How to Write Fast Numerical Code

Spring 2013

Lecture: Spiral (Computer generation of FFT code)

Instructor: Markus Püschel

TA: Georg Ofenbeck & Daniele Spampinato

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Rest of Semester

May 2013

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
			1	2	3	4
▨	6	7	8	9	10	11
▨	13	14	15 ●	16	17	18
▨	▨	21 ●	22 ●	23	24	25
▨	27 ●	28	29 ●	30	31 ●	

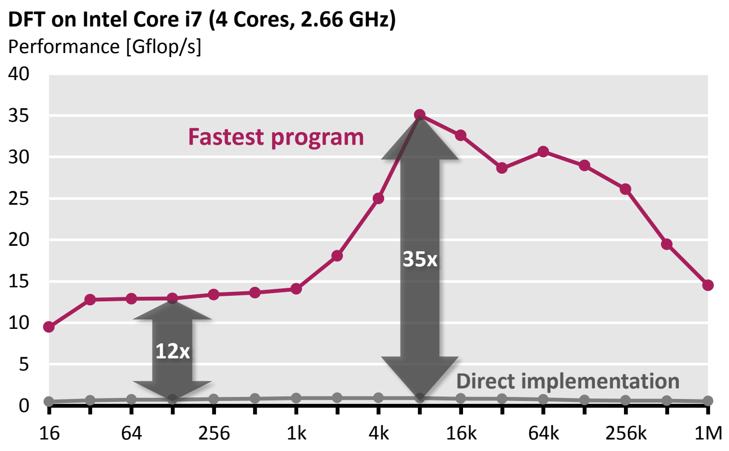
● **Today**

● **Lecture**

● **Project meetings**

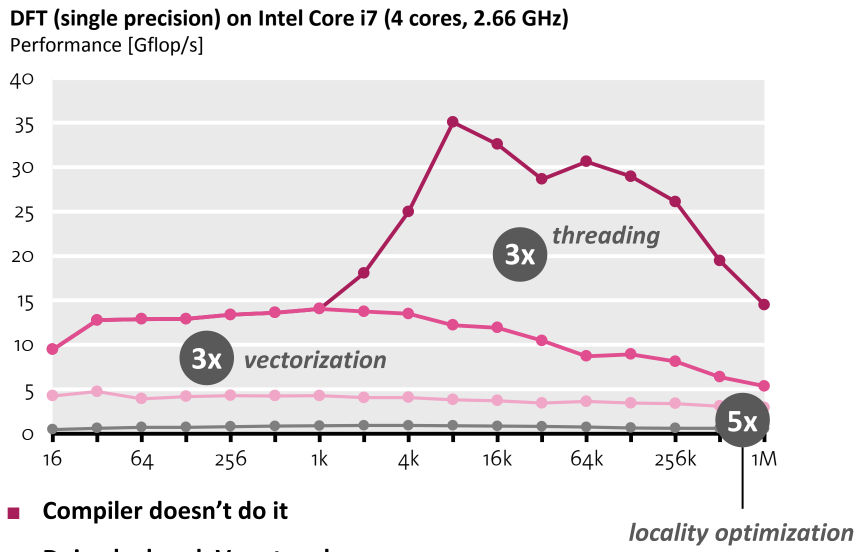
● **Project presentations**
- 10 minutes each
- random order
- random speaker

The Problem: Example DFT



- Same number of operations
- Best compiler

DFT: Analysis



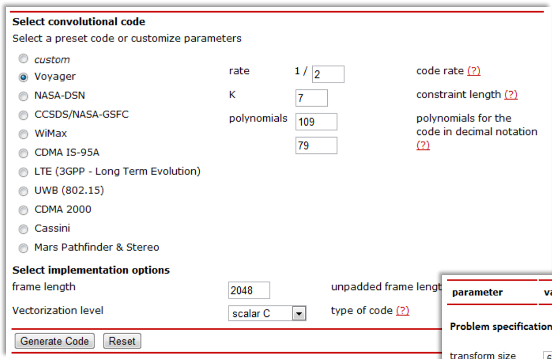
- Compiler doesn't do it
- Doing by hand: Very tough

