

How to Write Fast Numerical Code

Spring 2016

Lecture: Computer generation of fast code (Spiral)

Instructor: Markus Püschel

TA: Gagandeep Singh, Daniele Spampinato, Alen Stojanov

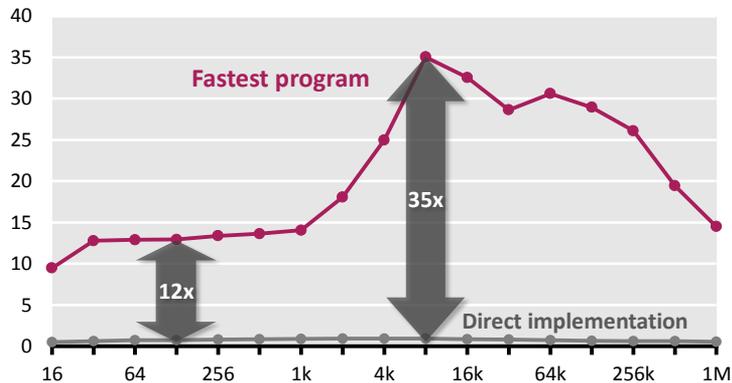
ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

The Problem: Example DFT

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

Performance [Gflop/s]

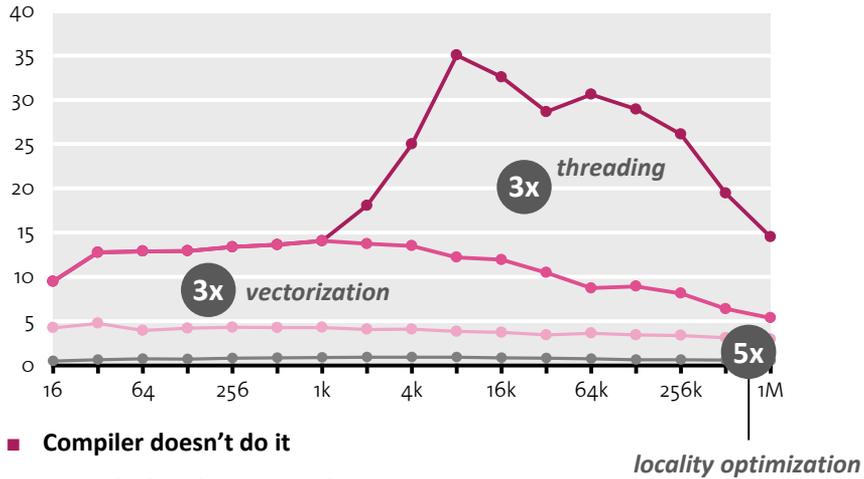


- Same number of operations
- Best compiler

DFT: Analysis

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

Performance [Gflop/s]



Our Goal:

Computer writes high performance library code

Generate Code



"click"

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 / code rate [\(?\)](#)

K constraint length [\(?\)](#)

polynomials polynomials for the code in decimal notation [\(?\)](#)

Select implementation options

frame length unpadded frame length

Vectorization level type of code [\(?\)](#)

DFT IP Cores

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

Viterbi Decoder

[@ www.spiral.net](http://www.spiral.net)

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

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

- *SPL (Signal processing language):* Mathematical, declarative, point-free
- Divide-and-conquer algorithms = breakdown rules in SPL

Decomposition Rules (>200 for >40 Transforms)

