Advanced Systems Lab

Spring 2022

Lecture: DSL-based program generation for performance (Spiral)

Instructor: Markus Püschel, Ce Zhang

TA: Joao Rivera, several more

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Spiral: DSL-Based Program Generation for Performance

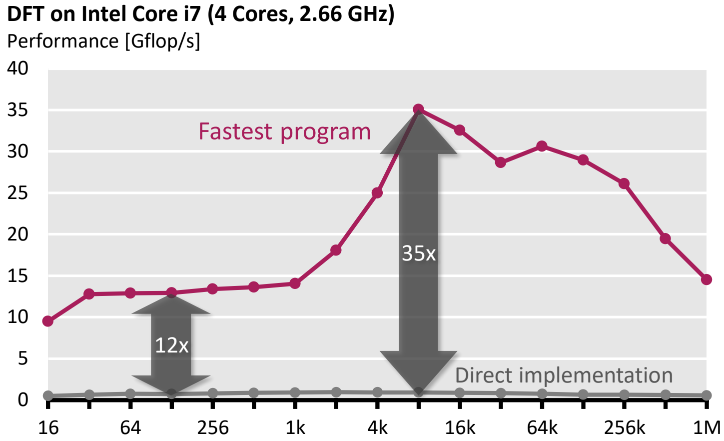
www.spiral.net (started 1998)

P, 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 Nicholas Rizzolo,
SPIRAL: Code Generation for DSP Transforms
Proceedings of the IEEE, special issue on «Program Generation, Optimization, and Adaptation'', Vol. 93, No. 2, pp. 232-275, 2005

P, Franz Franchetti and Yevgen Voronenko
Spiral
in Encyclopedia of Parallel Computing, Eds. David Padua, pp. 1920-1933, Springer 2011

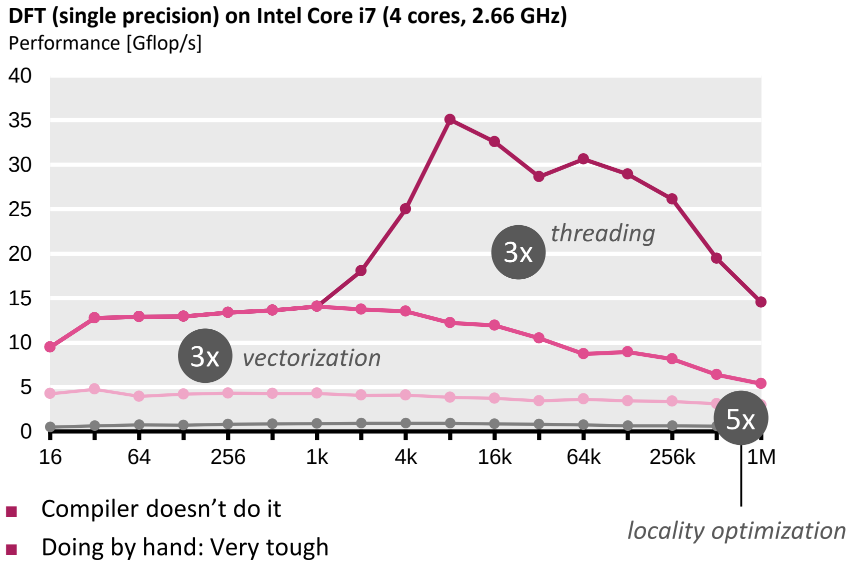
Franz Franchetti, Tze-Meng Low, Thom Popovici, Richard Veras, Daniele G. Spampinato, Jeremy Johnson, P, James C. Hoe and José M. F. Moura
SPIRAL: Extreme Performance Portability
Proceedings of the IEEE, special issue on ``From High Level Specification to High Performance Code'', Vol. 106, No. 11, 2018

The Problem: Example DFT



- Same number of operations
- Best compiler

DFT: Analysis

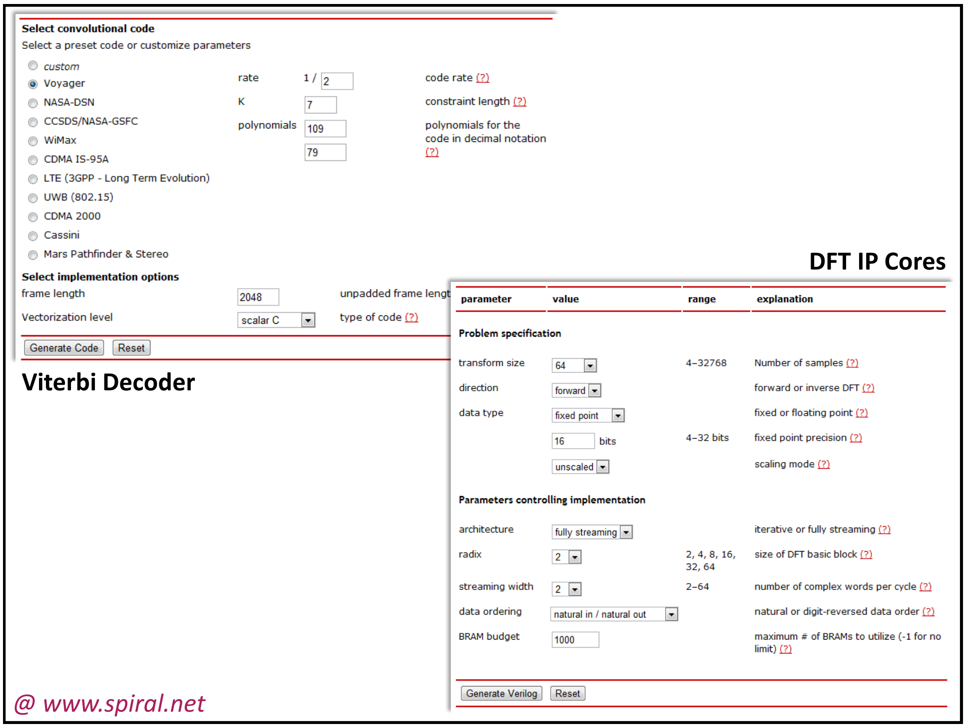


- Compiler doesn't do it
- Doing by hand: Very tough

Goal of Spiral:

Computer writes high performance library code

"click"



Possible Approach:

Capturing algorithm knowledge:
Domain-specific languages (DSLs)

Structural optimization:
Rewriting systems

High performance code style:
Compiler

Decision making for choices:
Machine learning

Organization

Spiral: Basic system

Vectorization

General input size

Results

Final remarks

Algorithms: Example FFT, n = 4

Fast Fourier transform (FFT)

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

Representation using matrix algebra

$$\mathbf{DFT}_4 = (\mathbf{DFT}_2 \otimes \mathrm{I}_2)\,\mathrm{T}^4_2\,(\mathrm{I}_2 \otimes \mathbf{DFT}_2)\,\mathrm{L}^4_2$$

SPL (Signal processing language): Mathematical, declarative, point-free

Divide-and-conquer algorithms = breakdown rules in SPL

Decomposition Rules (>200 for >40 Transforms)