Our Goal:

Computer writes high performance library code

Generate Code

"click"



Select convolutional code

Select a preset code or customize parameters

○ *custom*	rate	1 / 2	code rate (?)
● Voyager	K	7	constraint length (?)
○ NASA-DSN	polynomials	109	polynomials for the code in decimal notation (?)
○ CCSDS/NASA-GSFC		79	
○ WiMax			
○ CDMA IS-95A			
○ LTE (3GPP - Long Term Evolution)			
○ UWB (802.15)			
○ CDMA 2000			
○ Cassini			
○ Mars Pathfinder & Stereo			

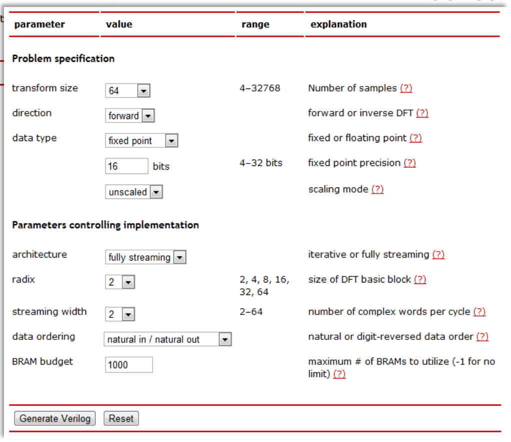
Select implementation options

frame length	2048	unpadded frame lengt
Vectorization level	scalar C	type of code (?)

Generate Code | Reset

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) (?)

Generate Verilog | Reset

@ 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)

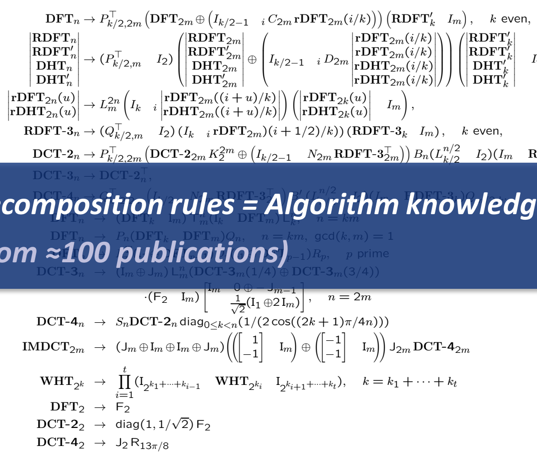
$$\begin{bmatrix}1&1&1&1\\1&i&-1&-i\\1&-1&1&-1\\1&-i&-1&i\end{bmatrix}x=\begin{bmatrix}1&\cdot&1&\cdot\\\cdot&1&\cdot&1\\1&\cdot&-1&\cdot\\\cdot&1&\cdot&-1\end{bmatrix}\begin{bmatrix}1&\cdot&\cdot&\cdot\\\cdot&1&\cdot&\cdot\\\cdot&\cdot&1&\cdot\\\cdot&\cdot&\cdot&i\end{bmatrix}\begin{bmatrix}1&1&\cdot&\cdot\\1&-1&\cdot&\cdot\\\cdot&\cdot&1&1\\\cdot&\cdot&1&-1\en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

Representation using matrix algebra

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

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

Decomposition Rules (>200 for >40 Transforms)



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

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

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

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

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

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

$$\mathbf{DCT\text{-}4}_n\to Q^\top_{\text{[illegible]}}\left(I_{k/2}\ \ N_{2m}\,\mathbf{RDFT\text{-}3}^\top_{2m}\right)B'_n(L^{n/2}_{\text{[illegible]}}\ \ I_2)(I_m\ \ \mathbf{RDFT\text{-}3}_k)Q_{\text{[illegible]}}$$

