

Advanced Systems Lab

Spring 2024

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

Instructor: Markus Püschel

TA: Tommaso Pegolotti, several more

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

1

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

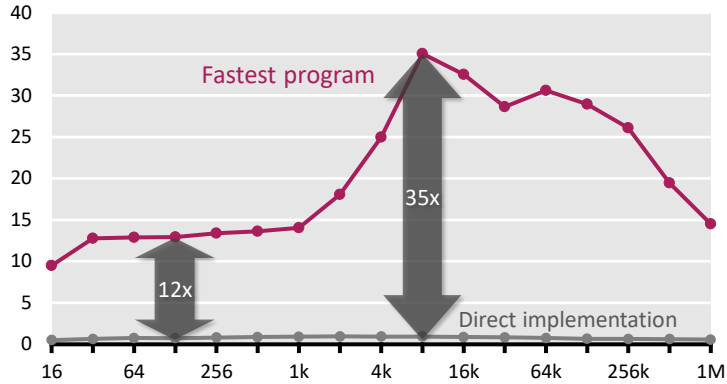
2

2

The Problem: Example DFT

DFT on Intel Core i7 (4 Cores, 2.66 GHz)

Performance [Gflop/s]



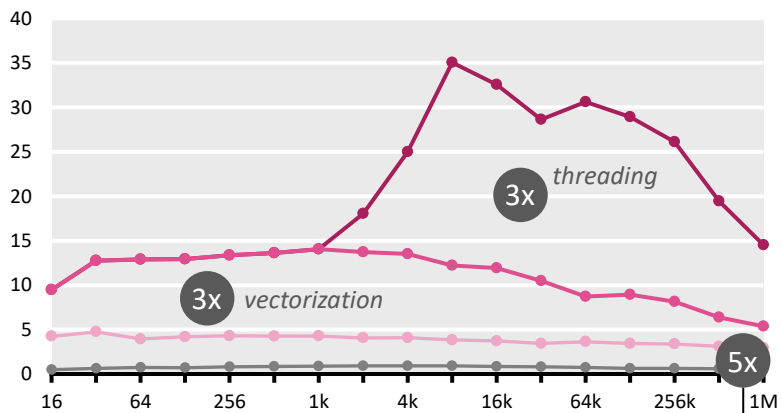
- Same number of operations
- Best compiler

3

DFT: Analysis

DFT (single precision) on Intel Core i7 (4 cores, 2.66 GHz)

Performance [Gflop/s]



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

locality optimization

4

Goal of Spiral:

Computer writes high performance library code

Generate Code



5

Select convolutional code
Select a preset code or customize parameters

- custom
- Voyager
- NASA-DSN
- CCSDS/NASA-GSFC
- WiMax
- CDMA IS-95A
- LTE (3GPP - Long Term Evolution)
- UWB (802.15)
- CDMA 2000
- Cassini
- Mars Pathfinder & Stereo

rate: 1 / 2
K: 7
polynomials: 109
79

code rate (?)
constraint length (?)
polynomials for the code in decimal notation (?)

Select implementation options
frame length: 2048
unpadded frame length:
Vectorization level: scalar C
type of code (?)

Viterbi Decoder

DFT IP Cores

parameter	value	range	explanation
Problem specification			
transform size	64	4-32768	Number of samples (?)
direction	forward		forward or inverse DFT (?)
data type	fixed point		fixed or floating point (?)
	16 bits	4-32 bits	fixed point precision (?)
	unscaled		scaling mode (?)
Parameters controlling implementation			
architecture	fully streaming		iterative or fully streaming (?)
radix	2	2, 4, 8, 16, 32, 64	size of DFT basic block (?)
streaming width	2	2-64	number of complex words per cycle (?)
data ordering	natural in / natural out		natural or digit-reversed data order (?)
BRAM budget	1000		maximum # of BRAMs to utilize (-1 for no limit) (?)

@ www.spiral.net

6

Possible Approach:

Capturing algorithm knowledge:
Domain-specific languages (DSLs)

Structural optimization:
Rewriting systems

High performance code style:
Compiler

Decision making for choices:
Machine learning

7

Organization

Spiral: Basic system

Vectorization

General input size

Results

Final remarks

8

Algorithms: Example FFT, n = 4

Fast Fourier transform (FFT)

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

Representation using matrix algebra

$$\text{DFT}_4 = (\text{DFT}_2 \otimes I_2) T_2^4 (I_2 \otimes \text{DFT}_2) L_2^4$$

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

Divide-and-conquer algorithms = breakdown rules in SPL

9

Decomposition Rules (>200 for >40 Transforms)

$$\begin{aligned} \text{DFT}_n &\rightarrow P_{n/2,2m}^\top (\text{DFT}_{2m} \oplus (I_{k/2-1} \otimes C_{2m} \text{rDFT}_{2m}(i/k))) (\text{RDFT}_k^i \otimes I_m), \quad k \text{ even,} \\ \begin{pmatrix} \text{RDFT}_n \\ \text{RDFT}_n^\top \\ \text{DHT}_n \\ \text{DHT}_n^\top \end{pmatrix} &\rightarrow (P_{k/2,m}^\top \otimes I_2) \begin{pmatrix} \text{RDFT}_{2m} \\ \text{RDFT}_{2m}^\top \\ \text{DHT}_{2m} \\ \text{DHT}_{2m}^\top \end{pmatrix} \oplus \left(I_{k/2-1} \otimes D_{2m} \begin{pmatrix} \text{rDFT}_{2m}(i/k) \\ \text{rDFT}_{2m}^\top(i/k) \\ \text{rDHT}_{2m}(i/k) \\ \text{rDHT}_{2m}^\top(i/k) \end{pmatrix} \right) \begin{pmatrix} \text{RDFT}_k^i \\ \text{RDFT}_k^i \\ \text{DHT}_k^i \\ \text{DHT}_k^i \end{pmatrix} \otimes I_m, \quad k \text{ even,} \\ \begin{pmatrix} \text{rDFT}_{2n}(u) \\ \text{rDHT}_{2n}(u) \end{pmatrix} &\rightarrow L_m^{2n} \left(I_k \otimes \begin{pmatrix} \text{rDFT}_{2m}((i+u)/k) \\ \text{rDHT}_{2m}((i+u)/k) \end{pmatrix} \right) \left(\begin{pmatrix} \text{rDFT}_{2k}(u) \\ \text{rDHT}_{2k}(u) \end{pmatrix} \otimes I_m \right), \\ \text{RDFT-3}_n &\rightarrow (Q_{k/2,m}^\top \otimes I_2) (I_k \otimes \text{rDFT}_{2m}) (i+1/2/k) (\text{RDFT-3}_k \otimes I_m), \quad k \text{ even,} \\ \text{DCT-2}_n &\rightarrow P_{k/2,2m}^\top (\text{DCT-2}_{2m} K_2^m \oplus (I_{k/2-1} \otimes N_{2m} \text{RDFT-3}_{2m}^\top)) B_n (L_{k/2}^{n/2} \otimes I_2) (I_m \otimes \text{RDFT}_k) Q_{m/2,k}, \\ \text{DCT-3}_n &\rightarrow \text{DCT-2}_n, \end{aligned}$$

