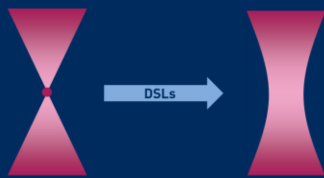
Exploiting Domain-Specific Knowledge: Spiral

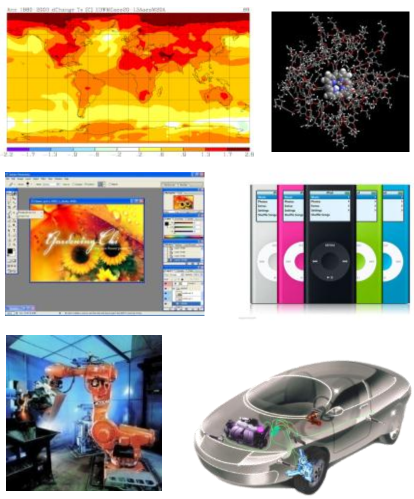
Markus Püschel & Georg Ofenbeck

Computer Science
ETH *zürich*

SPIRAL
www.spiral.net



Mathematical Computing



Science simulations

Audio, image, Video processing

Signal processing, communication, control

Security

Machine learning, data analytics

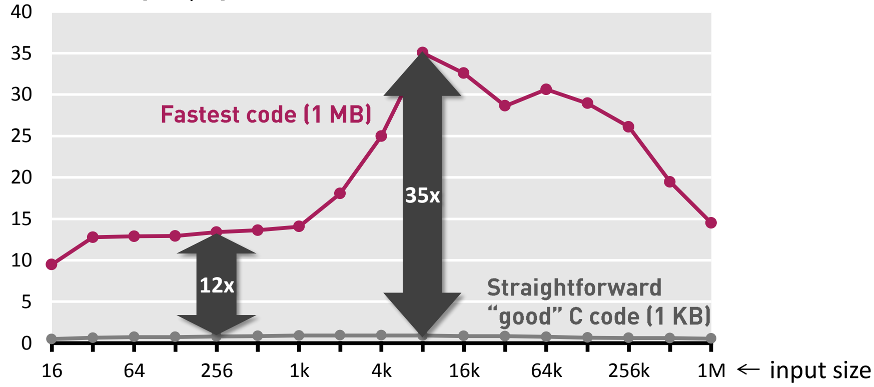
Optimization

Highest performance is often crucial

Example: Discrete Fourier Transform

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

Performance [Gflop/s]

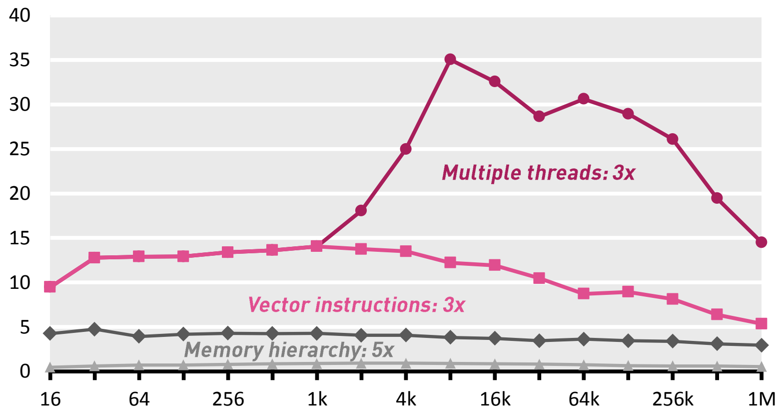


Vendor compiler, best flags

Roughly same operations count

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

Performance [Gflop/s]



Compiler doesn’t do the job

Doing by hand = restructure algorithm for locality & parallelism, handle choices, choose proper code style, use vector intrinsics, ...
= nightmare

Optimization: Register Locality and ILP

```
// straightforward code
for(i = 0; i < N; i += 1)
  for(j = 0; j < N; j += 1)
    for(k = 0; k < N; k += 1)
      c[i][j] += a[i][k]*b[k][j];
```

Concise and slow

Removes aliasing

Enables register allocation and instruction scheduling

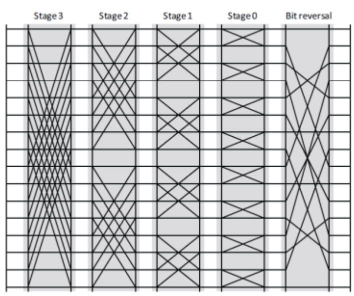
Compiler does not do well:

- often illegal
- many choices

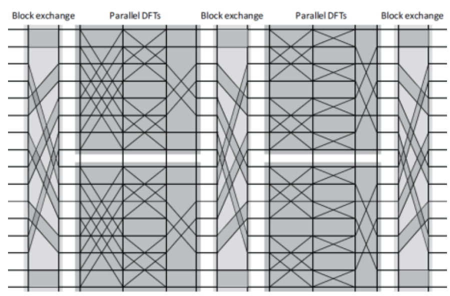
```
// unrolling + scalar replacement
for(i = 0; i < N; i += MU) {
    for(j = 0; j < N; j += NU) {
      for(k = 0; k < N; k += KU) {
        t1 = A[i*N + k];
        t2 = A[i*N + k + 1];
        t3 = A[i*N + k + 2];            } load
        t4 = A[i*N + k + 3];
        t5 = A[(i + 1)*N + k];
        <more copies>

        t10 = t1  * t9;
        t17 = t17 + t10;
        t21 = t1  * t8;
        t18 = t18 + t21;
        t12 = t5  * t9;                 } compute
        t19 = t19 + t12;
        t13 = t5  * t8;
        t20 = t20 + t13;
        <more ops>

        C[i*N + j]         = t17;
        C[i*N + j + 1]     = t18;       } store
        C[(i+1)*N + j]     = t19;
        C[(i+1)*N + j + 1] = t20;
      }
    }
  }
```

Ugly and fast

Optimization for Parallelism (Threads)



Parallelism is present, but is not in the "right shape"

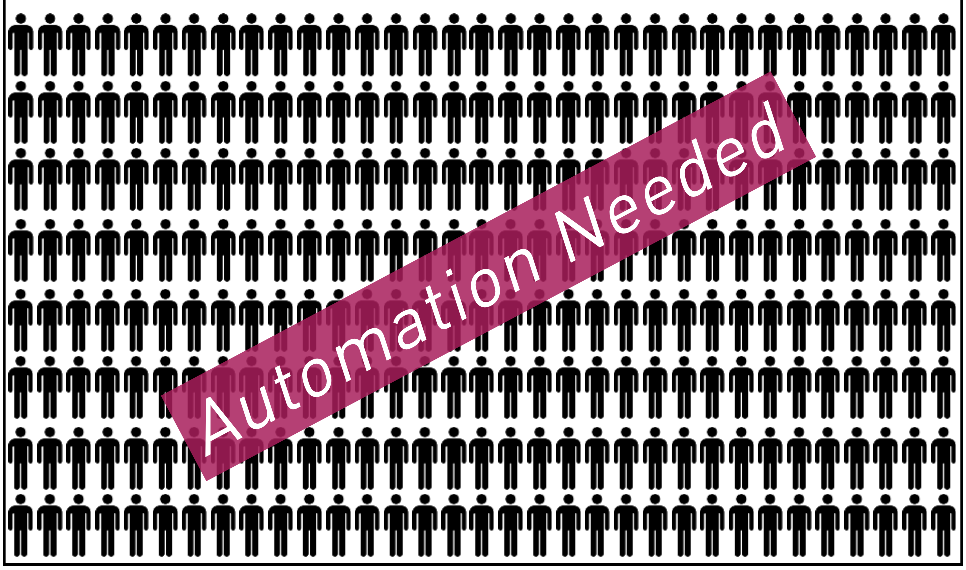


Restructured for locality and parallelism (shared memory, 2 cores, 2 elements per cache line)