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

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

$$\mathbf{DFT}_p\ \to\ \text{[illegible]}_{p-1})R_p,\quad p\text{ prime}$$

$$\mathbf{DCT\text{-}3}_n\ \to\ (\mathrm{I}_m\oplus\mathrm{J}_m)\,\mathrm{L}^n_m(\mathbf{DCT\text{-}3}_m(1/4)\oplus\mathbf{DCT\text{-}3}_m(3/4))$$
$$\cdot(\mathrm{F}_2\ \ \mathrm{I}_m)\begin{bmatrix}\mathrm{I}_m & 0\oplus-\mathrm{J}_{m-1}\\ & \frac{1}{\sqrt{2}}(\mathrm{I}_1\oplus 2\,\mathrm{I}_m)\end{bmatrix},\quad n=2m$$

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

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

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

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

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

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

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

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];`

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

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

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

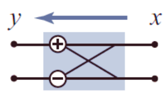
+ Search or Learning

Organization

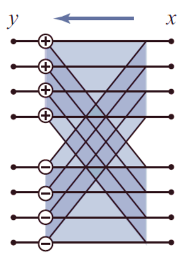
Example: Vectorization in Spiral

- **Relationship SPL expressions ↔ vectorization?**

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



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



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:**

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

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

$$\begin{aligned}
\mathrm{L}^{n\nu}_n &\rightarrow \left(\mathrm{I}_{n/\nu} \quad \mathrm{L}^{\nu^2}_\nu\right)\left(\mathrm{L}^{n}_{n/\nu} \quad \mathrm{I}_\nu\right)\\
\mathrm{L}^{n\nu}_\nu &\rightarrow \left(\mathrm{L}^{n}_\nu \quad \mathrm{I}_\nu\right)\left(\mathrm{I}_{n/\nu} \quad \mathrm{L}^{\nu^2}_\nu\right)\\
\mathrm{L}^{mn}_m &\rightarrow \left(\mathrm{L}^{mn/\nu}_m \quad \mathrm{I}_\nu\right)\left(\mathrm{I}_{mn/\nu^2} \quad \mathrm{L}^{\nu^2}_\nu\right)\left(\mathrm{I}_{n/\nu} \quad \mathrm{L}^{m}_{m/\nu} \quad \mathrm{I}_\nu\right)\\
\mathrm{I}_l \quad \mathrm{L}^{kmn}_n \quad \mathrm{I}_r &\rightarrow \left(\mathrm{I}_l \quad \mathrm{L}^{kn}_n \quad \mathrm{I}_{mr}\right)\left(\mathrm{I}_{kl} \quad \mathrm{L}^{mn}_n \quad \mathrm{I}_r\right)\\
\mathrm{I}_l \quad \mathrm{L}^{kmn}_n \quad \mathrm{I}_r &\rightarrow \left(\mathrm{I}_l \quad \mathrm{L}^{kmn}_{kn} \quad \mathrm{I}_r\right)\left(\mathrm{I}_l \quad \mathrm{L}^{kmn}_{mn} \quad \mathrm{I}_r\right)\\
\mathrm{I}_l \quad \mathrm{L}^{kmn}_{km} \quad \mathrm{I}_r &\rightarrow \left(\mathrm{I}_{kl} \quad \mathrm{L}^{mn}_m \quad \mathrm{I}_r\right)\left(\mathrm{I}_l \quad \mathrm{L}^{kn}_k \quad \mathrm{I}_{mr}\right)\\
\mathrm{I}_l \quad \mathrm{L}^{kmn}_{km} \quad \mathrm{I}_r &\rightarrow \left(\mathrm{I}_l \quad \mathrm{L}^{kmn}_k \quad \mathrm{I}_r\right)\left(\mathrm{I}_l \quad \mathrm{L}^{kmn}_m \quad \mathrm{I}_r\right)\\