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

$$\begin{aligned} \text{DFT}_n &\rightarrow P_n (\text{DFT}_n \otimes \text{DFT}_n) Q_n, \quad n = 2^m, \text{gcd}(j, m) = 1 \\ \text{DCT-3}_n &\rightarrow (I_m \oplus J_m) L_m (\text{DCT-3}_n(1/4) \oplus \text{DCT-3}_n(3/4)) \\ &\quad (F_2 \otimes I_m) \begin{bmatrix} I_m & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & 1 \end{bmatrix}, \quad n = 2m \\ \text{DCT-4}_n &\rightarrow S_n \text{DCT-2}_n \text{diag}_{0 \leq k < n} (1/(2 \cos((2k+1)\pi/4n))) \\ \text{IMDCT}_{2m} &\rightarrow (I_m \oplus I_m \oplus I_m \oplus J_m) \left(\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \otimes I_m \right) \oplus \left(\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \otimes I_m \right) J_{2m} \text{DCT-4}_{2m} \\ \text{WHT}_{2^k} &\rightarrow \prod_{i=1}^k (I_{2^{k-i}} \oplus I_{2^{k-i}} \otimes \text{WHT}_{2^{k-i}} \otimes I_{2^{k-i}}), \quad k = k_1 + \dots + k_l \\ \text{DFT}_2 &\rightarrow F_2 \\ \text{DCT-2}_2 &\rightarrow \text{diag}(1, 1/\sqrt{2}) F_2 \\ \text{DCT-4}_2 &\rightarrow J_2 R_{13\pi/8} \end{aligned}$$

Combining these rules yields many algorithms for every given transform

10

SPL to Code

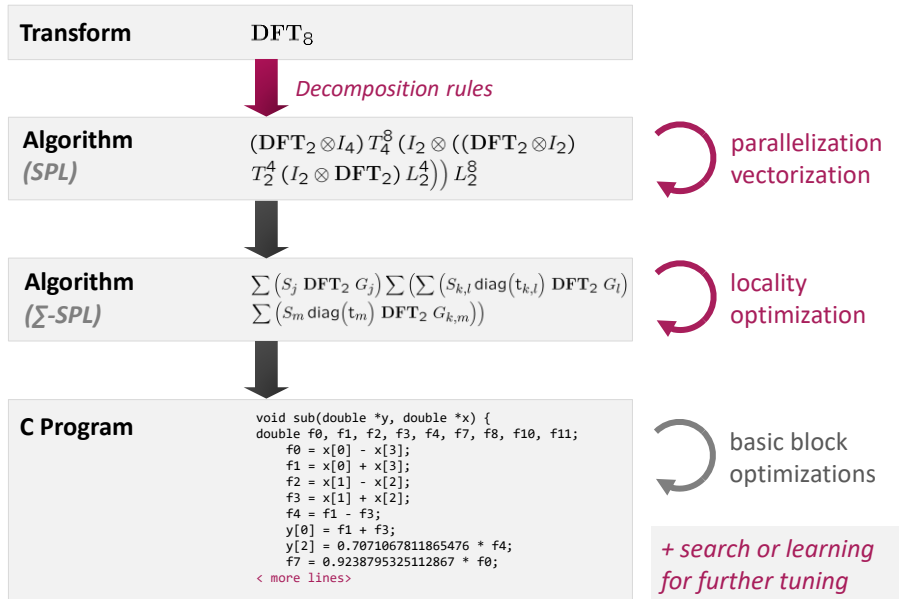
SPL S	Pseudo code for $y = Sx$
$A_n B_n$	<code for: $t = Bx$ > <code for: $y = At$ >
$I_m \otimes A_n$	for (i=0; i<m; i++) <code for: $y[i*n:1:i*n+n-1] = A(x[i*n:1:i*n+n-1])$ >
$A_m \otimes I_n$	for (i=0; i<n; i++) <code for: $y[i:n:i+m*n-n] = A(x[i:n:i+m*n-n])$ >
D_n	for (i=0; i<n; i++) $y[i] = D[i]*x[i]$;
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	$y[0] = x[0] + x[1]$; $y[1] = x[0] - x[1]$;

$$I_m \otimes A_n = \begin{bmatrix} A_n & & \\ & \dots & \\ & & A_n \end{bmatrix}$$

Correct code: easy fast code: very difficult

11

Program Generation in Spiral



12

Organization

Spiral: Basic system

Vectorization

General input size

Results

Final remarks

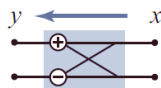
13

Example: Vectorization in Spiral

Goal: Translate SPL expressions directly into SIMD code

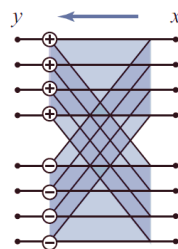
Relationship SPL expressions \leftrightarrow vectorization?