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

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

$$\begin{aligned}
 \text{DFT}_n &\rightarrow B_n (\text{DFT}_{n/2} \otimes \text{DFT}_{n/2}) Q_n, \quad n = 4m, \text{gcd}(4,m) = 1 \\
 \text{DCT-3}_n &\rightarrow (I_m \oplus J_m) L_m (\text{DCT-3}_{n/2} \otimes I_2) = \text{DCT-3}_{n/2} (3/4) \\
 &\quad (F_2 \otimes I_m) \begin{bmatrix} I_m & \oplus \oplus \oplus \oplus \\ \sqrt{2}(I_1 \oplus 2I_m) \end{bmatrix}, \quad n = 2m \\
 \text{DCT-4}_n &\rightarrow S_n \text{DCT-2}_n \text{diag}_{0 \leq k < n} (1/(2 \cos((2k+1)\pi/4n))) \\
 \text{IMDCT}_{2m} &\rightarrow (I_m \oplus I_m \oplus I_m \oplus J_m) \left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} \otimes I_m \right) \oplus \left(\begin{bmatrix} -1 \\ 1 \end{bmatrix} \otimes I_m \right) J_{2m} \text{DCT-4}_{2m} \\
 \text{WHT}_{2k} &\rightarrow \prod_{i=1}^k (I_{2^{k_1+\dots+k_{i-1}}} \otimes \text{WHT}_{2^{k_i}} \otimes I_{2^{k_{i+1}+\dots+k_i}}), \quad k = k_1 + \dots + k_l \\
 \text{DFT}_2 &\rightarrow F_2 \\
 \text{DCT-2}_2 &\rightarrow \text{diag}(1, 1/\sqrt{2}) F_2 \\
 \text{DCT-4}_2 &\rightarrow J_2 R_{13\pi/8}
 \end{aligned}$$

Combining these rules yields many algorithms for every given transform

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:
 $y[i*n:i*n+n-1] = A(x[i*n:i*n+n-1])$ >

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

$A_m \otimes I_n$ 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 $y[0] = x[0] + x[1];$
 $y[1] = x[0] - x[1];$

Correct code: easy fast code: very difficult

Program Generation in Spiral

Transform

DFT_8

Decomposition rules

Algorithm
(SPL)

$$(DFT_2 \otimes I_4) T_4^8 (I_2 \otimes ((DFT_2 \otimes I_2) T_2^4 (I_2 \otimes DFT_2) L_2^4)) L_2^8$$

 parallelization
vectorization

Algorithm
(Σ -SPL)

$$\sum (S_j DFT_2 G_j) \sum (\sum (S_{k,l} \text{diag}(t_{k,l}) DFT_2 G_l) \sum (S_m \text{diag}(t_m) DFT_2 G_{k,m}))$$

 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*

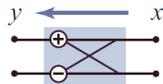
Organization

- Spiral: Basic system
- *Vectorization*
- General input size
- Results
- Final remarks

Example: Vectorization in Spiral

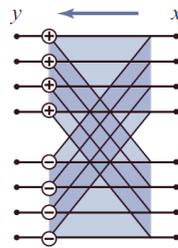
- Relationship SPL expressions \leftrightarrow vectorization?

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



one addition
one subtraction

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



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

Step 1: Identify “Good” Vector Constructs

- Vector length: ν
- Good (= easily vectorizable) SPL constructs:

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

$$\text{L}_\nu^{\nu^2}, \text{L}_2^{2\nu}, \text{L}_\nu^{2\nu} \quad \textit{base cases}$$

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

$$\begin{aligned}
 L_n^{n\nu} &\rightarrow (I_{n/\nu} \otimes L_\nu^{2\nu}) (L_{n/\nu}^n \otimes I_\nu) \\
 L_\nu^{n\nu} &\rightarrow (L_\nu^n \otimes I_\nu) (I_{n/\nu} \otimes L_\nu^{2\nu}) \\
 L_m^{mn} &\rightarrow (L_m^{mn/\nu} \otimes I_\nu) (I_{mn/\nu^2} \otimes L_\nu^{2\nu}) (I_{n/\nu} \otimes L_{m/\nu}^m \otimes I_\nu) \\
 I_l \otimes L_n^{kmn} \otimes I_r &\rightarrow (I_l \otimes L_n^{kn} \otimes I_{mr}) (I_{kl} \otimes L_n^{mn} \otimes I_r) \\
 I_l \otimes L_n^{kmn} \otimes I_r &\rightarrow (I_l \otimes L_{kn}^{kmn} \otimes I_r) (I_l \otimes L_{mn}^{kmn} \otimes I_r) \\
 I_l \otimes L_{km}^{kmn} \otimes I_r &\rightarrow (I_{kl} \otimes L_{mn}^{kmn} \otimes I_r) (I_l \otimes L_k^{kmn} \otimes I_{mr})
 \end{aligned}$$