\left(\mathrm{I}_m \quad A^{n\times n}\right) &\rightarrow \mathrm{L}^{mn}_\nu \left(\left(\mathrm{I}_{m/\nu} \quad A^{n\times n}\right) \quad \mathrm{I}_\nu\right)\mathrm{L}^{mn}_{mn/\nu}\\
\left(\mathrm{I}_m \quad A^{n\times n}\right)\mathrm{L}^{mn}_m &\rightarrow \left(\mathrm{I}_{m/\nu} \quad \mathrm{L}^{n\nu}_\nu\left(A^{n\times n} \quad \mathrm{I}_\nu\right)\right)\left(\mathrm{L}^{mn/\nu}_{m/\nu} \quad \mathrm{I}_\nu\right)\\
\mathrm{L}^{mn}_n\left(\mathrm{I}_m \quad A^{n\times n}\right) &\rightarrow \left(\mathrm{L}^{mn/\nu}_n \quad \mathrm{I}_\nu\right)\left(\mathrm{I}_{m/\nu} \quad \left(A^{n\times n} \quad \mathrm{I}_\nu\right)\mathrm{L}^{n\nu}_n\right)\\
\left(\mathrm{I}_k \quad \left(\mathrm{I}_m \quad A^{n\times n}\right)\mathrm{L}^{mn}_m\right)\mathrm{L}^{kmn}_k &\rightarrow \left(\mathrm{L}^{km}_k \quad \mathrm{I}_n\right)\left(\mathrm{I}_m \quad \left(\mathrm{I}_k \quad A^{n\times n}\right)\mathrm{L}^{kn}_k\right)\left(\mathrm{L}^{mn}_m \quad \mathrm{I}_k\right)\\
\mathrm{L}^{kmn}_{mn}\left(\mathrm{I}_k \quad \mathrm{L}^{mn}_n\left(\mathrm{I}_m \quad A^{n\times n}\right)\right) &\rightarrow \left(\mathrm{L}^{mn}_n \quad \mathrm{I}_k\right)\left(\mathrm{I}_m \quad \mathrm{L}^{kn}_n\left(\mathrm{I}_k \quad A^{n\times n}\right)\right)\left(\mathrm{L}^{km}_m \quad \mathrm{I}_n\right)\\
\overline{A\,B} &\rightarrow \overline{A}\,\overline{B}\\
\overline{A^{m\times m} \quad \mathrm{I}_\nu} &\rightarrow \left(\mathrm{I}_m \quad \mathrm{L}^{2\nu}_\nu\right)\left(\overline{A^{m\times m}} \quad \mathrm{I}_\nu\right)\left(\mathrm{I}_m \quad \mathrm{L}^{2\nu}_2\right)\\
\overline{\mathrm{I}_m \quad A^{n\times n}} &\rightarrow \mathrm{I}_m \quad \overline{A^{n\times n}}\\
\overline{D} &\rightarrow \left(\mathrm{I}_{n/\nu} \quad \mathrm{L}^{2\nu}_\nu\right)\vec{D}\left(\mathrm{I}_{n/\nu} \quad \mathrm{L}^{2\nu}_2\right)\\
\overline{P} &\rightarrow P \quad \mathrm{I}_2
\end{aligned}$$

Manipulation rules = Processor knowledge in Spiral

Example

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

$\ldots$

$\ldots$

$\ldots$

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

$$\underline{\left(\mathrm{I}_{\frac{m}{\nu}} \quad \left(\mathsf{L}_\nu^{2n} \quad \mathrm{I}_\nu\right)\left(\mathrm{I}_{\frac{2n}{\nu}} \quad \mathsf{L}_\nu^{\nu^2}\right)\left(\mathrm{I}_{\frac{n}{\nu}} \quad \mathsf{L}_2^{2\nu} \quad \mathrm{I}_\nu\right)} \underline{\left(\overline{\mathbf{DFT}}_n \quad \mathrm{I}_\nu\right)}\right) \underline{\left(\mathsf{L}_{\frac{m}{\nu}}^{\frac{mn}{\nu}} \quad \mathsf{L}_2^{2\nu}\right)}$$

vectorized arithmetic

vectorized data accesses

Automatically Generate Base Case Library

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

  $\nu = 4: \quad \left\{\mathsf{L}_2^4, \mathrm{I}_2 \quad \mathsf{L}_2^4, \mathsf{L}_2^4 \quad \mathrm{I}_2, \mathsf{L}_2^8, \mathsf{L}_4^8\right\}$