$$y = \text{DFT}_2 x$$



one addition
one subtraction

$$y = (\text{DFT}_2 \otimes \text{I}_4) x$$



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

14

Step 1: Identify “Good” Vector Constructs

Vector length: ν

Good (= easily vectorizable) SPL constructs:

$$A \otimes I_\nu$$

$$L_\nu^{\nu^2}, L_2^{2\nu}, L_\nu^{2\nu} \quad \text{base cases}$$

SPL expressions recursively built from those

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

15

Step 2: Find Manipulation Rules

$$L_n^{n\nu} \rightarrow (I_{n/\nu} \otimes L_\nu^{\nu^2})(L_{n/\nu}^n \otimes I_\nu)$$

$$L_\nu^{n\nu} \rightarrow (L_\nu^n \otimes I_\nu)(I_{n/\nu} \otimes L_\nu^{\nu^2})$$

$$L_m^{mn} \rightarrow (L_m^{mn/\nu} \otimes I_\nu)(I_{mn/\nu^2} \otimes L_\nu^{\nu^2})(I_{n/\nu} \otimes L_{m/\nu}^m \otimes I_\nu)$$

$$I_l \otimes L_n^{kmn} \otimes I_r \rightarrow (I_l \otimes L_n^{kn} \otimes I_{mr})(I_{kl} \otimes L_n^{mn} \otimes I_r)$$

$$I_l \otimes L_n^{kmn} \otimes I_r \rightarrow (I_l \otimes L_{kn}^{kmn} \otimes I_r)(I_l \otimes L_{mn}^{kmn} \otimes I_r)$$

$$I_l \otimes L_{kn}^{kmn} \otimes I_r \rightarrow (I_{lj} \otimes L_m^{mn} \otimes I_r)(I_l \otimes L_k^{kn} \otimes I_{mr})$$

$$I_l \otimes L_{kn}^{kmn} \otimes I_r \rightarrow (I_l \otimes L_k^{kmn} \otimes I_r)(I_l \otimes L_{mn}^{kmn} \otimes I_r)$$

Manipulation rules = SIMD knowledge in Spiral_{mn/\nu}

$$(I_m \otimes A^{n \times n}) L_m^{mn} \rightarrow (I_{m/\nu} \otimes L_\nu^{m\nu} (A^{n \times n} \otimes I_\nu))(L_{m/\nu}^{mn/\nu} \otimes I_\nu)$$

$$L_n^{mn} (I_m \otimes A^{n \times n}) \rightarrow (L_n^{mn/\nu} \otimes I_\nu)(I_{m/\nu} \otimes (A^{n \times n} \otimes I_\nu) L_n^{\nu})$$

$$(I_k \otimes (I_m \otimes A^{n \times n}) L_m^{mn}) L_k^{kmn} \rightarrow (L_k^{km} \otimes I_n)(I_m \otimes (I_k \otimes A^{n \times n}) L_k^{kn})(L_m^{mn} \otimes I_k)$$

$$L_{mn}^{kmn} (I_k \otimes L_n^{mn} (I_m \otimes A^{n \times n})) \rightarrow (L_{mn}^{mn} \otimes I_k)(I_m \otimes L_n^{kn} (I_k \otimes A^{n \times n}))(L_m^{km} \otimes I_n)$$

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

$$\overline{A^{m \times m} \otimes I_\nu} \rightarrow (I_m \otimes L_\nu^{2\nu})(\overline{A^{m \times m}} \otimes I_\nu)(I_m \otimes L_2^{2\nu})$$

$$\overline{I_m \otimes A^{n \times n}} \rightarrow I_m \otimes A^{n \times n}$$

$$\overline{\overline{D}} \rightarrow (I_{n/\nu} \otimes L_\nu^{2\nu}) \overline{\overline{D}} (I_{n/\nu} \otimes L_2^{2\nu})$$

$$\overline{\overline{P}} \rightarrow P \otimes I_2$$

16

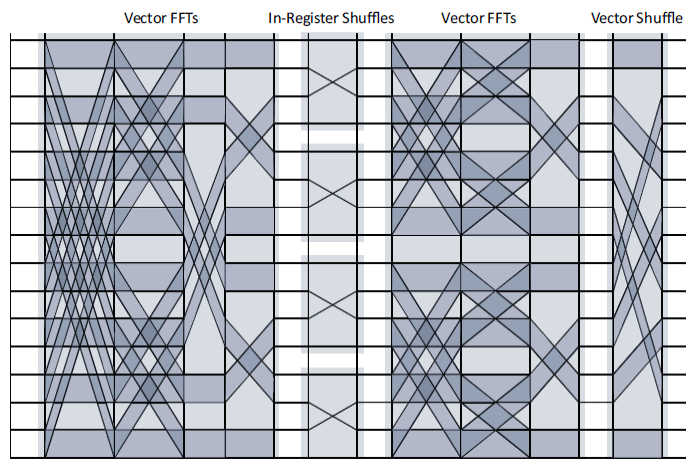
Example

$$\begin{aligned} \underbrace{\text{DFT}_{mn}}_{\text{vec}(\nu)} &\rightarrow \underbrace{(\text{DFT}_m \otimes \mathbf{I}_n) \overline{\mathbf{T}}_n^{mn} (\mathbf{I}_m \otimes \text{DFT}_n) \mathbf{L}_m^{mn}}_{\text{vec}(\nu)} \\ &\dots \\ &\dots \\ &\rightarrow \underbrace{\left(\mathbf{I}_{\frac{mn}{\nu}} \otimes \mathbf{L}_{\nu}^{2\nu} \right)}_{\text{vectorized data accesses}} \underbrace{\left(\overline{\text{DFT}}_m \otimes \mathbf{I}_{\frac{n}{\nu}} \otimes \mathbf{I}_{\nu} \right)}_{\text{vectorized arithmetic}} \overline{\mathbf{T}}_n^{mn} \\ &\quad \underbrace{\left(\mathbf{I}_{\frac{m}{\nu}} \otimes (\mathbf{L}_{\nu}^{2n} \otimes \mathbf{I}_{\nu}) \right)}_{\text{vectorized data accesses}} \underbrace{\left(\mathbf{I}_{\frac{n}{\nu}} \otimes \mathbf{L}_{\nu}^{2\nu} \otimes \mathbf{I}_{\nu} \right)}_{\text{vectorized arithmetic}} \underbrace{\left(\overline{\text{DFT}}_n \otimes \mathbf{I}_{\nu} \right)}_{\text{vectorized arithmetic}} \underbrace{\left(\mathbf{L}_{\frac{m}{\nu}}^{\frac{mn}{\nu}} \otimes \mathbf{L}_{\nu}^{2\nu} \right)}_{\text{vectorized data accesses}} \end{aligned}$$

vectorized arithmetic
vectorized data accesses

17

Sketch for complex $\nu = 2$



$$\left(\left((\text{DFT}_2 \otimes I_2) T_2^4 (I_2 \otimes \text{DFT}_2) L_2^4 \otimes I_2 \otimes I_2 \right) T_4^{16} \left(I_2 \otimes (L_2^4 \otimes I_2) (I_2 \otimes L_2^4) ((\text{DFT}_2 \otimes I_2) T_2^4 (I_2 \otimes \text{DFT}_2) L_2^4) \otimes I_2 \right) (L_2^8 \otimes I_2) \right)$$