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

$$\begin{vmatrix}\mathbf{RDFT}_n\\\mathbf{RDFT}'_n\\\mathbf{DHT}_n\\\mathbf{DHT}'_n\end{vmatrix} \to (P^\top_{k/2,m} \otimes I_2)\left(\begin{vmatrix}\mathbf{RDFT}_{2m}\\\mathbf{RDFT}'_{2m}\\\mathbf{DHT}_{2m}\\\mathbf{DHT}'_{2m}\end{vmatrix} \oplus \left(I_{k/2-1} \otimes_i D_{2m}\begin{vmatrix}\mathbf{rDFT}_{2m}(i/k)\\\mathbf{rDFT}_{2m}(i/k)\\\mathbf{rDHT}_{2m}(i/k)\\\mathbf{rDHT}_{2m}(i/k)\end{vmatrix}\right)\right)\left(\begin{vmatrix}\mathbf{RDFT}'_k\\\mathbf{RDFT}'_k\\\mathbf{DHT}'_k\\\mathbf{DHT}'_k\end{vmatrix} \otimes I_m\right), \quad k \text{ even},$$

$$\begin{vmatrix}\mathbf{rDFT}_{2n}(u)\\\mathbf{rDHT}_{2n}(u)\end{vmatrix} \to L^{2n}_m\left(I_k \otimes_i \begin{vmatrix}\mathbf{rDFT}_{2m}((i+u)/k)\\\mathbf{rDHT}_{2m}((i+u)/k)\end{vmatrix}\right)\left(\begin{vmatrix}\mathbf{rDFT}_{2k}(u)\\\mathbf{rDHT}_{2k}(u)\end{vmatrix} \otimes I_m\right),$$

$$\mathbf{RDFT\text{-}3}_n \to (Q^\top_{k/2,m} \otimes I_2)\,(I_k \otimes_i \mathbf{rDFT}_{2m})(i+1/2)/k))\,(\mathbf{RDFT\text{-}3}_k \otimes I_m), \quad k \text{ even},$$

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

$$\mathbf{DCT\text{-}3}_n \to \mathbf{DCT\text{-}2}^\top_n,$$

Decomposition rules = Algorithm knowledge in Spiral
(from ≈100 publications)

$$\mathbf{DFT}_n \to (\mathbf{DFT}_k \otimes \mathrm{I}_m)\,\mathrm{T}^n_m(\mathrm{I}_k \otimes \mathbf{DFT}_m)\,\mathrm{L}^n_k, \quad n = km$$

$$\mathbf{DFT}_n \to P_n(\mathbf{DFT}_k \otimes \mathbf{DFT}_m)Q_n, \quad n = km,\ \gcd(k,m) = 1$$

$$\mathbf{DCT\text{-}3}_n \to (\mathrm{I}_m \oplus \mathrm{J}_m)\,\mathrm{L}^n_m(\mathbf{DCT\text{-}3}_m(1/4) \oplus \mathbf{DCT\text{-}3}_m(3/4))$$

$$\cdot(\mathrm{F}_2 \otimes \mathrm{I}_m)\begin{bmatrix}\mathrm{I}_m & 0 \oplus -\mathrm{J}_{m-1}\\ & \frac{1}{\sqrt{2}}(\mathrm{I}_1 \oplus 2\,\mathrm{I}_m)\end{bmatrix}, \quad n = 2m$$

$$\mathbf{DCT\text{-}4}_n \to S_n\mathbf{DCT\text{-}2}_n\,\mathrm{diag}_{0\le k<n}(1/(2\cos((2k+1)\pi/4n)))$$

$$\mathbf{IMDCT}_{2m} \to (\mathrm{J}_m \oplus \mathrm{I}_m \oplus \mathrm{I}_m \oplus \mathrm{J}_m)\left(\left(\begin{bmatrix}1\\-1\end{bmatrix} \otimes \mathrm{I}_m\right) \oplus \left(\begin{bmatrix}-1\\-1\end{bmatrix} \otimes \mathrm{I}_m\right)\right)\mathrm{J}_{2m}\,\mathbf{DCT\text{-}4}_{2m}$$

$$\mathbf{WHT}_{2^k} \to \prod_{i=1}^{t}(\mathrm{I}_{2^{k_1+\cdots+k_{i-1}}} \otimes \mathbf{WHT}_{2^{k_i}} \otimes \mathrm{I}_{2^{k_{i+1}+\cdots+k_t}}), \quad k = k_1 + \cdots + k_t$$

$$\mathbf{DFT}_2 \to \mathrm{F}_2$$

$$\mathbf{DCT\text{-}2}_2 \to \mathrm{diag}(1, 1/\sqrt{2})\,\mathrm{F}_2$$

$$\mathbf{DCT\text{-}4}_2 \to \mathrm{J}_2\,\mathrm{R}_{13\pi/8}$$

Combining these rules yields many algorithms for every given transform

SPL to Code

SPL S	Pseudo code for $y = Sx$
$A_n B_n$	`<code for: t = Bx>` `<code for: y = At>`

$I_m \otimes A_n$	`for (i=0; i<m; i++)`
	`<code for:`
	`y[i*n:1:i*n+n-1] = A(x[i*n:1:i*n+n-1])>`

$A_m \otimes I_n$	`for (i=0; i<n; i++)`
	`<code for:`
	`y[i:n:i+m*n-n] = A(x[i:n:i+m*n-n])>`

D_n	`for (i=0; i<n; i++)` `y[i] = D[i]*x[i];`

L_k^{km}	`for (i=0; i<k; i++)`
	`for (j=0; j<m; j++)`
	`y[i*m+j] = x[j*k+i];`

F_2	`y[0] = x[0] + x[1];` `y[1] = x[0] - x[1];`

$$\mathrm{I}_m \otimes A_n = \begin{bmatrix} A_n & & \\ & \ddots & \\ & & A_n \end{bmatrix}$$

Correct code: ***easy*** *fast code:* ***very difficult***

Program Generation in Spiral

Transform $\mathbf{DFT}_8$

↓ *Decomposition rules*

Algorithm *(SPL)*

$$(\mathbf{DFT}_2 \otimes I_4)\, T_4^8\, (I_2 \otimes ((\mathbf{DFT}_2 \otimes I_2)\, T_2^4\, (I_2 \otimes \mathbf{DFT}_2)\, L_2^4))\, L_2^8$$

parallelization
vectorization

↓

Algorithm *(∑-SPL)*

$$\sum \left(S_j\, \mathbf{DFT}_2\, G_j\right) \sum \left(\sum \left(S_{k,l}\, \mathrm{diag}\left(\mathrm{t}_{k,l}\right)\, \mathbf{DFT}_2\, G_l\right) \sum \left(S_m\, \mathrm{diag}\left(\mathrm{t}_m\right)\, \mathbf{DFT}_2\, G_{k,m}\right)\right)$$

locality
optimization

↓

C Program

```
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;
< more lines>
```

basic block
optimizations

+ search or learning
for further tuning

Organization

Spiral: Basic system

Vectorization

General input size

Results

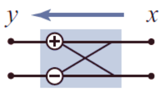
Final remarks

Example: Vectorization in Spiral

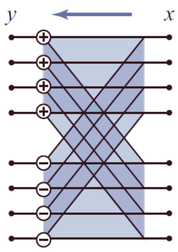
Goal: Translate SPL expressions directly into SIMD code

Relationship SPL expressions ↔ vectorization?

$$y = \mathbf{DFT}_2\, x$$



$$y = \left(\mathbf{DFT}_2 \otimes \mathrm{I}_4\right) x$$



one addition
one subtraction

one (4-way) vector addition
one (4-way) vector subtraction

Step 1: Identify "Good" Vector Constructs

Vector length: ν

Good (= easily vectorizable) SPL constructs:

$$A \otimes \mathrm{I}_\nu$$

$$\mathrm{L}_\nu^{\nu^2},\ \mathrm{L}_2^{2\nu}, \mathrm{L}_\nu^{2\nu}$$ *base cases*