Manipulation rules = Processor knowledge in Spiral

$$\begin{aligned}
 (I_m \otimes A^{n \times n}) L_m^{mn} &\rightarrow (I_{m/\nu} \otimes L_\nu^{2\nu} (A^{n \times n} \otimes I_\nu)) (L_{m/\nu}^{mn/\nu} \otimes I_\nu) \\
 L_n^{mn} (I_m \otimes A^{n \times n}) &\rightarrow (L_n^{mn/\nu} \otimes I_\nu) (I_{m/\nu} \otimes (A^{n \times n} \otimes I_\nu) L_n^{n\nu}) \\
 (I_k \otimes (I_m \otimes A^{n \times n}) L_m^{mn}) L_k^{kmn} &\rightarrow (L_k^{kn} \otimes I_n) (I_m \otimes (I_k \otimes A^{n \times n}) L_k^{kn}) (L_m^{mn} \otimes I_k) \\
 L_{mn}^{kmn} (I_k \otimes L_n^{mn} (I_m \otimes A^{n \times n})) &\rightarrow (L_{mn}^{mn} \otimes I_k) (I_m \otimes L_n^{kn} (I_k \otimes A^{n \times n})) (L_m^{km} \otimes I_n) \\
 \overline{AB} &\rightarrow \overline{A} \overline{B} \\
 \overline{A^{m \times m} \otimes I_\nu} &\rightarrow (I_m \otimes L_\nu^{2\nu}) (\overline{A^{m \times m}} \otimes I_\nu) (I_m \otimes L_\nu^{2\nu}) \\
 \overline{I_m \otimes A^{n \times n}} &\rightarrow I_m \otimes \overline{A^{n \times n}} \\
 \overline{D} &\rightarrow (I_{n/\nu} \otimes L_\nu^{2\nu}) \overline{D} (I_{n/\nu} \otimes L_\nu^{2\nu}) \\
 \overline{P} &\rightarrow P \otimes I_2
 \end{aligned}$$

Example

$$\begin{aligned}
 \frac{\overline{\text{DFT}}_{mn}}{\text{vec}(\nu)} &\rightarrow \frac{(\overline{\text{DFT}}_m \otimes I_n) \overline{T}_n^{mn} (I_m \otimes \overline{\text{DFT}}_n) L_m^{mn}}{\text{vec}(\nu)} \\
 &\dots \\
 &\dots \\
 &\dots \\
 &\rightarrow \frac{(I_{m/\nu} \otimes L_\nu^{2\nu}) (\overline{\text{DFT}}_m \otimes I_n \otimes I_\nu) \overline{T}_n^{mn}}{(I_{m/\nu} \otimes (L_\nu^{2n} \otimes I_\nu)) (I_{2n/\nu} \otimes L_\nu^{2\nu}) (I_n \otimes L_\nu^{2\nu} \otimes I_\nu) (\overline{\text{DFT}}_n \otimes I_\nu)} (L_{m/\nu}^{mn} \otimes L_\nu^{2\nu})
 \end{aligned}$$

vectorized arithmetic
vectorized data accesses

Automatically Generate Base Case Library

- **Goal:** Given instruction set, generate base cases

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

- **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));`

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

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

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

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

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



Same Approach for Different Paradigms

Threading:

$$\begin{aligned} \text{DFT}_{\text{mp}}^{\text{mp}(\mu,\mu)} &\rightarrow \underbrace{(\text{DFT}_m \otimes I_n) \text{T}_n^{\text{mn}} (I_m \otimes \text{DFT}_n) L_m^{\text{mn}}}_{\text{mp}(\mu,\mu)} \\ &\dots \\ &\rightarrow \underbrace{(\text{DFT}_m \otimes I_n)}_{\text{mp}(\mu,\mu)} \underbrace{\text{T}_n^{\text{mn}}}_{\text{mp}(\mu,\mu)} \underbrace{(I_m \otimes \text{DFT}_n)}_{\text{mp}(\mu,\mu)} \underbrace{L_m^{\text{mn}}}_{\text{mp}(\mu,\mu)} \\ &\dots \\ &\rightarrow \underbrace{((L_m^{\text{mn}} \otimes I_{n/p}) \otimes I_p)}_{\text{mp}(\mu,\mu)} \underbrace{(I_p \otimes (\text{DFT}_m \otimes I_{n/p}))}_{\text{mp}(\mu,\mu)} \underbrace{((L_n^{\text{mn}} \otimes I_{m/p}) \otimes I_p)}_{\text{mp}(\mu,\mu)} \\ &\quad \left(\bigoplus_{i=0}^{p-1} \text{T}_n^{\text{mn},i} \right) \underbrace{(I_p \otimes (I_{n/p} \otimes \text{DFT}_n))}_{\text{mp}(\mu,\mu)} \underbrace{(I_p \otimes L_{n/p}^{\text{mn}})}_{\text{mp}(\mu,\mu)} \underbrace{((L_m^{\text{mn}} \otimes I_{m/p}) \otimes I_p)}_{\text{mp}(\mu,\mu)} \end{aligned}$$

Vectorization:

$$\begin{aligned} \text{DFT}_{\text{mv}}^{\text{mv}(\nu)} &\rightarrow \underbrace{(\text{DFT}_m \otimes I_n) \text{T}_n^{\text{mn}} (I_m \otimes \text{DFT}_n) L_m^{\text{mn}}}_{\text{vec}(\nu)} \\ &\dots \\ &\rightarrow \underbrace{(\text{DFT}_m \otimes I_n)}_{\text{vec}(\nu)} \underbrace{\text{T}_n^{\text{mn}}}_{\text{vec}(\nu)} \underbrace{(I_m \otimes \text{DFT}_n)}_{\text{vec}(\nu)} \underbrace{L_m^{\text{mn}}}_{\text{vec}(\nu)} \\ &\dots \\ &\rightarrow \underbrace{(I_{m/p} \otimes L_{n/p}^{\text{mn}})}_{\text{vec}(\nu)} \underbrace{(\text{DFT}_m \otimes I_{n/p} \otimes I_p)}_{\text{vec}(\nu)} \underbrace{\text{T}_n^{\text{mn}}}_{\text{vec}(\nu)} \\ &\quad \left(I_{m/p} \otimes (L_n^{\text{mn}} \otimes I_p) \right) \underbrace{(I_{n/p} \otimes (L_2^{\text{mn}} \otimes I_2))}_{\text{vec}(\nu)} \underbrace{(L_2^{\text{mn}} \otimes I_2)}_{\text{vec}(\nu)} \underbrace{(\text{DFT}_n \otimes I_p)}_{\text{vec}(\nu)} \\ &\quad \underbrace{((L_m^{\text{mn}} \otimes I_2) \otimes I_p)}_{\text{vec}(\nu)} \end{aligned}$$

GPUs:

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

Verilog for FPGAs:

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

- 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

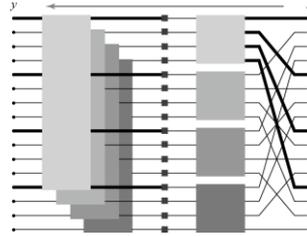
```
DFT(n, x, y) {  
  ...  
  for(i = ...) {  
    DFT_strided(m, x+mi, y+i, 1, k)  
  }  
  ...  
}
```

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

Challenge: Recursion Steps

- Cooley-Tukey FFT

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



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

```
void DFT(int n, cpx *y, cpx *x) {
    int k = choose_dft_radix(n);

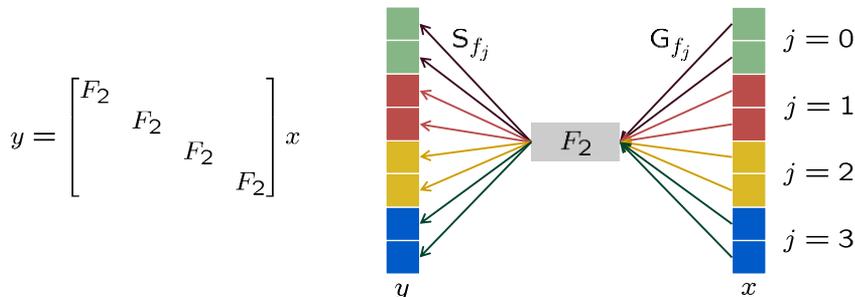
    for (int i=0; i < k; ++i)
        DFTrec(m, y + m*i, x + i, k, 1);
    for (int j=0; j < m; ++j)
        DFTscaled(k, y + j, t[j], m);
}
```