18

18

Automatically Generate Base Case Library

Goal: Given instruction set, generate base cases

$$\nu = 4 : \{ L_2^4, I_2 \otimes L_2^4, L_2^4 \otimes I_2, L_2^8, L_4^8 \}$$

Idea: Instructions as matrices + search

`y = _mm_unpacklo_ps(x0, x1);`

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

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

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

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

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

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

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



No base case

19

Automatically Generate Base Case Library

Goal: Given instruction set, generate base cases

$$\nu = 4 : \{ L_2^4, I_2 \otimes L_2^4, L_2^4 \otimes I_2, L_2^8, L_4^8 \}$$

Idea: Instructions as matrices + search

`y = _mm_unpacklo_ps(x0, x1);`

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

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

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

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

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

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

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



$L_2^4 \otimes I_2$
Base case

20

Same Approach for Different Paradigms

Threading:

$$\begin{aligned} \underbrace{\text{DFT}_{mn}}_{\text{smp}(\mu,\mu)} &\rightarrow \underbrace{(\text{DFT}_m \otimes I_n) \mathbf{T}_n^{mn} (I_m \otimes \text{DFT}_n)}_{\text{smp}(\mu,\mu)} \mathbf{L}_m^{mn} \\ &\dots \\ &\rightarrow \underbrace{(\text{DFT}_m \otimes I_n)}_{\text{smp}(\mu,\mu)} \underbrace{\mathbf{T}_n^{mn}}_{\text{smp}(\mu,\mu)} \underbrace{(I_m \otimes \text{DFT}_n)}_{\text{smp}(\mu,\mu)} \underbrace{\mathbf{L}_m^{mn}}_{\text{smp}(\mu,\mu)} \\ &\dots \\ &\rightarrow ((I_m^{mp} \otimes I_{n/p}) \otimes_{\mu} I_{\mu}) (I_p \otimes (\text{DFT}_m \otimes I_{n/p})) ((L_p^{mp} \otimes I_{n/p}) \otimes_{\mu} I_{\mu}) \\ &\quad \left(\bigotimes_{n=0}^{p-1} \mathbf{T}_n^{mn} \right) (I_p \otimes (I_{n/p} \otimes \text{DFT}_n)) (I_p \otimes I_{n/p}^{mn/p}) ((L_p^{mp} \otimes I_{n/p}) \otimes_{\mu} I_{\mu}) \end{aligned}$$

Vectorization:

$$\begin{aligned} \underbrace{(\text{DFT}_{mn})}_{\text{vec}(v)} &\rightarrow \underbrace{(\text{DFT}_m \otimes I_n) \mathbf{T}_n^{mn} (I_m \otimes \text{DFT}_n)}_{\text{vec}(v)} \mathbf{L}_m^{mn} \\ &\dots \\ &\rightarrow \underbrace{(\text{DFT}_m \otimes I_n)}_{\text{vec}(v)} \underbrace{(\mathbf{T}_n^{mn})^v}_{\text{vec}(v)} \underbrace{(I_m \otimes \text{DFT}_n)}_{\text{vec}(v)} \underbrace{\mathbf{L}_m^{mn}}_{\text{vec}(v)} \\ &\dots \\ &\rightarrow (I_{mn/\nu} \otimes \underbrace{L_p^{2\nu}}_{\text{sse}}) (\text{DFT}_m \otimes \underbrace{I_{n/p}}_{\text{sse}} \otimes I_{\nu}) (\mathbf{T}_n^{mn})^v \\ &\quad (I_{n/\nu} \otimes \underbrace{L_p^{2\nu}}_{\text{sse}} \otimes I_{\nu}) (I_{n/\nu} \otimes \underbrace{L_p^{2\nu}}_{\text{sse}} \otimes I_{\nu}) (I_2 \otimes \underbrace{L_p^{2\nu}}_{\text{sse}}) (L_2^{2\nu} \otimes I_{\nu}) (\text{DFT}_n \otimes I_{\nu}) \\ &\quad ((L_m^{mn} \otimes I_2) \otimes I_{\nu}) (I_{mn/\nu} \otimes \underbrace{L_p^{2\nu}}_{\text{sse}}) \end{aligned}$$

GPUs:

$$\begin{aligned} \underbrace{(\text{DFT}_{r,k})}_{\text{gpu}(r,c)} &\rightarrow \underbrace{\left(\prod_{i=0}^{k-1} L_r^{r,k} (I_{k-1} \otimes \text{DFT}_r) (L_{r-k-i-1}^{r,k} \otimes \mathbf{T}_r^{r-k-i-1}) L_{r+i-1}^{r,k} \right)}_{\text{gpu}(r,c)} \mathbf{R}_r^{r,k} \\ &\dots \\ &\rightarrow \left(\prod_{i=0}^{k-1} (L_r^{r,2} \otimes I_2) (I_{r-1/2} \otimes \times \underbrace{(\text{DFT}_r \otimes I_2)}_{\text{shd}(r,c)} L_r^{2r}) \mathbf{T}_i \right) \\ &\quad (L_r^{r/2} \otimes I_2) (I_{r-1/2} \otimes \times \underbrace{L_r^{2r}}_{\text{shd}(r,c)}) (\mathbf{R}_r^{r-1} \otimes I_r) \end{aligned}$$