SPL expressions recursively built from those

Idea: Convert a given SPL expression into a "good" SPL expression through rewriting (structural manipulation)

Step 2: Find Manipulation Rules

$$\mathrm{L}_n^{n\nu} \rightarrow \left(\mathrm{I}_{n/\nu} \otimes \mathrm{L}_\nu^{\nu^2}\right)\left(\mathrm{L}_{n/\nu}^{n} \otimes \mathrm{I}_\nu\right)$$

$$\mathrm{L}_\nu^{n\nu} \rightarrow \left(\mathrm{L}_\nu^{n} \otimes \mathrm{I}_\nu\right)\left(\mathrm{I}_{n/\nu} \otimes \mathrm{L}_\nu^{\nu^2}\right)$$

$$\mathrm{L}_m^{mn} \rightarrow \left(\mathrm{L}_m^{mn/\nu} \otimes \mathrm{I}_\nu\right)\left(\mathrm{I}_{mn/\nu^2} \otimes \mathrm{L}_\nu^{\nu^2}\right)\left(\mathrm{I}_{n/\nu} \otimes \mathrm{L}_{m/\nu}^{m} \otimes \mathrm{I}_\nu\right)$$

$$\mathrm{I}_l \otimes \mathrm{L}_n^{kmn} \otimes \mathrm{I}_r \rightarrow \left(\mathrm{I}_l \otimes \mathrm{L}_n^{kn} \otimes \mathrm{I}_{mr}\right)\left(\mathrm{I}_{kl} \otimes \mathrm{L}_n^{mn} \otimes \mathrm{I}_r\right)$$

$$\mathrm{I}_l \otimes \mathrm{L}_n^{kmn} \otimes \mathrm{I}_r \rightarrow \left(\mathrm{I}_l \otimes \mathrm{L}_{kn}^{kmn} \otimes \mathrm{I}_r\right)\left(\mathrm{I}_l \otimes \mathrm{L}_{mn}^{kmn} \otimes \mathrm{I}_r\right)$$

$$\mathrm{I}_l \otimes \mathrm{L}_{km}^{kmn} \otimes \mathrm{I}_r \rightarrow \left(\mathrm{I}_{kl} \otimes \mathrm{L}_m^{mn} \otimes \mathrm{I}_r\right)\left(\mathrm{I}_l \otimes \mathrm{L}_k^{kn} \otimes \mathrm{I}_{mr}\right)$$

$$\mathrm{I}_l \otimes \mathrm{L}_{km}^{kmn} \otimes \mathrm{I}_r \rightarrow \left(\mathrm{I}_l \otimes \mathrm{L}_k^{kmn} \otimes \mathrm{I}_r\right)\left(\mathrm{I}_l \otimes \mathrm{L}_m^{kmn} \otimes \mathrm{I}_r\right)$$

Manipulation rules = SIMD knowledge in Spiral

$$\left(\mathrm{I}_m \otimes A^{n\times n}\right)\mathrm{L}_m^{mn} \rightarrow \left(\mathrm{I}_{m/\nu} \otimes \mathrm{L}_\nu^{n\nu}\left(A^{n\times n} \otimes \mathrm{I}_\nu\right)\right)\left(\mathrm{L}_{m/\nu}^{mn/\nu} \otimes \mathrm{I}_\nu\right)$$

$$\mathrm{L}_n^{mn}\left(\mathrm{I}_m \otimes A^{n\times n}\right) \rightarrow \left(\mathrm{L}_n^{mn/\nu} \otimes \mathrm{I}_\nu\right)\left(\mathrm{I}_{m/\nu} \otimes \left(A^{n\times n} \otimes \mathrm{I}_\nu\right)\mathrm{L}_n^{n\nu}\right)$$

$$\left(\mathrm{I}_k \otimes \left(\mathrm{I}_m \otimes A^{n\times n}\right)\mathrm{L}_m^{mn}\right)\mathrm{L}_k^{kmn} \rightarrow \left(\mathrm{L}_k^{km} \otimes \mathrm{I}_n\right)\left(\mathrm{I}_m \otimes \left(\mathrm{I}_k \otimes A^{n\times n}\right)\mathrm{L}_k^{kn}\right)\left(\mathrm{L}_m^{mn} \otimes \mathrm{I}_k\right)$$

$$\mathrm{L}_{mn}^{kmn}\left(\mathrm{I}_k \otimes \mathrm{L}_n^{mn}\left(\mathrm{I}_m \otimes A^{n\times n}\right)\right) \rightarrow \left(\mathrm{L}_n^{mn} \otimes \mathrm{I}_k\right)\left(\mathrm{I}_m \otimes \mathrm{L}_n^{kn}\left(\mathrm{I}_k \otimes A^{n\times n}\right)\right)\left(\mathrm{L}_m^{km} \otimes \mathrm{I}_n\right)$$

$$\overline{A\,B} \rightarrow \overline{A}\,\overline{B}$$

$$\overline{A^{m\times m} \otimes \mathrm{I}_\nu} \rightarrow \left(\mathrm{I}_m \otimes \mathrm{L}_\nu^{2\nu}\right)\left(\overline{A^{m\times m}} \otimes \mathrm{I}_\nu\right)\left(\mathrm{I}_m \otimes \mathrm{L}_2^{2\nu}\right)$$

$$\overline{\mathrm{I}_m \otimes A^{n\times n}} \rightarrow \mathrm{I}_m \otimes \overline{A^{n\times n}}$$

$$\overline{D} \rightarrow \left(\mathrm{I}_{n/\nu} \otimes \mathrm{L}_\nu^{2\nu}\right)\vec{D}\left(\mathrm{I}_{n/\nu} \otimes \mathrm{L}_2^{2\nu}\right)$$

$$\overline{P} \rightarrow P \otimes \mathrm{I}_2$$

Example

$$\underbrace{\mathbf{DFT}_{mn}}_{\text{vec}(\nu)} \rightarrow \underbrace{(\mathbf{DFT}_m \otimes \mathrm{I}_n)\mathsf{T}_n^{mn}(\mathrm{I}_m \otimes \mathbf{DFT}_n)\,\mathsf{L}_m^{mn}}_{\text{vec}(\nu)}$$

$$\dots$$

$$\dots$$

$$\dots$$

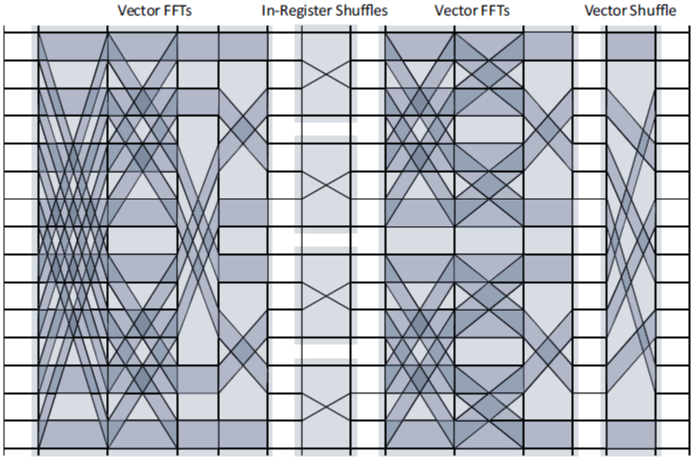
$$\rightarrow \left(\mathrm{I}_{\frac{mn}{\nu}} \otimes \mathsf{L}_{\nu}^{2\nu}\right)\left(\overline{\mathbf{DFT}_m \otimes \mathrm{I}_{\frac{n}{\nu}}} \otimes \mathrm{I}_\nu\right)\overline{\mathsf{T}}'^{mn}_{n}$$