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

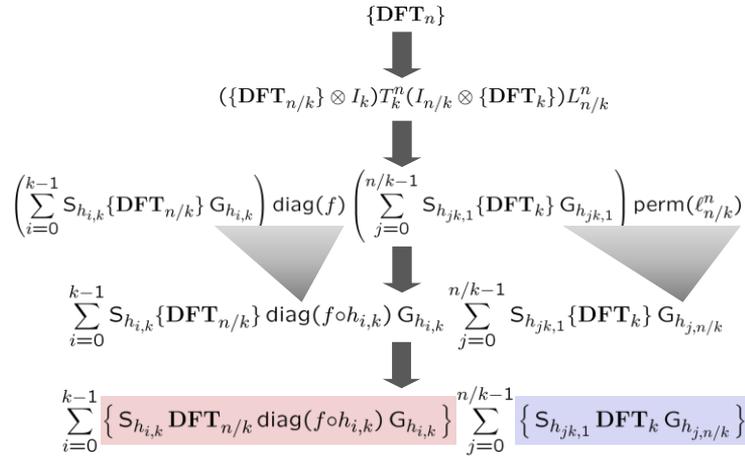
- Σ -SPL formulas = matrix factorizations ³

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



Find Recursion Step Closure

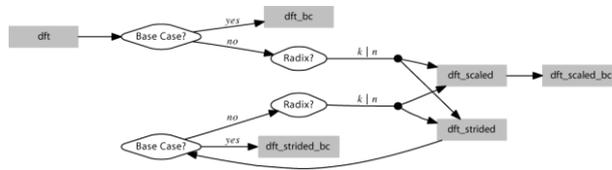
Voronenko, 2008



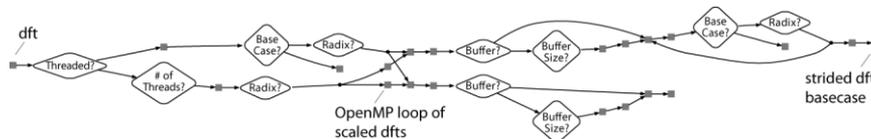
Repeat until closure

Recursion Step Closure: Examples

DFT: scalar code



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

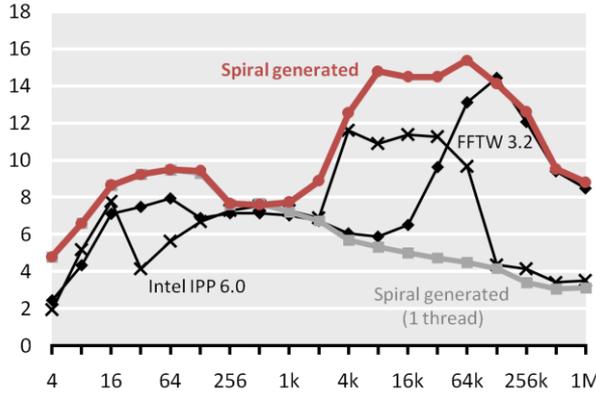
Organization

- Spiral: Basic system
- Vectorization
- General input size
- **Results**
- Final remarks