Verilog for FPGAs:

$$\begin{aligned} \underbrace{(\text{DFT}_{r,k})}_{\text{stream}(r^*)} &\rightarrow \left[\prod_{i=0}^{k-1} L_r^{r,k} (I_{k-1} \otimes \text{DFT}_r) (L_{r-k-i-1}^{r,k} \otimes \mathbf{T}_r^{r-k-i-1}) L_{r+i-1}^{r,k} \right] \mathbf{R}_r^{r,k} \\ &\dots \\ &\rightarrow \left[\prod_{i=0}^{k-1} \underbrace{L_r^{r,k}}_{\text{stream}(r^*)} \underbrace{(I_{k-1} \otimes \text{DFT}_r)}_{\text{stream}(r^*)} \underbrace{(L_{r-k-i-1}^{r,k} \otimes \mathbf{T}_r^{r-k-i-1}) L_{r+i-1}^{r,k}}_{\text{stream}(r^*)} \right] \mathbf{R}_r^{r,k} \\ &\dots \\ &\rightarrow \left[\prod_{i=0}^{k-1} \underbrace{L_r^{r,k}}_{\text{stream}(r^*)} (I_{k-1} \otimes (I_{r-1} \otimes \text{DFT}_r)) \underbrace{\mathbf{T}_r^i}_{\text{stream}(r^*)} \right] \mathbf{R}_r^{r,k} \\ &\quad \underbrace{\mathbf{R}_r^{r,k}}_{\text{stream}(r^*)} \end{aligned}$$

- Rigorous, correct by construction
- Overcomes compiler limitations

21

Organization

Spiral: Basic system

Vectorization

General input size

Results

Final remarks

22

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)
  }
  ...
}
```

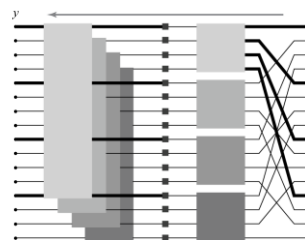
- Algorithm cannot be fixed
- Recursive code
- Creates many challenges

23

Challenge: Recursion Steps

Cooley-Tukey FFT

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



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);
}
```

24

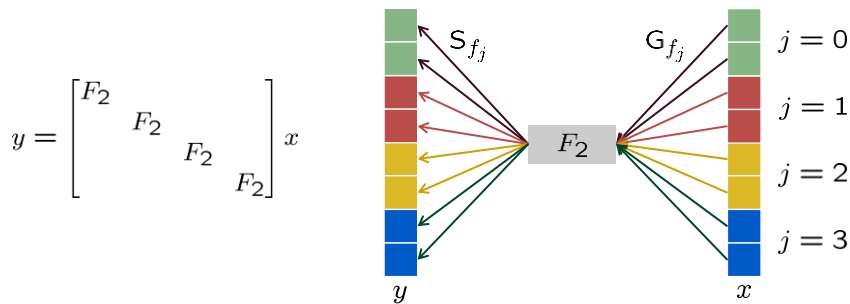
Σ -SPL : Basic Idea

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: $y = (I_4 \otimes F_2)x \rightarrow y = \sum_{j=0}^3 S_{f_j} F_2 G_{f_j} x$



25

Find Recursion Step Closure

Voronenko, 2008

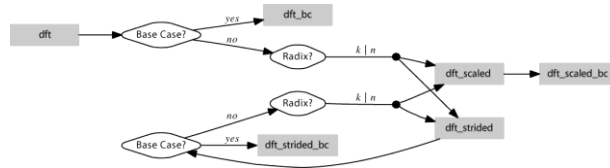
$$\begin{aligned}
 & \{DFT_n\} \\
 & \downarrow \\
 & ((DFT_{n/k} \otimes I_k) T_k^n (I_{n/k} \otimes DFT_k)) L_{n/k}^{n/k} \\
 & \downarrow \\
 & \left(\sum_{i=0}^{k-1} S_{h_{i,k}} \{DFT_{n/k}\} G_{h_{i,k}} \right) \text{diag}(f) \left(\sum_{j=0}^{n/k-1} S_{h_{j,k,1}} \{DFT_k\} G_{h_{j,k,1}} \right) \text{perm}(L_{n/k}^n) \\
 & \downarrow \\
 & \sum_{i=0}^{k-1} S_{h_{i,k}} \{DFT_{n/k}\} \text{diag}(f \circ h_{i,k}) G_{h_{i,k}} \quad \sum_{j=0}^{n/k-1} S_{h_{j,k,1}} \{DFT_k\} G_{h_{j,n/k}} \\
 & \downarrow \\
 & \sum_{i=0}^{k-1} \{ S_{h_{i,k}} DFT_{n/k} \text{diag}(f \circ h_{i,k}) G_{h_{i,k}} \} \quad \sum_{j=0}^{n/k-1} \{ S_{h_{j,k,1}} DFT_k G_{h_{j,n/k}} \}
 \end{aligned}$$

Repeat until closure

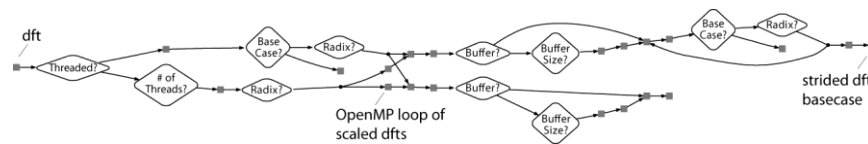
26

Recursion Step Closure: Examples

DFT: scalar code (like FFTW 2.x)



DFT: full-fledged (vectorized and parallel code)



27

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

28

Organization

Spiral: Basic system

Vectorization

General input size

Results

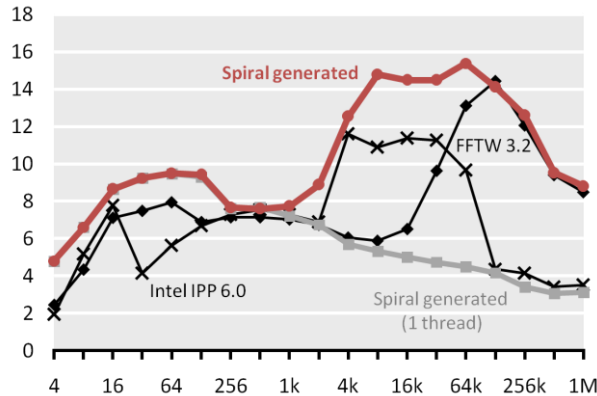
Final remarks

29


DFT on Intel Multicore

Complex DFT (Intel Core i7, 2.66 GHz, 4 cores)