- ***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));
```

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

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

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

$\mathrm{L}_2^4 \otimes \mathrm{I}_2$

No base case

Base case

Same Approach for Different Paradigms

Threading:

$$\underbrace{\mathbf{DFT}_{mn}}_{\mathsf{smp}(p,\mu)} \rightarrow \underbrace{\left((\mathbf{DFT}_m\otimes \mathrm{I}_n)\mathsf{T}^{mn}_n(\mathrm{I}_m\otimes\mathbf{DFT}_n)\,\mathsf{L}^{mn}_m\right)}_{\mathsf{smp}(p,\mu)}$$

$$\dots$$

$$\rightarrow \underbrace{\left(\mathbf{DFT}_m\otimes \mathrm{I}_n\right)}_{\mathsf{smp}(p,\mu)}\underbrace{\mathsf{T}^{mn}_n}_{\mathsf{smp}(p,\mu)}\underbrace{\left(\mathrm{I}_m\otimes \mathbf{DFT}_n\right)}_{\mathsf{smp}(p,\mu)}\underbrace{\mathsf{L}^{nm}_m}_{\mathsf{smp}(p,\mu)}$$

$$\dots$$

$$\rightarrow \left((\mathsf{L}^{mp}_m\otimes \mathrm{I}_{n/p\mu})\otimes_\mu \mathrm{I}_\mu\right)\left(\mathrm{I}_p\otimes_\| (\mathbf{DFT}_m\otimes \mathrm{I}_{n/p})\right)\left((\mathsf{L}^{mp}_p\otimes \mathrm{I}_{n/p\mu})\otimes_\mu \mathrm{I}_\mu\right)$$
$$\left(\bigoplus_{i=0}^{p-1}{}_\| \,\mathsf{T}^{mn,i}_n\right)\left(\mathrm{I}_p\otimes_\|(\mathrm{I}_{m/p}\otimes\mathbf{DFT}_n)\right)\left(\mathrm{I}_p\otimes_\| \mathsf{L}^{mn/p}_{m/p}\right)\left((\mathsf{L}^{pn}_p\otimes \mathrm{I}_{m/p\mu})\otimes_\mu \mathrm{I}_\mu\right)$$

Vectorization:

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

$$\dots$$

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

$$\dots$$

$$\rightarrow (\mathrm{I}_{mn/\nu}\otimes \underbrace{\mathsf{L}^{2\nu}_\nu}_{\mathsf{sse}})(\overline{\mathrm{DFT}_m\otimes \mathrm{I}_{n/\nu}}\vec{\otimes}\, \mathrm{I}_\nu)\underbrace{\overline{(\mathsf{T}^{mn}_n)}^{\nu}}_{\mathsf{sse}}$$
