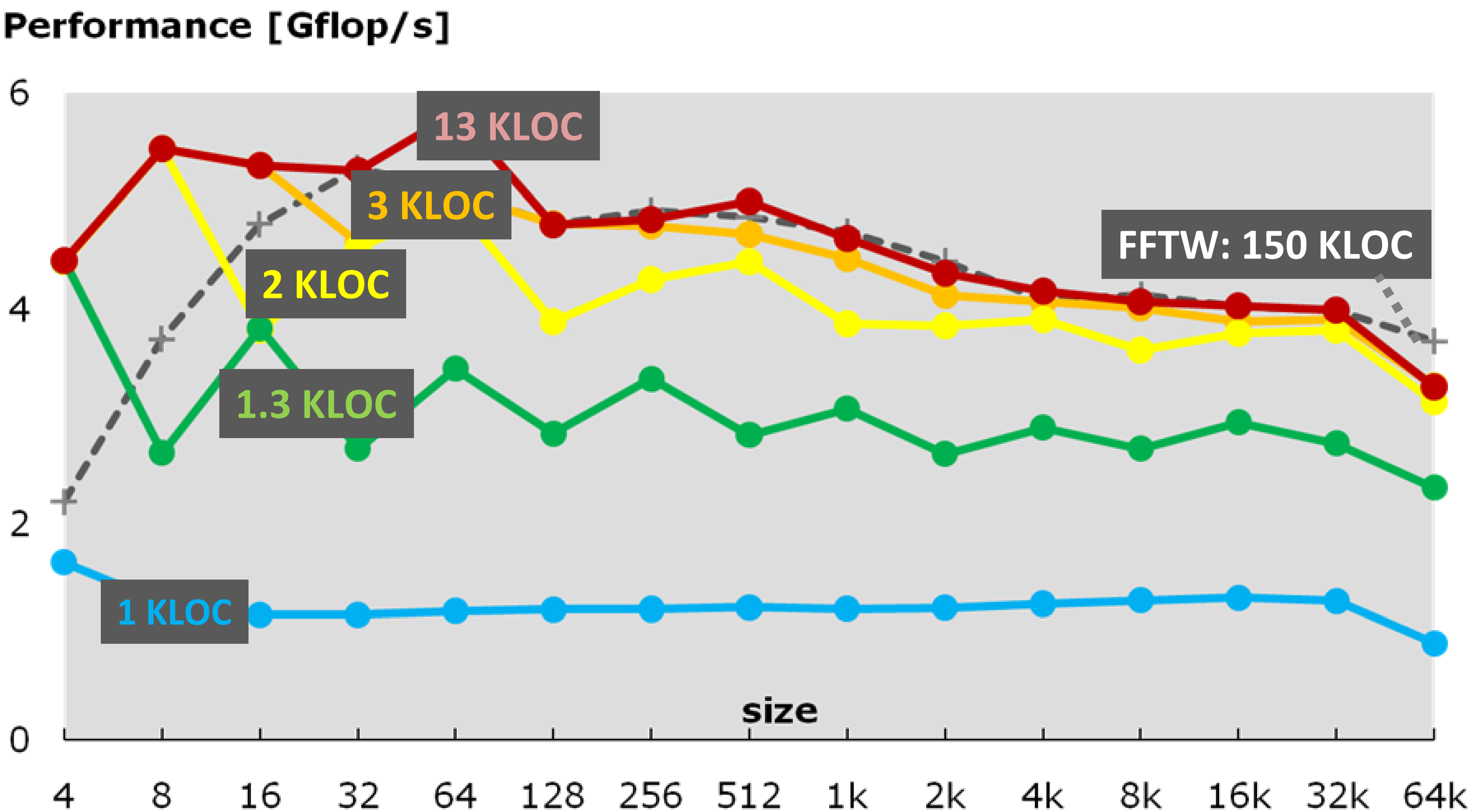




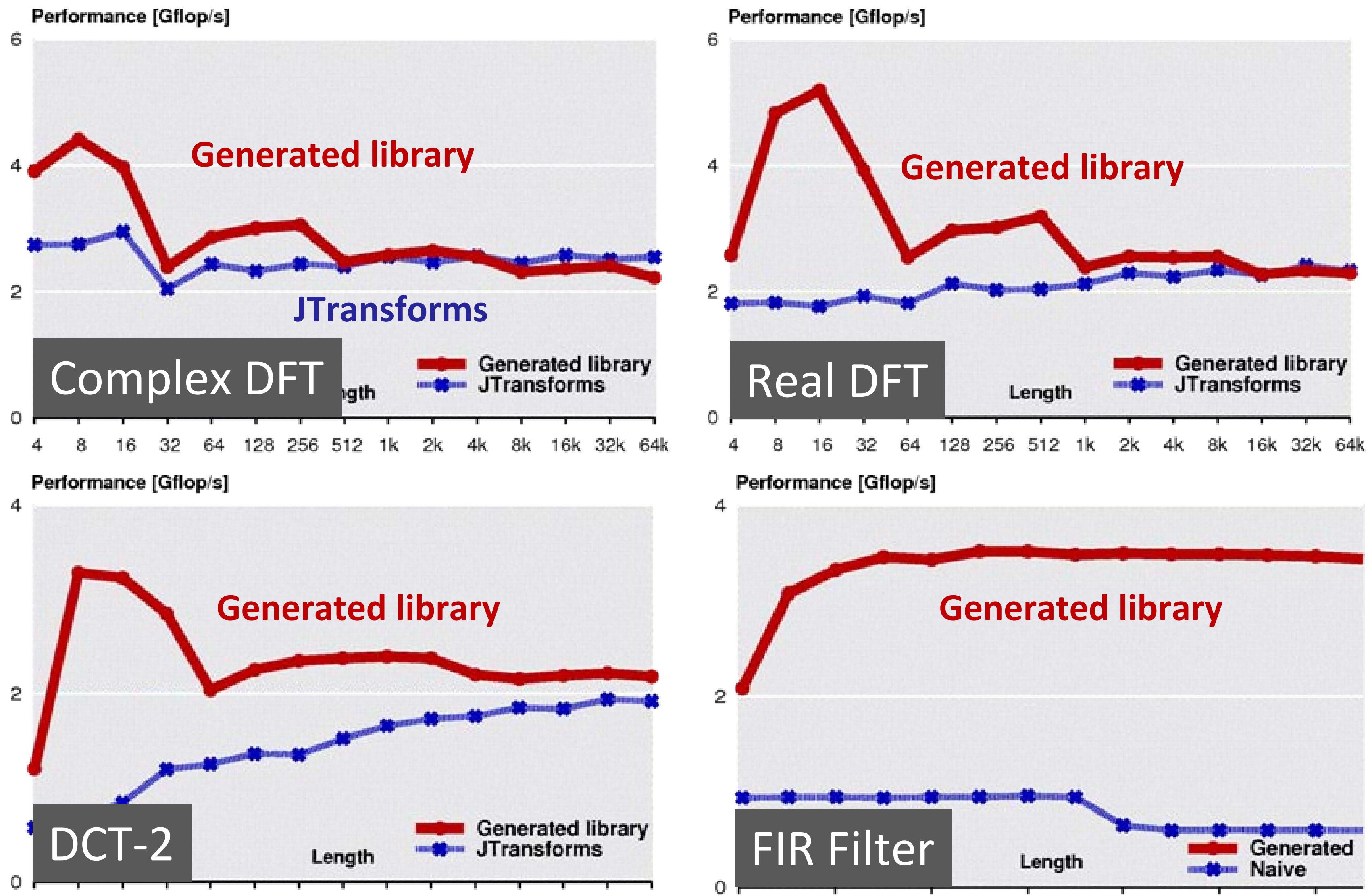




Customization: Code Size



Backend Customization: Java



Portable, but only 50% of scalar C performance

Summary

- **Full automation:**
  Textbook to adaptive library
- **Performance**
  - SIMD
  - Multicore
- **Customization**
- **Industry collaboration**
  - Intel IPP 6.0 will include Spiral generated code