Performance [Gflop/s] vs. input size



$$\begin{aligned}
 \text{DFT}_n &\rightarrow (\text{DFT}_k \otimes I_m) T_m^T(I_k \otimes \text{DFT}_m) L_k^T \\
 \text{DFT}_n &\rightarrow P_{k/2,2m}^T (\text{DFT}_{2m} \oplus (I_{k/2-1} \otimes C_{2m} \text{rDFT}_{2m}(i/k))) (\text{RDFT}_k \otimes I_m) \\
 \text{RDFT}_n &\rightarrow (P_{k/2,2m}^T \otimes I_2) (\text{RDFT}_{2m} \oplus (I_{k/2-1} \otimes D_{2m} \text{rDFT}_{2m}(i/k))) (\text{RDFT}_k \otimes I_m) \\
 \text{rDFT}_{2n}(u) &\rightarrow L_{2m}^{2n}(I_k \otimes_i \text{rDFT}_{2m}((i+u)/k)) (\text{rDFT}_{2k}(u) \otimes I_m)
 \end{aligned}$$


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

30

Generating 100s of FFTWs

PhD thesis Voronenko, 2009

$$\begin{aligned}
 & \text{DFT}_n \rightarrow P_{k/2,2m}^\top \left(\text{DFT}_{2m} \oplus (I_{k/2-1} \otimes C_{2m} \text{rDFT}_{2m}(i/k)) \right) \left(\text{RDFT}'_k \otimes I_m \right), \quad k \text{ even,} \\
 & \begin{pmatrix} \text{RDFT}'_n \\ \text{RDFT}'_n \\ \text{DHT}'_n \\ \text{DHT}'_n \end{pmatrix} \rightarrow (P_{k/2,m}^\top \otimes I_2) \left(\begin{pmatrix} \text{RDFT}'_{2m} \\ \text{RDFT}'_{2m} \\ \text{DHT}'_{2m} \\ \text{DHT}'_{2m} \end{pmatrix} \oplus \left(I_{k/2-1} \otimes D_{2m} \begin{pmatrix} \text{rDFT}_{2m}(i/k) \\ \text{rDFT}_{2m}(i/k) \\ \text{rDHT}_{2m}(i/k) \\ \text{rDHT}_{2m}(i/k) \end{pmatrix} \right) \right) \begin{pmatrix} \text{RDFT}'_k \\ \text{RDFT}'_k \\ \text{DHT}'_k \\ \text{DHT}'_k \end{pmatrix} \otimes I_m, \quad k \text{ even,} \\
 & \begin{pmatrix} \text{rDFT}_{2n}(u) \\ \text{rDHT}_{2n}(u) \end{pmatrix} \rightarrow L_{2m}^{2n} \left(I_k \otimes \begin{pmatrix} \text{rDFT}_{2m}((i+u)/k) \\ \text{rDHT}_{2m}((i+u)/k) \end{pmatrix} \right) \left(\begin{pmatrix} \text{rDFT}_{2k}(u) \\ \text{rDHT}_{2k}(u) \end{pmatrix} \otimes I_m \right), \\
 & \text{RDFT-3}_n \rightarrow (Q_{k/2,m}^\top \otimes I_2) (I_k \otimes \text{rDFT}_{2m}(i+1/2/k)) (\text{RDFT-3}_k \otimes I_m), \quad k \text{ even,} \\
 & \text{DCT-2}_n \rightarrow P_{k/2,2m}^\top \left(\text{DCT-2}_{2m} K_2^{2m} \oplus (I_{k/2-1} \otimes N_{2m} \text{RDFT-3}_{2m}^\top) \right) B_n(L_{k/2}^{n/2} \otimes I_2) (I_m \otimes \text{RDFT}'_k) Q_{m/2,k}, \\
 & \text{DCT-3}_n \rightarrow \text{DCT-2}_n^\top, \\
 & \text{DCT-4}_n \rightarrow Q_{k/2,2m}^\top (I_{k/2} \otimes N_{2m} \text{RDFT-3}_{2m}^\top) B'_n(L_{k/2}^{n/2} \otimes I_2) (I_m \otimes \text{RDFT-3}_k) Q_{m/2,k}. \\
 & \text{DFT}_n \rightarrow (\text{DFT}'_k \otimes I_m) T'_m(I_k \otimes \text{DFT}_m) L'_k, \quad n = km \\
 & \text{DFT}_n \rightarrow P_n(\text{DFT}'_k \otimes \text{DFT}_m) Q_n, \quad n = km, \text{gcd}(k, m) = 1 \\
 & \text{DFT}_p \rightarrow R_p^t (I_1 \oplus \text{DFT}_{p-1}) D_p (I_1 \oplus \text{DFT}_{p-1}) R_p, \quad p \text{ prime} \\
 & \text{DCT-3}_n \rightarrow (I_m \oplus J_m) L'_m(\text{DCT-3}_m(1/4) \oplus \text{DCT-3}_m(3/4)) \\
 & \quad \cdot (F_2 \otimes I_m) \begin{bmatrix} I_m & 0 & -J_{m-1} \\ & \frac{1}{\sqrt{2}}(I_1 \oplus 2I_m) & \end{bmatrix}, \quad n = 2m \\
 & \text{DCT-4}_n \rightarrow S_n \text{DCT-2}_n \text{diag}_{0 \leq k < n} (1/(2 \cos((2k+1)\pi/4n))) \\
 & \text{IMDCT}_{2m} \rightarrow (J_m \oplus I_m \oplus I_m \oplus J_m) \left(\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} \otimes I_m \right) \oplus \left(\begin{bmatrix} -1 \\ 1 \end{bmatrix} \otimes I_m \right) \right) J_{2m} \text{DCT-4}_{2m} \\
 & \text{WHT}_{2^k} \rightarrow \prod_{i=1}^k (I_{2^{k_1+\dots+k_{i-1}}} \otimes \text{WHT}_{2^{k_i}} \otimes I_{2^{k_1+\dots+k_i}}), \quad k = k_1 + \dots + k_t \\
 & \text{DFT}_2 \rightarrow F_2 \\
 & \text{DCT-2}_2 \rightarrow \text{diag}(1, 1/\sqrt{2}) F_2 \\
 & \text{DCT-4}_2 \rightarrow J_2 R_{13\pi/8}
 \end{aligned}$$