$$\left(\mathrm{I}_{\frac{m}{\nu}} \otimes \left(\mathsf{L}_{\nu}^{2n} \otimes \mathrm{I}_\nu\right)\left(\mathrm{I}_{\frac{2n}{\nu}} \otimes \mathsf{L}_{\nu}^{\nu^2}\right)\left(\mathrm{I}_{\frac{n}{\nu}} \otimes \mathsf{L}_{2}^{2\nu} \otimes \mathrm{I}_\nu\right)\left(\overline{\mathbf{DFT}}_n \otimes \mathrm{I}_\nu\right)\right)\left(\mathsf{L}_{\frac{m}{\nu}}^{\frac{mn}{\nu}} \otimes \mathsf{L}_{2}^{2\nu}\right)$$

vectorized arithmetic

vectorized data accesses

Sketch for complex ν = 2



$$\Big(\big((\mathrm{DFT}_2 \otimes I_2)T_2^4(I_2 \otimes \mathrm{DFT}_2)L_2^4\big) \otimes I_2\big) \otimes I_2\Big)T_4^{16}\Big(I_2 \otimes (L_2^4 \otimes I_2)(I_2 \otimes L_2^4)\big((\mathrm{DFT}_2 \otimes I_2)T_2^4(I_2 \otimes \mathrm{DFT}_2)L_2^4\big) \otimes I_2\Big)(L_2^8 \otimes I_2)$$

Automatically Generate Base Case Library

Goal: Given instruction set, generate base cases

$$\nu = 4: \quad \left\{ L^4_2,\ I_2 \otimes L^4_2,\ L^4_2 \otimes I_2,\ L^8_2,\ L^8_4 \right\}$$

Idea: Instructions as matrices + search

```
y = _mm_unpacklo_ps(x0, x1);
```

$$y = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0 \end{bmatrix} \begin{bmatrix} \vec{x}_0 \\ \vec{x}_1 \end{bmatrix}$$

```
y = _mm_shuffle_ps(x0, x1, _MM_SHUFFLE(1,2,1,2));
```

$$y = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0 \end{bmatrix} \begin{bmatrix} \vec{x}_0 \\ \vec{x}_1 \end{bmatrix}$$

```
y = _mm_shuffle_ps(x0, x1, _MM_SHUFFLE(3,4,3,4));
```

$$y = \begin{bmatrix} 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{bmatrix} \begin{bmatrix} \vec{x}_0 \\ \vec{x}_1 \end{bmatrix}$$

$$\left[\begin{array}{cccccccc} 1&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0\\ \hline 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{array}\right]$$

```
y0 = _mm_unpacklo_ps(x[0], x[1]);
y1 = _mm_shuffle_ps(x0, x1,
        _MM_SHUFFLE(3,4,3,4));
```

No base case

Automatically Generate Base Case Library

Goal: Given instruction set, generate base cases

$$\nu = 4: \quad \left\{ L^4_2,\ I_2 \otimes L^4_2,\ L^4_2 \otimes I_2,\ L^8_2,\ L^8_4 \right\}$$

Idea: Instructions as matrices + search

```
y = _mm_unpacklo_ps(x0, x1);
```

$$y = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0 \end{bmatrix} \begin{bmatrix} \vec{x}_0 \\ \vec{x}_1 \end{bmatrix}$$

```
y = _mm_shuffle_ps(x0, x1, _MM_SHUFFLE(1,2,1,2));
```

$$y = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0 \end{bmatrix} \begin{bmatrix} \vec{x}_0 \\ \vec{x}_1 \end{bmatrix}$$

```
y = _mm_shuffle_ps(x0, x1, _MM_SHUFFLE(3,4,3,4));
```

$$y = \begin{bmatrix} 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{bmatrix} \begin{bmatrix} \vec{x}_0 \\ \vec{x}_1 \end{bmatrix}$$

$$\left[\begin{array}{cccccccc} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ \hline 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{array}\right]$$

```
y0 = _mm_shuffle_ps(x0, x1,
        _MM_SHUFFLE(1,2,1,2));
y1 = _mm_shuffle_ps(x0, x1,
        _MM_SHUFFLE(3,4,3,4));
```

$L^4_2 \otimes I_2$
Base case

Same Approach for Different Paradigms

Threading:

$$\underbrace{\mathbf{DFT}_{mn}}_{\mathsf{smp}(p,\mu)} \to \underbrace{\left((\mathbf{DFT}_m \otimes I_n)\, T_n^{mn} (I_m \otimes \mathbf{DFT}_n)\, L_m^{mn}\right)}_{\mathsf{smp}(p,\mu)}$$

$$\dots$$

$$\to \underbrace{\left(\mathbf{DFT}_m \otimes I_n\right)}_{\mathsf{smp}(p,\mu)} \underbrace{T_n^{mn}}_{\mathsf{smp}(p,\mu)} \underbrace{\left(I_m \otimes \mathbf{DFT}_n\right)}_{\mathsf{smp}(p,\mu)} \underbrace{L_m^{nm}}_{\mathsf{smp}(p,\mu)}$$

$$\dots$$

$$\to \left((L_m^{mp} \otimes I_{n/p\mu}) \otimes_\mu I_\mu\right)\left(I_p \otimes_\| (\mathbf{DFT}_m \otimes I_{n/p})\right)\left((L_p^{mp} \otimes I_{n/p\mu}) \otimes_\mu I_\mu\right)$$
$$\left(\bigoplus_{i=0}^{p-1}{}_\| \, T_n^{mn,i}\right)\left(I_p \otimes_\| (I_{m/p} \otimes \mathbf{DFT}_n)\right)\left(I_p \otimes_\| L_{m/p}^{mn/p}\right)\left((L_p^{pn} \otimes I_{m/p\mu}) \otimes_\mu I_\mu\right)$$

Vectorization:

$$\underbrace{\left(\overline{\mathbf{DFT}_{mn}}\right)}_{\mathsf{vec}(\nu)} \to \underbrace{\left((\mathbf{DFT}_m \otimes I_n)\, T_n^{mn} (I_m \otimes \mathbf{DFT}_n)\, L_m^{mn}\right)}_{\mathsf{vec}(\nu)}$$

$$\dots$$

$$\to \overleftrightarrow{\underbrace{(\mathsf{DFT}_m \otimes I_n)}_{\mathsf{vec}(\nu)}}^{\nu} \overline{\underbrace{(T_n^{mn})}_{\mathsf{vec}(\nu)}}^{\nu} \overrightarrow{\underbrace{(I_m \otimes \mathsf{DFT}_n)\, L_m^{mn}}_{\mathsf{vec}(\nu)}}^{\nu}$$

$$\dots$$

$$\to \left(I_{mn/\nu} \otimes \underbrace{L_\nu^{2\nu}}_{\mathsf{sse}}\right)\left(\overline{\mathsf{DFT}_m \otimes I_{n/\nu}} \vec{\otimes} I_\nu\right)\overline{\underbrace{\left(T_n^{mn}\right)}_{\mathsf{sse}}}^{\nu}$$
$$\left(I_{m/\nu} \otimes (\overline{L_\nu^n} \vec{\otimes} I_\nu)(I_{n/\nu} \otimes (L_\nu^{2\nu} \vec{\otimes} I_\nu)(I_2 \otimes \underbrace{L_\nu^{\nu^2}}_{\mathsf{sse}})(L_2^{2\nu} \vec{\otimes} I_\nu))(\overline{\mathsf{DFT}_n} \vec{\otimes} I_\nu)\right)$$
$$\left((L_m^{mn} \otimes I_2) \vec{\otimes} I_\nu\right)\left(I_{mn/\nu} \otimes \underbrace{L_2^{2\nu}}_{\mathsf{sse}}\right)$$