$$\left(\mathrm{I}_{m/\nu}\otimes(\overline{\mathsf{L}^n_\nu}\vec{\otimes}\, \mathrm{I}_\nu)(\mathrm{I}_{n/\nu}\otimes(\mathsf{L}^{2\nu}_\nu\vec{\otimes}\, \mathrm{I}_\nu)(\mathrm{I}_2\otimes\underbrace{\mathsf{L}^{\nu^2}_\nu}_{\mathsf{sse}})(\mathsf{L}^{2\nu}_2\vec{\otimes}\, \mathrm{I}_\nu))(\overline{\mathrm{DFT}_n}\vec{\otimes}\, \mathrm{I}_\nu)\right)$$
$$\left((\mathsf{L}^{mn}_m\otimes \mathrm{I}_2)\vec{\otimes}\, \mathrm{I}_\nu\right)(\mathrm{I}_{mn/\nu}\otimes\underbrace{\mathsf{L}^{2\nu}_2}_{\mathsf{sse}})$$

GPUs:

$$\underbrace{\left(\mathbf{DFT}_{r^k}\right)}_{\mathsf{gpu}(t,c)} \rightarrow \underbrace{\left(\prod_{i=0}^{k-1}\mathsf{L}^{r^k}_r\left(\mathrm{I}_{r^{k-1}}\otimes\mathbf{DFT}_r\right)\left(\mathsf{L}^{r^k}_{r^{k-i-1}}(\mathrm{I}_{r^i}\otimes\mathsf{T}^{r^{k-i}}_{r^{k-i-1}})\underbrace{\mathsf{L}^{r^k}_{r^{i+1}}}_{\mathsf{vec}(c)}\right)\right)\mathsf{R}^{r^k}_r}_{\mathsf{gpu}(t,c)}$$

$$\dots$$

$$\rightarrow \left(\prod_{i=0}^{k-1}(\mathsf{L}^{r^n/2}_r\vec{\otimes}\,\mathrm{I}_2)\left(\mathrm{I}_{r^{n-1}/2}\otimes_\times \underbrace{(\mathbf{DFT}_r\vec{\otimes}\,\mathrm{I}_2)\,\mathsf{L}^{2r}_r}_{\mathsf{shd}(t,c)}\right)\mathsf{T}_i\right)$$
$$(\mathsf{L}^{r^n/2}_r\vec{\otimes}\,\mathrm{I}_2)(\mathrm{I}_{r^{n-1}/2}\otimes_\times \underbrace{\mathsf{L}^{2r}_r}_{\mathsf{shd}(t,c)})(\mathsf{R}^{r^{n-1}}_r\vec{\otimes}\,\mathrm{I}_r)$$

Verilog for FPGAs:

$$\underbrace{\left(\mathbf{DFT}_{r^k}\right)}_{\mathsf{stream}(r^s)} \rightarrow \underbrace{\left[\prod_{i=0}^{k-1}\mathsf{L}^{r^k}_r\left(\mathrm{I}_{r^{k-1}}\otimes\mathbf{DFT}_r\right)\left(\mathsf{L}^{r^k}_{r^{k-i-1}}(\mathrm{I}_{r^i}\otimes\mathsf{T}^{r^{k-i}}_{r^{k-i-1}})\,\mathsf{L}^{r^k}_{r^{i+1}}\right)\right]\mathsf{R}^{r^k}_r}_{\mathsf{stream}(r^s)}$$