31

Generating 100s of FFTWs

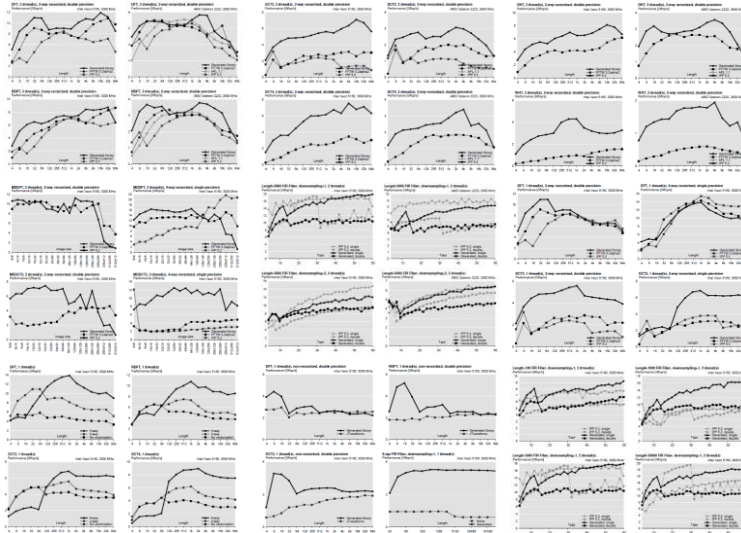
PhD thesis Voronenko, 2009

Transform	Code size	
	non-parallelized	parallelized
<i>no vectorization</i>		
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	—
<i>2-way vectorization</i>		
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
<i>4-way vectorization</i>		
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

32

Generating 100s of FFTWs

PhD thesis Voronenko, 2009



33

Computer generated Functions for Intel IPP 6.0



3984 C functions
1M lines of code

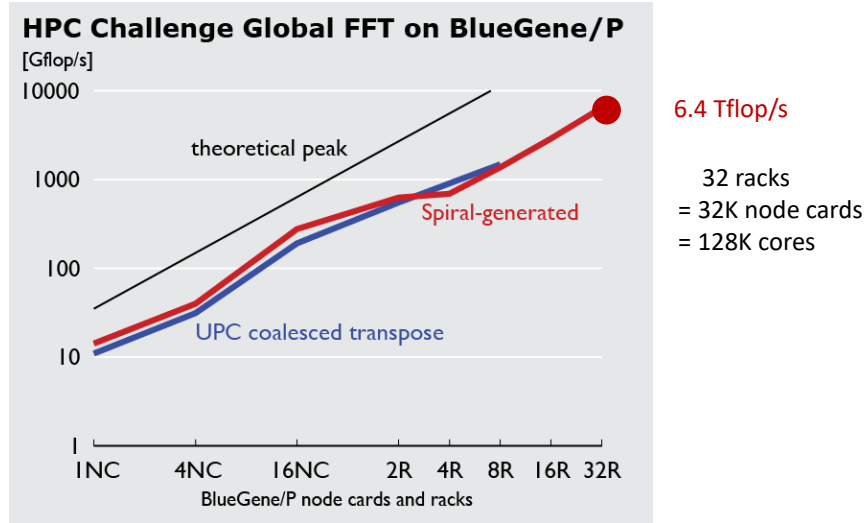
Transforms: DFT (fwd+inv), RDFT (fwd+inv), DCT2, DCT3, DCT4, DHT, WHT
 Sizes: 2-64 (DFT, RDFT, DHT); 2-powers (DCTs, WHT)
 Precision: single, double
 Data type: scalar, SSE, AVX (DFT, DCT), LRB (DFT)

Computer generated

Results: SpiralGen Inc.

34

Very Large Scale: BG/P



2010 HPC Challenge Class I Award, Almasi et al.

35

Organization

Spiral: Basic system

Vectorization

General input size

Results

Final remarks

36

Spiral: Summary

Spiral:

Successful approach to automating the development of computing software

Commercial proof-of-concept



DFT₆₄



```
void dft64(float *v, float *X) {
    __m512 U912, U913, U914, U915, ...
    __m512 *a2153, *a2155;
    a2153 = ((__m512 *) X); a1107 = *(a2153);
    a1108 = *((a2153 + 4)); t1323 = __mm512_add_ps(a1107, a1108);
    t1324 = __mm512_sub_ps(a1107, a1108);
    <many more lines>
    U926 = __mm512_qltupsoovv_s32(...);
    a1121 = __mm512_madd231_ps(__mm512_mul_ps(__mm512_mak_dc_pi(
        __mm512_mak_tval6_ps(0.70710678118654757), 0x0000, a2154, U926), t1341),
        __mm512_mak_sub_ps(__mm512_swt_tval6_ps(0.70710678118654757), ...),
        __mm512_qltupsoovv_s32(t1341), __mm512_R00_CDAB);
    U927 = __mm512_qltupsoovv_s32
    <many more lines>
}
```

Key ideas:

Algorithm knowledge:
Domain specific symbolic representation

Platform knowledge:
Tagged rewrite rules, SIMD specification

$$DFT_4 \rightarrow (DFT_2 \otimes I_2) T_2^4 (I_2 \otimes DFT_2) L_2^4$$

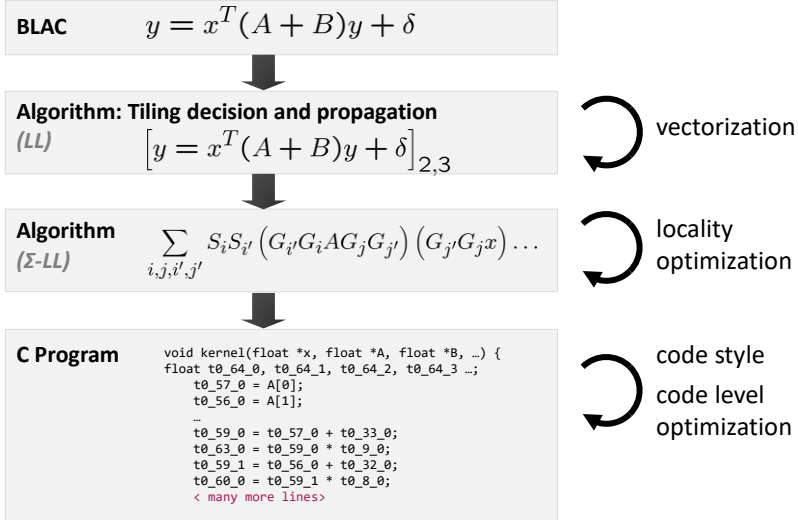
$$\underbrace{I_m \otimes A_n}_{\text{simd}(p, \mu)} \rightarrow I_p \otimes \left(\left(I_{m/p} \otimes A_n \right) \right)$$

Glimpse of other topics ...