GPUs:

$$\underbrace{\left(\mathbf{DFT}_{r^k}\right)}_{\mathsf{gpu}(t,c)} \to \underbrace{\left(\prod_{i=0}^{k-1} L_r^{r^k}\left(I_{r^{k-1}} \otimes \mathbf{DFT}_r\right)\left(L_{r^{k-i-1}}^{r^k}(I_{r^i} \otimes T_{r^{k-i-1}}^{r^{k-i}})\underbrace{L_{r^{i+1}}^{r^k}}_{\mathsf{vec}(c)}\right)\right) R_r^{r^k}}_{\mathsf{gpu}(t,c)}$$

$$\dots$$

$$\to \left(\prod_{i=0}^{k-1} (L_r^{r^n/2} \vec{\otimes} I_2)\left(I_{r^{n-1}/2} \otimes_\times \underbrace{(\mathbf{DFT}_r \vec{\otimes} I_2)\, L_r^{2r}}_{\mathsf{shd}(t,c)}\right) T_i\right)$$
$$(L_r^{r^n/2} \vec{\otimes} I_2)(I_{r^{n-1}/2} \otimes_\times \underbrace{L_r^{2r}}_{\mathsf{shd}(t,c)})(R_r^{r^{n-1}} \vec{\otimes} I_r)$$

Verilog for FPGAs:

$$\underbrace{\left(\mathbf{DFT}_{r^k}\right)}_{\mathsf{stream}(r^s)} \to \underbrace{\left[\prod_{i=0}^{k-1} L_r^{r^k}\left(I_{r^{k-1}} \otimes \mathbf{DFT}_r\right)\left(L_{r^{k-i-1}}^{r^k}(I_{r^i} \otimes T_{r^{k-i-1}}^{r^{k-i}})\, L_{r^{i+1}}^{r^k}\right)\right] R_r^{r^k}}_{\mathsf{stream}(r^s)}$$

$$\dots$$

$$\to \left[\prod_{i=0}^{k-1} \underbrace{L_r^{r^k}}_{\mathsf{stream}(r^s)} \underbrace{\left(I_{r^{k-1}} \otimes \mathbf{DFT}_r\right)}_{\mathsf{stream}(r^s)} \underbrace{\left(L_{r^{k-i-1}}^{r^k}(I_{r^i} \otimes T_{r^{k-i-1}}^{r^{k-i}})\, L_{r^{i+1}}^{r^k}\right)}_{\mathsf{stream}(r^s)}\right] \underbrace{R_r^{r^k}}_{\mathsf{stream}(r^s)}$$

$$\dots$$

$$\to \left[\prod_{i=0}^{k-1} \underbrace{L_r^{r^k}}_{\mathsf{stream}(r^s)} \left(I_{r^{k-s-1}} \otimes_s (I_{r^{s-1}} \otimes \mathbf{DFT}_r)\right) \underbrace{T_i'}_{\mathsf{stream}(r^s)}\right] \underbrace{R_r^{r^k}}_{\mathsf{stream}(r^s)}$$

- Rigorous, correct by construction
- Overcomes compiler limitations

Organization

Spiral: Basic system

Vectorization

General input size

Results

Final remarks

Challenge: General Size Libraries

So far:

Code specialized to fixed input size

```
DFT_384(x, y) {
  …
  for(i = …) {
   t[2i]   = x[2i] + x[2i+1]
   t[2i+1] = x[2i] - x[2i+1]
  }
  …
}
```

- Algorithm fixed
- Nonrecursive code

Challenge:

Library for general input size

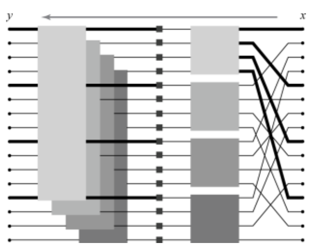
```
DFT(n, x, y) {
  …
  for(i = …) {
    DFT_strided(m, x+mi, y+i, 1, k)
  }
  …
}
```

- Algorithm cannot be fixed
- Recursive code
- Creates many challenges

Challenge: Recursion Steps

Cooley-Tukey FFT

$$y = (\mathbf{DFT}_k \otimes I_m) T_m^{km} (I_k \otimes \mathbf{DFT}_m) L_k^{km} x$$



Implementation that increases locality (e.g., FFTW 2.x)

```
void DFT(int n, cpx *y, cpx *x) {
  int k = choose_dft_radix(n);
  …
  for (int i=0; i < k; ++i)
    DFTrec(m, y + m*i, x + i, k, 1);
  for (int j=0; j < m; ++j)
    DFTscaled(k, y + j, t[j], m);
}
```

Σ–SPL : Basic Idea

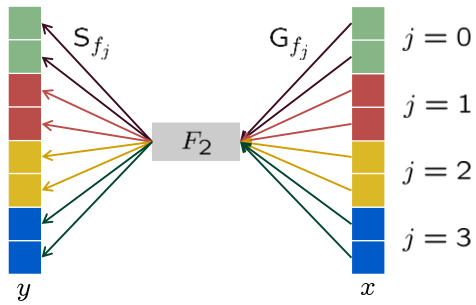
Four additional matrix constructs: Σ, G, S, Perm

- *Σ (sum)* — ***explicit loop***
- *G_f (gather)* — ***load data with index mapping f***
- *S_f (scatter)* — ***store data with index mapping f***
- *$Perm_f$* — ***permute data with the index mapping f***

Σ-SPL formulas = matrix factorizations

Example: $y = (I_4 \otimes F_2)x \;\rightarrow\; y = \sum_{j=0}^{3} \mathsf{S}_{f_j} F_2 \mathsf{G}_{f_j} x$

$$y = \begin{bmatrix} F_2 & & & \\ & F_2 & & \\ & & F_2 & \\ & & & F_2 \end{bmatrix} x$$



Find Recursion Step Closure

Voronenko, 2008

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

$$\downarrow$$

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

$$\downarrow$$

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

$$\downarrow$$

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

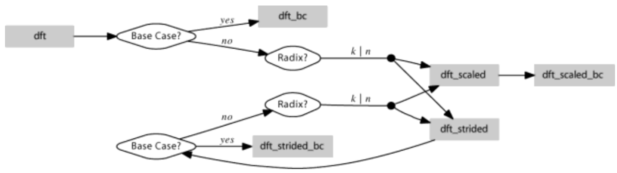
$$\downarrow$$

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

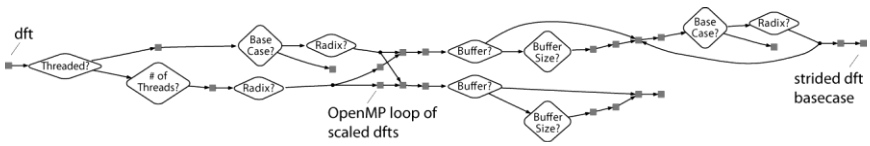
Repeat until closure

Recursion Step Closure: Examples

DFT: scalar code (like FFTW 2.x)



DFT: full-fledged (vectorized and parallel code)