$$\dots$$

$$\rightarrow \left[\prod_{i=0}^{k-1}\underbrace{\mathsf{L}^{r^k}_r}_{\mathsf{stream}(r^s)}\underbrace{\left(\mathrm{I}_{r^{k-1}}\otimes\mathbf{DFT}_r\right)}_{\mathsf{stream}(r^s)}\underbrace{\left(\mathsf{L}^{r^k}_{r^{k-i-1}}(\mathrm{I}_{r^i}\otimes\mathsf{T}^{r^{k-i}}_{r^{k-i-1}})\,\mathsf{L}^{r^k}_{r^{i+1}}\right)}_{\mathsf{stream}(r^s)}\right]\underbrace{\mathsf{R}^{r^k}_r}_{\mathsf{stream}(r^s)}$$

$$\dots$$

$$\rightarrow \left[\prod_{i=0}^{k-1}\underbrace{\mathsf{L}^{r^k}_r}_{\mathsf{stream}(r^s)}\left(\mathrm{I}_{r^{k-s-1}}\otimes_s(\mathrm{I}_{r^{s-1}}\otimes\mathbf{DFT}_r)\right)\underbrace{\mathsf{T}'_i}_{\mathsf{stream}(r^s)}\right]\underbrace{\mathsf{R}^{r^k}_r}_{\mathsf{stream}(r^s)}$$

- 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

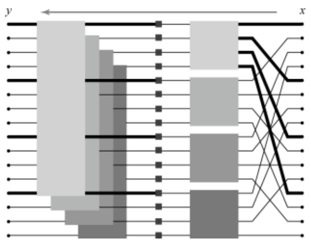
```
DFT(n, x, y) {
  …
  for(i = …) {
    DFT_strided(m, x+mi, y+i, 1, k)
  }
  …
}
```

- 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

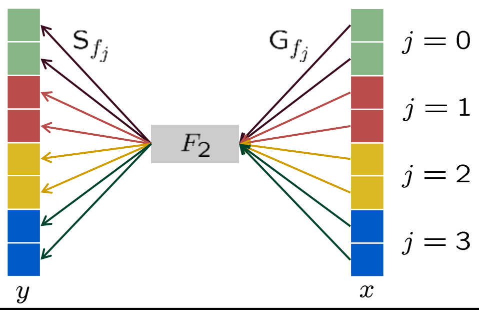
- Four additional matrix constructs: Σ, G, S, Perm

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

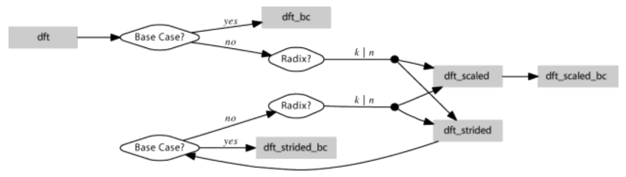
Voronenko, 2008

$$\{\mathbf{DFT}_n\}$$
$$\Downarrow$$
$$(\{\mathbf{DFT}_{n/k}\} \otimes I_k) T^n_k (I_{n/k} \otimes \{\mathbf{DFT}_k\}) L^n_{n/k}$$
$$\Downarrow$$
$$\left(\sum_{i=0}^{k-1} \mathsf{S}_{h_{i,k}} \{\mathbf{DFT}_{n/k}\}\, \mathsf{G}_{h_{i,k}}\right) \mathsf{diag}(f) \left(\sum_{j=0}^{n/k-1} \mathsf{S}_{h_{jk,1}} \{\mathbf{DFT}_k\}\, \mathsf{G}_{h_{jk,1}}\right) \mathsf{perm}(\ell^n_{n/k})$$
$$\Downarrow$$
$$\sum_{i=0}^{k-1} \mathsf{S}_{h_{i,k}} \{\mathbf{DFT}_{n/k}\}\, \mathsf{diag}(f \circ h_{i,k})\, \mathsf{G}_{h_{i,k}} \sum_{j=0}^{n/k-1} \mathsf{S}_{h_{jk,1}} \{\mathbf{DFT}_k\}\, \mathsf{G}_{h_{j,n/k}}$$