DFT on Intel Multicore

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

Performance [Gflop/s] vs. input size



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

➔ **5MB vectorized, threaded, general-size, adaptive library**
Spiral

Generating 100s of FFTWs

PhD thesis Voronenko, 2009

$$\begin{aligned}
 \text{DFT}_n &\rightarrow P_{k/2,2m}^\top (\text{DFT}_{2m} \oplus (I_{k/2-1} \otimes_i C_{2m} \text{rDFT}_{2m}(i/k))) (\text{RDFT}_k' \otimes I_m), \quad k \text{ even,} \\
 \begin{pmatrix} \text{RDFT}_n \\ \text{RDFT}'_n \\ \text{DHT}_n \\ \text{DHT}'_n \end{pmatrix} &\rightarrow (P_{k/2,2m}^\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_i 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{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_{2n}^{2n} \left(I_k \otimes_i \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-3n} &\rightarrow (Q_{k/2,2m}^\top \otimes I_2) (I_k \otimes_i \text{rDFT}_{2m}(i + 1/2/k)) (\text{RDFT-3k} \otimes I_m), \quad k \text{ even,} \\
 \text{DCT-2n} &\rightarrow P_{k/2,2m}^\top (\text{DCT-2}_{2m} K_{2m}^{2m} \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-3n} &\rightarrow \text{DCT-2}_{n,1}^\top, \\
 \text{DCT-4n} &\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-3k}) Q_{m/2,k} \\
 \text{DFT}_n &\rightarrow (\text{DFT}_k \otimes I_m) T_m^n (I_k \otimes \text{DFT}_m) L_k^n, \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^\top (I_1 \oplus \text{DFT}_{p-1}) D_p (I_1 \oplus \text{DFT}_{p-1}) R_p, \quad p \text{ prime} \\
 \text{DCT-3n} &\rightarrow (I_m \oplus J_m) L_m^n (\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 \oplus -J_{m-1} \\ \frac{1}{\sqrt{2}}(I_1 \oplus 2I_m) \end{bmatrix}, \quad n = 2m \\
 \text{DCT-4n} &\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(\begin{bmatrix} 1 \\ -1 \end{bmatrix} \otimes I_m \right) \oplus \left(\begin{bmatrix} -1 \\ 1 \end{bmatrix} \otimes I_m \right) J_{2m} \text{DCT-4}_{2m} \\
 \text{WHT}_{2k} &\rightarrow \prod_{i=1}^t (I_{2^{k_1+\dots+k_{i-1}}} \otimes \text{WHT}_{2^{k_i}} \otimes I_{2^{k_{i+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}$$

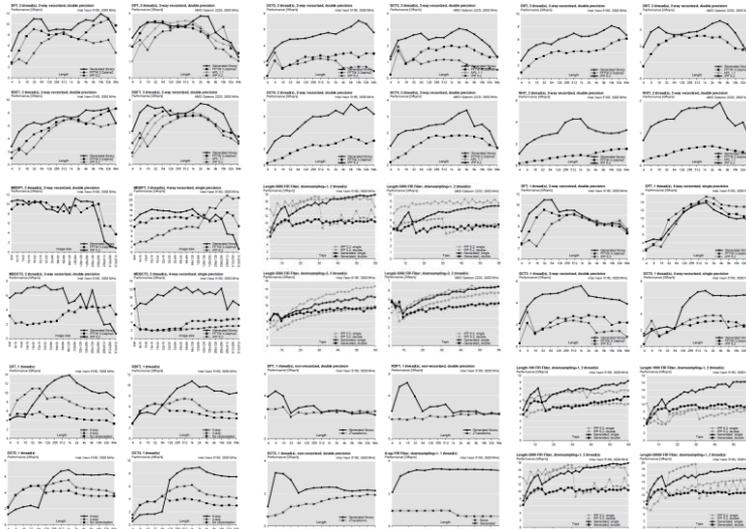
Generating 100s of FFTWs

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

Generating 100s of FFTWs

PhD thesis Voronenko, 2009



Computer generated Functions for Intel IPP 6.0

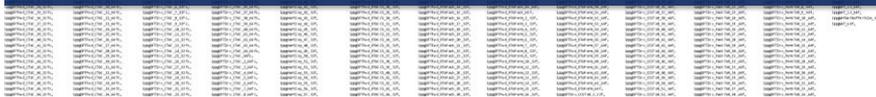


3984 C functions
1M lines of code

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)