Summary: Complete Automation for Transforms

- **Memory hierarchy optimization**
  Rewriting and search for algorithm selection
  Rewriting for loop optimizations
- **Vectorization**
  Rewriting
- **Parallelization**
  Rewriting

fixed input size code

- **Derivation of library structure**
  Rewriting
  Other methods

general input size library

Organization

Spiral: Basic system

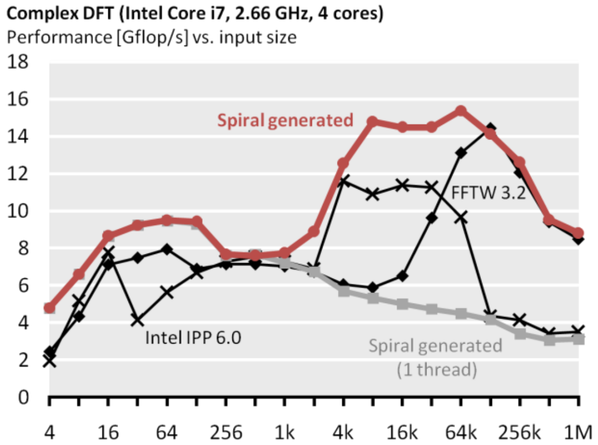
Vectorization

General input size

Results

Final remarks

DFT on Intel Multicore



$$\mathbf{DFT}_n \to (\mathbf{DFT}_k \otimes I_m) T^n_m (I_k \otimes \mathbf{DFT}_m) L^n_k$$
$$\mathbf{DFT}_n \to P^{\top}_{k/2,2m} \left(\mathbf{DFT}_{2m} \oplus \left(I_{k/2-1} \otimes_i C_{2m}\, \mathbf{rDFT}_{2m}(i/k)\right)\right) (\mathbf{RDFT}_k \otimes I_m)$$
$$\mathbf{RDFT}_n \to (P^{\top}_{k/2,m} \otimes I_2) \left(\mathbf{RDFT}_{2m} \oplus \left(I_{k/2-1} \otimes_i D_{2m}\, \mathbf{rDFT}_{2m}(i/k)\right)\right) (\mathbf{RDFT}_k \otimes I_m)$$
$$\mathbf{rDFT}_{2n}(u) \to L^{2n}_m \left(I_k \otimes_i \mathbf{rDFT}_{2m}((i+u)/k)\right) (\mathbf{rDFT}_{2k}(u) \otimes I_m)$$

Spiral → 5MB vectorized, threaded, general-size, adaptive library

Generating 100s of FFTWs

PhD thesis Voronenko, 2009

$$\mathbf{DFT}_n \to P_{k/2,2m}^\top \left(\mathbf{DFT}_{2m} \oplus \left(I_{k/2-1} \otimes_i C_{2m}\, \mathbf{rDFT}_{2m}(i/k)\right)\right) \left(\mathbf{RDFT}'_k \otimes I_m\right), \quad k \text{ even},$$

$$\begin{vmatrix}\mathbf{RDFT}_n \\ \mathbf{RDFT}'_n \\ \mathbf{DHT}_n \\ \mathbf{DHT}'_n\end{vmatrix} \to (P_{k/2,m}^\top \otimes I_2) \left(\begin{vmatrix}\mathbf{RDFT}_{2m} \\ \mathbf{RDFT}'_{2m} \\ \mathbf{DHT}_{2m} \\ \mathbf{DHT}'_{2m}\end{vmatrix} \oplus \left(I_{k/2-1} \otimes_i D_{2m} \begin{vmatrix}\mathbf{rDFT}_{2m}(i/k) \\ \mathbf{rDFT}_{2m}(i/k) \\ \mathbf{rDHT}_{2m}(i/k) \\ \mathbf{rDHT}_{2m}(i/k)\end{vmatrix}\right)\right) \left(\begin{vmatrix}\mathbf{RDFT}'_k \\ \mathbf{RDFT}'_k \\ \mathbf{DHT}'_k \\ \mathbf{DHT}'_k\end{vmatrix} \otimes I_m\right), \quad k \text{ even},$$

$$\begin{vmatrix}\mathbf{rDFT}_{2n}(u) \\ \mathbf{rDHT}_{2n}(u)\end{vmatrix} \to L_m^{2n} \left(I_k \otimes_i \begin{vmatrix}\mathbf{rDFT}_{2m}((i+u)/k) \\ \mathbf{rDHT}_{2m}((i+u)/k)\end{vmatrix}\right) \left(\begin{vmatrix}\mathbf{rDFT}_{2k}(u) \\ \mathbf{rDHT}_{2k}(u)\end{vmatrix} \otimes I_m\right),$$

$$\mathbf{RDFT\text{-}3}_n \to (Q_{k/2,m}^\top \otimes I_2)\,(I_k \otimes_i \mathbf{rDFT}_{2m})(i+1/2)/k))\,(\mathbf{RDFT\text{-}3}_k \otimes I_m), \quad k \text{ even},$$

$$\mathbf{DCT\text{-}2}_n \to P_{k/2,2m}^\top \left(\mathbf{DCT\text{-}2}_{2m} K_2^{2m} \oplus \left(I_{k/2-1} \otimes N_{2m}\, \mathbf{RDFT\text{-}3}_{2m}^\top\right)\right) B_n (L_{k/2}^{n/2} \otimes I_2)(I_m \otimes \mathbf{RDFT}'_k) Q_{m/2,k},$$

$$\mathbf{DCT\text{-}3}_n \to \mathbf{DCT\text{-}2}_n^\top,$$

$$\mathbf{DCT\text{-}4}_n \to Q_{k/2,2m}^\top \left(I_{k/2} \otimes N_{2m}\, \mathbf{RDFT\text{-}3}_{2m}^\top\right) B'_n (L_{k/2}^{n/2} \otimes I_2)(I_m \otimes \mathbf{RDFT\text{-}3}_k) Q_{m/2,k}.$$