$$\Downarrow$$
$$\sum_{i=0}^{k-1} \left\{ \mathsf{S}_{h_{i,k}}\, \mathbf{DFT}_{n/k}\, \mathsf{diag}(f \circ h_{i,k})\, \mathsf{G}_{h_{i,k}} \right\} \sum_{j=0}^{n/k-1} \left\{ \mathsf{S}_{h_{jk,1}}\, \mathbf{DFT}_k\, \mathsf{G}_{h_{j,n/k}} \right\}$$

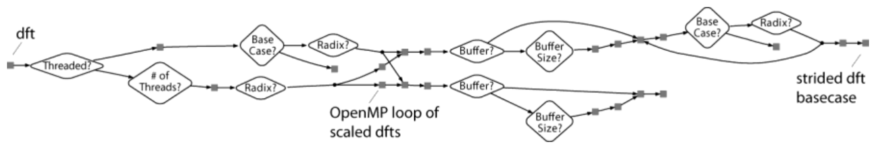
Repeat until closure

Recursion Step Closure: Examples

DFT: scalar code



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

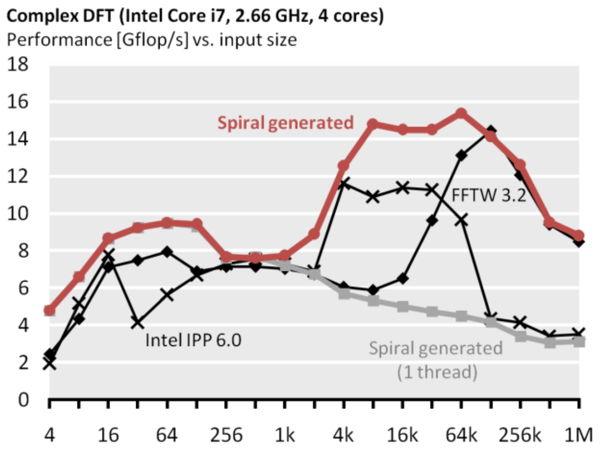
- **Derivation of library structure**
  Rewriting
  Other methods

general input size library

Organization

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

DFT on Intel Multicore



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

$\xrightarrow[\textit{Spiral}]{}$ **5MB vectorized, threaded, general-size, adaptive library**

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.

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

$$\mathrm{DFT}_{64}$$

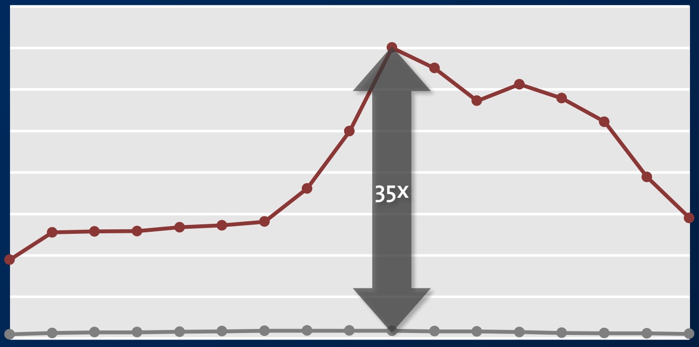
```
void dft64(float  *Y, float  *X) {
  __m512 U912, U913, U914, U915,...
  __m512  *a2153, *a2155;
  a2153 = ((__m512  *) X);  s1107 = *(a2153);
  s1108 = *((a2153 + 4));   t1323 = _mm512_add_ps(s1107,s1108);
  t1324 = _mm512_sub_ps(s1107,s1108);
    <many more lines>
  U926 = _mm512_swizupconv_r32(…);
  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>
}
```

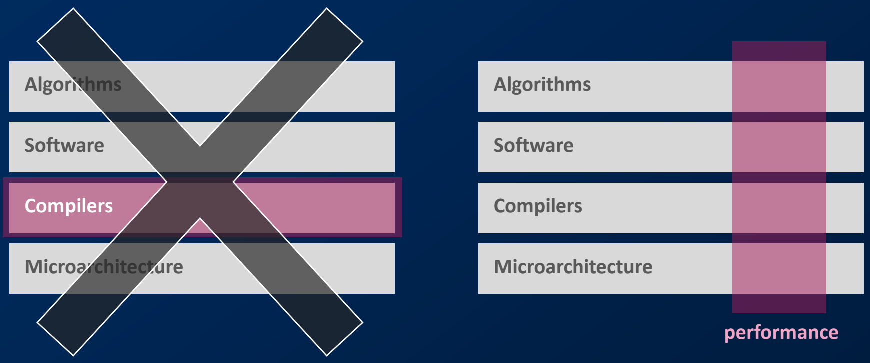
- **Key ideas:**
  Algorithm knowledge:
  Domain specific symbolic representation

  Platform knowledge:
  Tagged rewrite rules, SIMD specification

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

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





Performance is different than other software quality features

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?**