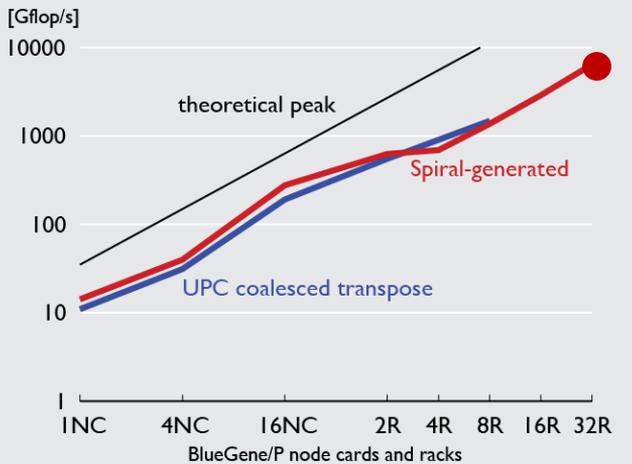
Computer generated

Results: SpiralGen Inc.



Very Large Scale: BG/P

HPC Challenge Global FFT on BlueGene/P



6.4 Tflop/s

32 racks
= 32K node cards
= 128K cores

2010 HPC Challenge Class I Award, Almasi et al.

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



- **Key ideas:**
Algorithm knowledge:
Domain specific symbolic representation
Platform knowledge:
Tagged rewrite rules, SIMD specification

DFT₆₄



```
void dft64(float *Y, float *X) {
    _mm512 U912, U913, U914, U915, ...
    _mm512 *a2153, *a2155;
    a2153 = ((*_mm512 * X); a1107 = *(a2153);
    a1108 = *((a2153 + 4); t1323 = _mm512_add_ps(a1107, a1108);
    t1204 = _mm512_sub_ps(a1107, a1108);
    Many more lines
    U926 = _mm512_wisupcoov_r32(.);
    a1121 = _mm512_mask031_pi(_mm512_mul_ps(_mm512_mask_or_pi(
        _mm512_sqr_1286_ps(0.70710678118654757).0xAAAA, a2154, U926), t1341);
    _mm512_mask_sub_ps(_mm512_sqr_1286_ps(0.70710678118654757), ...);
    _mm512_wisupcoov_r32(t1341, MM_SWI_REG_CDAB);
    U927 = _mm512_wisupcoov_r32
    Many more lines
}
```

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

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

Glimpse of other topics ...

35

LGen: Generator for Basic Linear Algebra

Spampinato & P, CGO 2014



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

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

 vectorization

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

 locality optimization

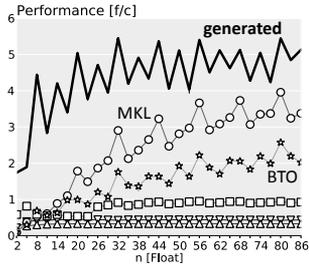
C Program

```
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 >
```

 code style
code level optimization

LGen: Sample Results

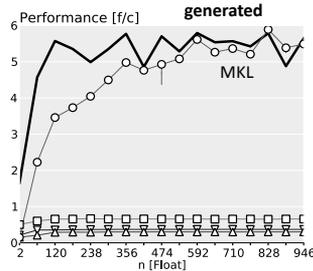
$$C = \alpha AB + \beta C$$



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

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

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



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

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

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

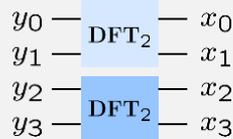
PL Support: Example Code Style

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



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

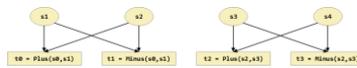
Data flow graph



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)
  }
}
```

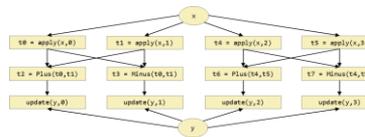
```
def f(x: Array[Rep[Double]],
    y: Array[Rep[Double]]) = {
  for (i <- 0 until 2) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```