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

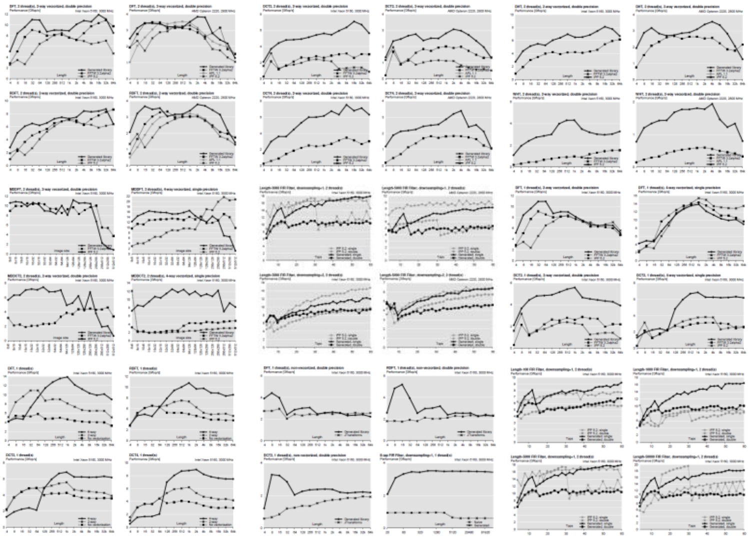
Generating 100s of FFTWs

PhD thesis Voronenko, 2009

	Code size	
Transform	non-parallelized	parallelized
no vectorization		
DFT	13.1 KLOC / 0.59 MB	10.3 KLOC / 0.45 MB
RDFT	8.5 KLOC / 0.36 MB	8.8 KLOC / 0.39 MB
DHT	9.1 KLOC / 0.40 MB	9.4 KLOC / 0.39 MB
DCT-2	12.0 KLOC / 0.55 MB	12.4 KLOC / 0.57 MB
DCT-3	12.0 KLOC / 0.56 MB	12.3 KLOC / 0.59 MB
DCT-4	6.8 KLOC / 0.33 MB	7.1 KLOC / 0.35 MB
WHT	5.6 KLOC / 0.21 MB	—
2-way vectorization		
DFT	14.8 KLOC / 0.73 MB	15.0 KLOC / 0.74 MB
RDFT	15.6 KLOC / 0.76 MB	16.0 KLOC / 0.81 MB
scaled RDFT	16.0 KLOC / 0.78 MB	—
DHT	16.9 KLOC / 0.83 MB	17.2 KLOC / 0.87 MB
DCT-2	20.7 KLOC / 1.10 MB	21.0 KLOC / 1.09 MB
DCT-3	27.9 KLOC / 1.56 MB	28.2 KLOC / 1.59 MB
DCT-4	7.8 KLOC / 0.47 MB	8.1 KLOC / 0.50 MB
WHT	6.9 KLOC / 0.32 MB	5.8 KLOC / 0.26 MB
FIR Filter	167 KLOC / 7.75 MB	120 KLOC / 5.12 MB
Downsampled FIR Filter	100 KLOC / 4.2 MB	68 KLOC / 2.76 MB
4-way vectorization		
DFT	17.9 KLOC / 1.09 MB	18.2 KLOC / 1.11 MB
RDFT	16.2 KLOC / 0.86 MB	16.5 KLOC / 0.91 MB
scaled RDFT	16.5 KLOC / 0.88 MB	—
DHT	17.9 KLOC / 1.02 MB	18.3 KLOC / 1.04 MB
DCT-2	23.3 KLOC / 1.50 MB	23.6 KLOC / 1.53 MB
DCT-3	32.0 KLOC / 2.17 MB	32.3 KLOC / 2.20 MB
DCT-4	8.3 KLOC / 0.63 MB	8.6 KLOC / 0.66 MB
WHT	8.5 KLOC / 0.53 MB	6.9 KLOC / 0.4 MB
2D DFT	20.6 KLOC / 1.32 MB	20.8 KLOC / 1.33 MB
2D DCT-2	27.0 KLOC / 2.1 MB	27.2 KLOC / 2.11 MB
FIR Filter	109 KLOC / 5.69 MB	74 KLOC / 3.44 MB
Downsampled FIR Filter	151 KLOC / 7.7 MB	92 KLOC / 4.61 MB

Generating 100s of FFTWs

PhD thesis Voronenko, 2009



Computer generated Functions for Intel IPP 6.0

3984 C functions

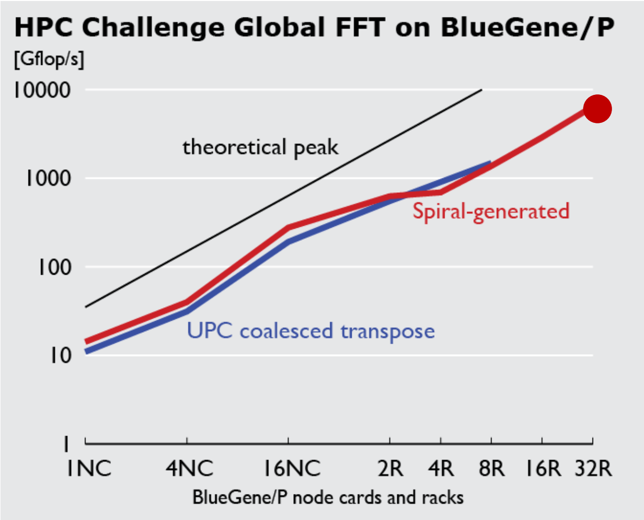
1M lines of code

- *Transforms: DFT (fwd+inv), RDFT (fwd+inv), DCT2, DCT3, DCT4, DHT, WHT*
- *Sizes: 2–64 (DFT, RDFT, DHT); 2-powers (DCTs, WHT)*
- *Precision: single, double*
- *Data type: scalar, SSE, AVX (DFT, DCT), LRB (DFT)*

Computer generated

Results: SpiralGen Inc.

Very Large Scale: BG/P



2010 HPC Challenge Class I Award, Almasi et al.

6.4 Tflop/s

32 racks
= 32K node cards
= 128K cores

Organization

Spiral: Basic system

Vectorization

General input size

Results

Final remarks

Spiral: Summary

Spiral:

Successful approach to automating the development of computing software

Commercial proof-of-concept

$\mathbf{DFT}_{64}$

```
void dft64(float  *Y, float  *X) {
  __m512 U912, U913, U914, U915,...
  __m512  *a2153, *a2155;
  a2153 = ((__m512  *) X);  s1107 = *(a2153);
  s1108 = *((a2153 + 4));   t1323 = _mm512_add_ps(s1107,s1108);
  t1324 = _mm512_sub_ps(s1107,s1108);
    <many more lines>
  U926 = _mm512_swizupconv_r32(…);
  s1121 = _mm512_madd231_ps(_mm512_mul_ps(_mm512_mask_or_pi(
    _mm512_set_1to16_ps(0.70710678118654757),0xAAAA,a2154,U926),t1341),
    _mm512_mask_sub_ps(_mm512_set_1to16_ps(0.70710678118654757),…),
    _mm512_swizupconv_r32(t1341,_MM_SWIZ_REG_CDAB));
  U927 = _mm512_swizupconv_r32
    <many more lines>
}
```

Key ideas:

Algorithm knowledge:
Domain specific symbolic representation

$$\mathbf{DFT}_4 \rightarrow (\mathbf{DFT}_2 \otimes \mathrm{I}_2)\mathsf{T}^4_2(\mathrm{I}_2 \otimes \mathbf{DFT}_2)\,\mathsf{L}^4_2$$

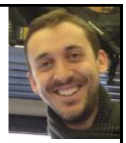
Platform knowledge:
Tagged rewrite rules, SIMD specification

$$\underbrace{\mathrm{I}_m \otimes A_n}_{\mathsf{smp}(p,\mu)} \rightarrow \mathrm{I}_p \otimes_{\|} \left(\mathrm{I}_{m/p} \otimes A_n\right)$$

Glimpse of other topics …

LGen: Generator for Basic Linear Algebra

Spampinato & P, CGO 2014



BLAC $y = x^T(A+B)y + \delta$

Algorithm: Tiling decision and propagation *(LL)* $\left[y = x^T(A+B)y + \delta\right]_{2,3}$

Algorithm *(Σ-LL)* $\sum_{i,j,i',j'} S_i S_{i'} \left(G_{i'} G_i A G_j G_{j'}\right)\left(G_{j'} G_j x\right) \ldots$