LGen: Generator for Basic Linear Algebra

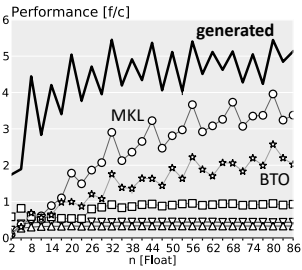
Spampinato & P, CGO 2014



39

LGen: Sample Results

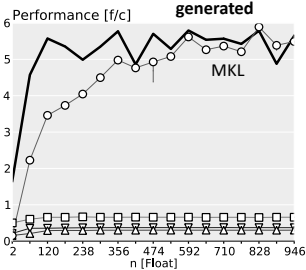
$$C = \alpha AB + \beta C$$



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

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

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



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

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

- LGen
- ▽ Handwritten fixed size
- △ Handwritten gen size
- MKL 11.0
- Eigen 3.1.3
- ★ BTO 1.3
- ◇ IPP 7.1

40

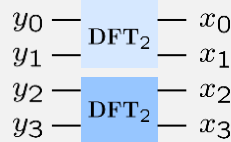
PL Support: Example Code Style

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



SPL $y = (I_2 \otimes \text{DFT}_2)x$

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)
  }
}
```

41

```
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;
}
```

42

```
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)
  }
}
```

```
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)
  }
}
```

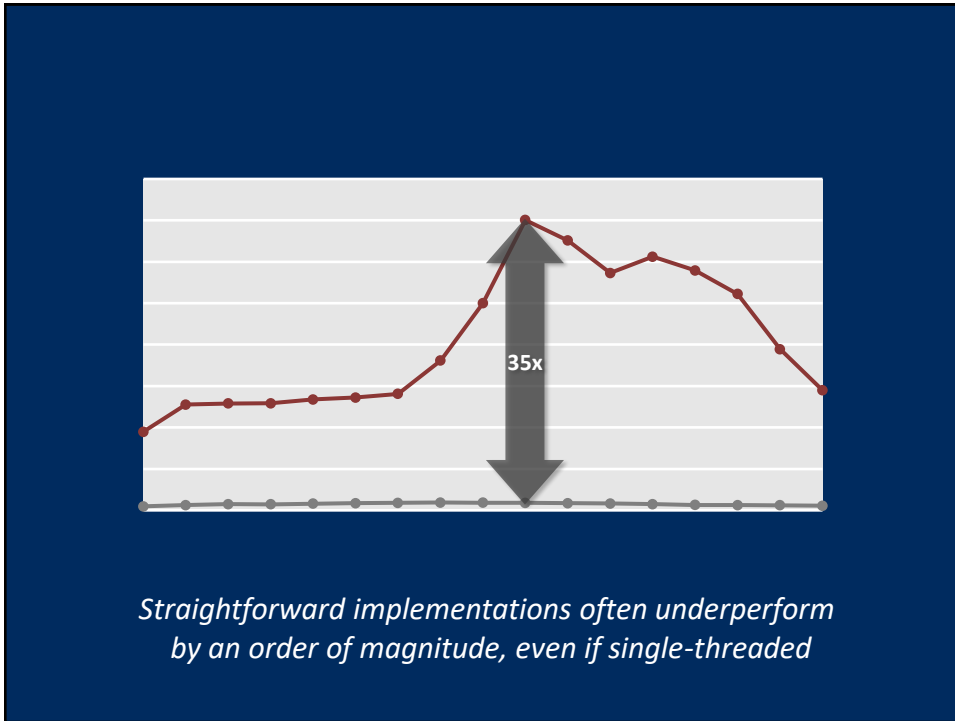
```
t0 = x[i];
t1 = x[i+1];
t2 = t0 + t1;
y[i] = t2;
t3 = t0 - t1;
y[i+1] = t3;
```

43

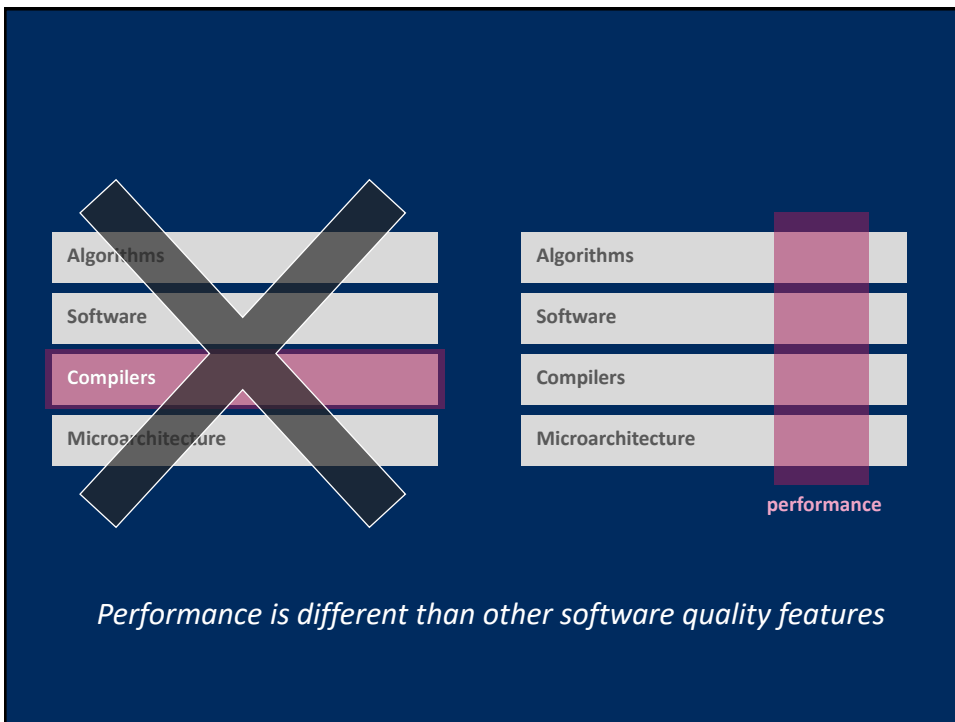
Advanced Systems Lab

Conclusions

44



45



46

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 for performance?

What environment should be used to build such generators?

How to formalize the mapping to fast code?

How to handle various forms of parallelism?

How to integrate into standard work flows?