scalarized

```
t0 = s0 + s1;
t1 = s0 - s1;
t2 = s2 + s3;
t3 = s2 - s3;
```

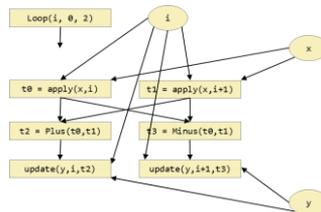
```
def f(x: Rep[Array[Double]],
    y: Rep[Array[Double]]) = {
  for (i <- 0 until 2) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```



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

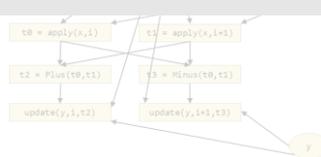
```
t0 = s0 + s1;
t1 = s0 - s1;
t2 = s2 + s3;
t3 = s2 - s3;
```

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)
  }
}
```

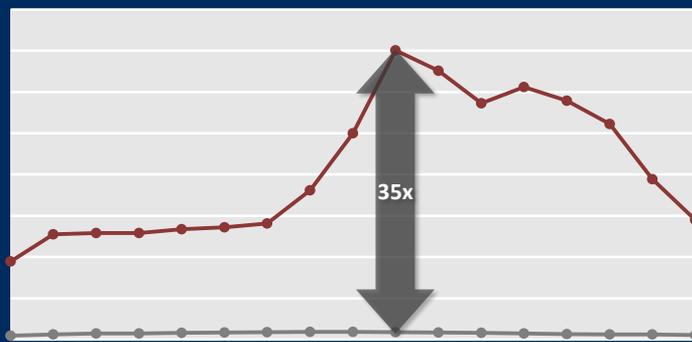
```
y: Rep[Array[Double]] = {
  for (i <- 0 until 2: Rep[Range]) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

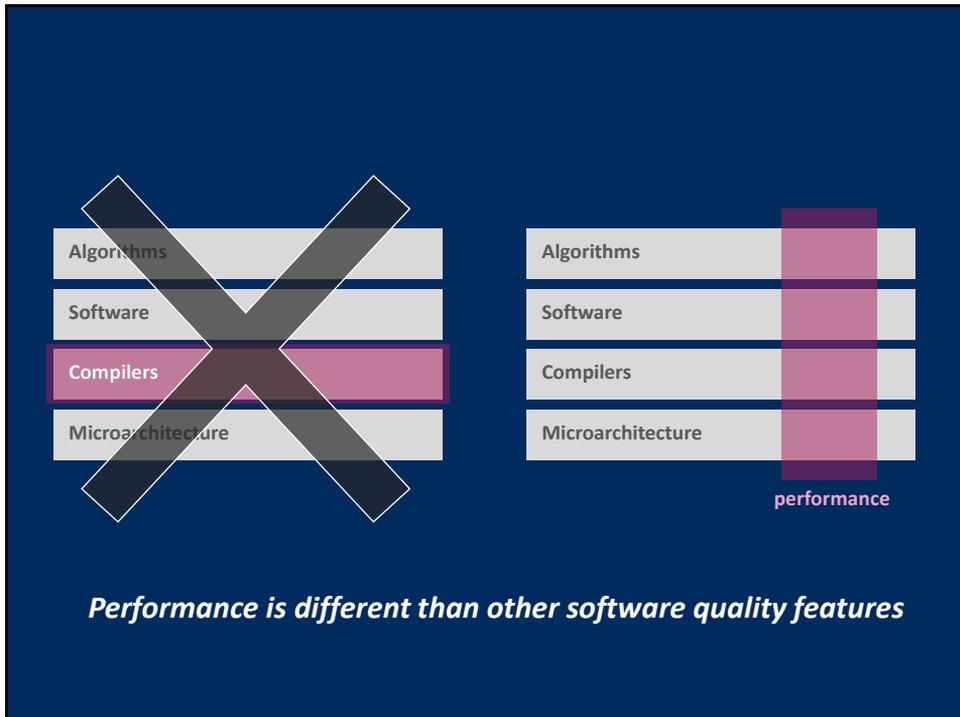


```
t0 = x[i];
t1 = x[i+1];
t2 = t0 + t1;
y[i] = t2;
t3 = t0 - t1;
y[i+1] = t3;
```

How to Write Fast Numerical Code

Conclusions





Research Questions

- 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?