C Program

```
void kernel(float *x, float *A, float *B, …) {
float t0_64_0, t0_64_1, t0_64_2, t0_64_3 …;
    t0_57_0 = A[0];
    t0_56_0 = A[1];
    …
    t0_59_0 = t0_57_0 + t0_33_0;
    t0_63_0 = t0_59_0 * t0_9_0;
    t0_59_1 = t0_56_0 + t0_32_0;
    t0_60_0 = t0_59_1 * t0_8_0;
    < many more lines>
```

vectorization

locality optimization

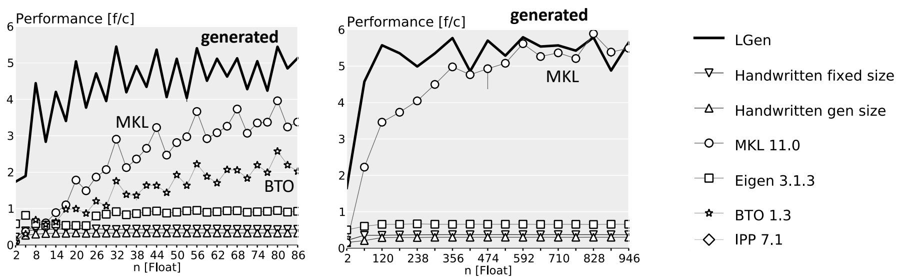
code style

code level optimization

LGen: Sample Results

$C = \alpha AB + \beta C$

$C = \alpha(A_0 + A_1)^T B + \beta C$



$A \in \mathbb{R}^{n\times 4}$

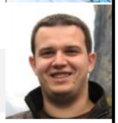
$B \in \mathbb{R}^{4\times n}$

$A_0 \in \mathbb{R}^{4\times 4}$

$B \in \mathbb{R}^{4\times n}$

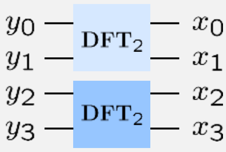
PL Support: Example Code Style

Ofenbeck, Rompf, Stojanov, Odersky & P, GPCE 2012



SPL $y = (\mathrm{I}_2 \otimes \mathbf{DFT}_2)x$

Data flow graph



Scala function

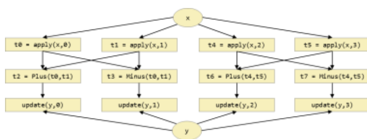
```
def f(x: Array[Double], y: Array[Double]) = {
    for (i <- 0 until 2) {
        y(2*i)   = x(i*2) + x(i*2+1)
        y(2*i+1) = x(i*2) - x(i*2+1)
    }
}
```

```
def f(x: Array[Rep[Double]],
      y: Array[Rep[Double]]) = {
      for (i <- 0 until 2) {
        y(2*i)   = x(i*2) + x(i*2+1)
        y(2*i+1) = x(i*2) - x(i*2+1)
      }
    }
```

scalarized

```
t0 = s0 + s1;
t1 = s0 - s1;
t2 = s2 + s3;
t2 = s2 - s3;
```

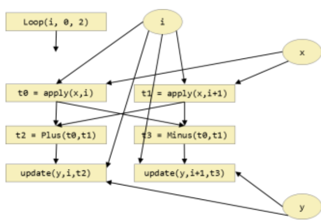
```
def f(x: Rep[Array[Double]],
      y: Rep[Array[Double]]) = {
      for (i <- 0 until 2) {
        y(2*i)   = x(i*2) + x(i*2+1)
        y(2*i+1) = x(i*2) - x(i*2+1)
      }
    }
```



unrolled, scalar repl.

```
t0 = x[0];
t1 = x[1];
t2 = t0 + t1;
y[0] = t2;
t3 = t0 - t1;
y[1] = t3;
t4 = x[0];
t5 = x[1];
t6 = t4 + x5;
y[0] = t6;
t7 = t4 – x5;
y[3] = t7;
```

```
def f(x: Rep[Array[Double]],
      y: Rep[Array[Double]]) = {
      for (i <- 0 until 2: Rep[Range]) {
        y(2*i)   = x(i*2) + x(i*2+1)
        y(2*i+1) = x(i*2) - x(i*2+1)
      }
    }
```



looped, scalar repl.

```
for (int i=0; i < 2; i++)
{
    t0 = x[i];
    t1 = x[i+1];
    t2 = t0 + t1;
    y[i] = t2;
    t3 = t0 - t1;
    y[i+1] = t3;
}
```

```
def f(x: Array[Rep[Double]],
      y: Array[Rep[Double]]) = {
      for (i <- 0 until 2) {
        y(2*i)   = x(i*2) + x(i*2+1)
        y(2*i+1) = x(i*2) - x(i*2+1)
      }
```


scalarized

```
t0 = s0 + s1;
t1 = s0 - s1;
t2 = s2 + s3;
t2 = s2 - s3;
```

Staging enables program generation

Abstracting over code style =
abstracting over staging decisions

```
def f[L[_],A[_],T](looptype: L, x: A[Array[T]], y: A[Array[T]]) = {
      for (i <- 0 until 2: L[Range]) {
        y(2*i)  = x(i*2) + x(i*2+1)
        y(2*i+1)= x(i*2) - x(i*2+1)
      }
}
```

```
      for (i <- 0 until 2: Rep[Range]) {
        y(2*i)   = x(i*2) + x(i*2+1)
        y(2*i+1) = x(i*2) - x(i*2+1)
      }
    }
```



```
t1 = x[i+1];
t2 = t0 + t1;
y[i] = t2;
t3 = t0 - t1;
y[i+1] = t3;
}
```

DSLs/Program Generation for Performance

Spiral: Linear transforms (2000-2008)

PetaBricks: Polyalgorithmic tuning (2009)

OptiML: Statistical inference (2011)

Liszt: PDE solvers (2011)

Pochoir: Stencils (2011)

Cl1ck/Clak: Linear algebra (2012)

Halide: Image processing (2013)

LGen: Small linear algebra (2014)

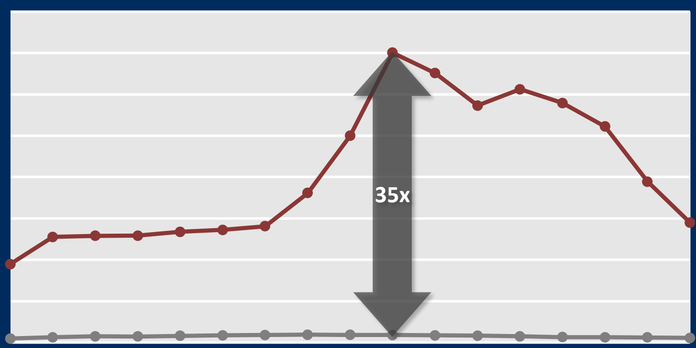
TACO: Tensor algebra (2017)

Lift: Stencils and more (2017)

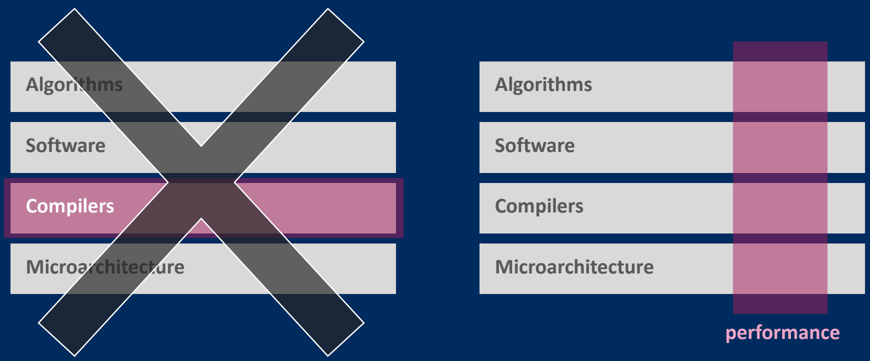
…. many dozens more, active field of research

Advanced Systems Lab

Conclusions



Straightforward implementations often underperform by an order of magnitude, even if single-threaded



Performance is different than other software quality features

Research Questions

How to port performance?

How to automate the production of fastest numerical code?

- *Domain-specific languages*
- *Rewriting*
- *Compilers*
- *Machine Learning*

What program language features help with program generation?

What environment should be used to build generators?

How to represent mathematical functionality?

How to formalize the mapping to fast code?

How to handle various forms of parallelism?

How to integrate into standard work flows?