Compiler usually does not do

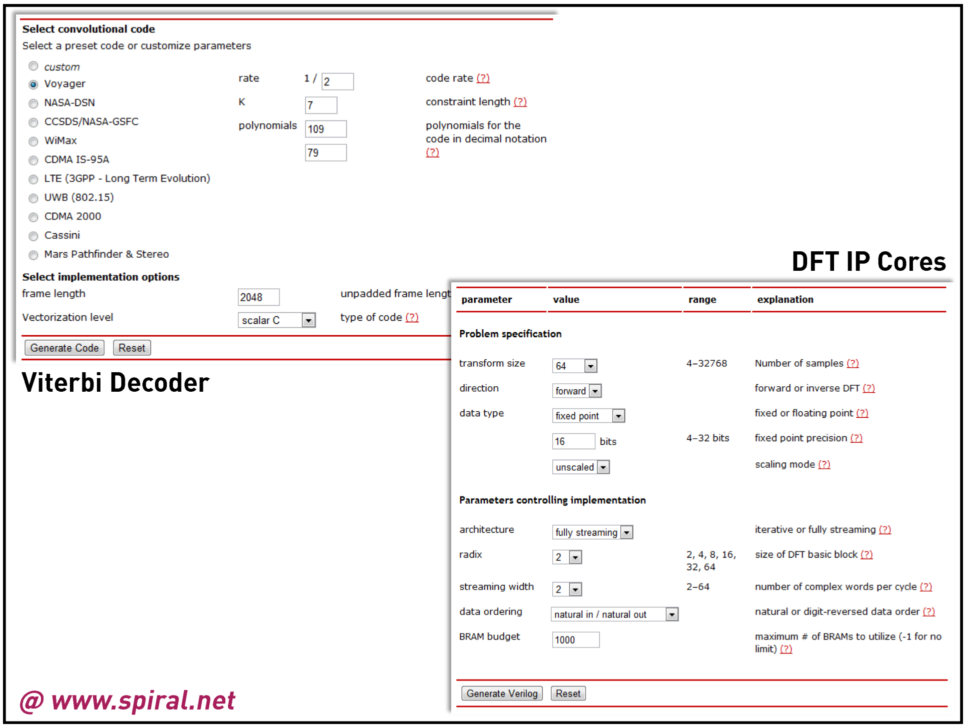
- analysis may be unfeasible
- may require algorithm changes
- may require domain knowledge
- may require processor parameters

Current practice: Thousands of programmers re-implement and re-optimize the same functionality for every new processor and for every new processor generation



Goal:

Computer generation of high performance code for ubiquitous performance-critical components



Possible Approach:

Capturing algorithm knowledge:
Domain-specific languages (DSLs)

Structural optimization:
Rewriting systems

High performance code style:
Compiler

Decision making for choices:
Machine learning

Spiral: Program Generation for Performance (www.spiral.net)

Franz Franchetti
Yevgen Voronenko
Jianxin Xiong
Bryan Singer
Srinivas Chellappa
Frédéric de Mesmay
Peter Milder
José Moura
David Padua
Jeremy Johnson
James Hoe
<many more>

funding: DARPA, NSF, ONR, Intel

Linear Transforms

$$\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} = y = Tx \leftarrow \boxed{T\cdot} \leftarrow x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix}$$

Output ***Input***

Example: $T = \mathbf{DFT}_n = [e^{-2k\ell\pi i/n}]_{0 \le k,\ell < n}$

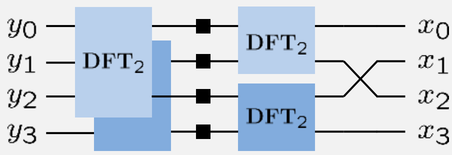
Algorithms: Example FFT, n = 4

Fast Fourier transform (FFT):

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

12 adds, 4 mults | 4 adds | 1 mult | 4 adds | 0 adds/mults

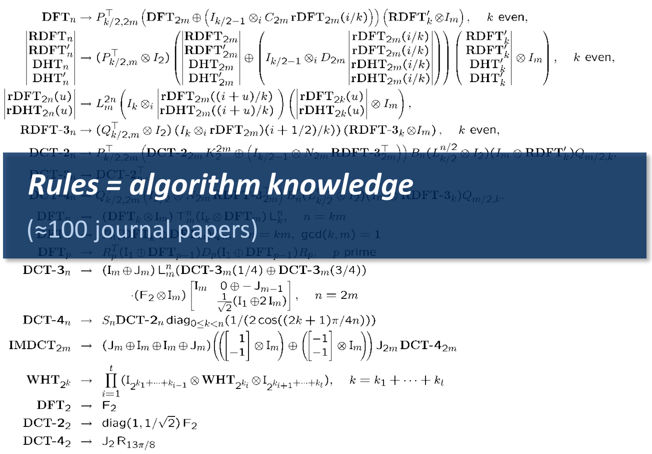
Data flow graph



Description with matrix algebra (SPL)

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

Decomposition Rules (>200 for >40 Transforms)



SPL to Code

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

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

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

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

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

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

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

Gives reasonable, straightforward code

Program Generation in Spiral

Transform $\mathbf{DFT}_8$

↓ *Decomposition rules (algorithm knowledge)*

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 (Σ-SPL)

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

↻ **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 >
```

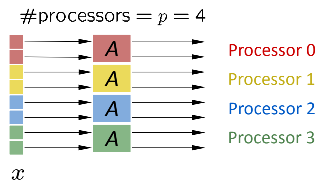
↻ **code style**
code level optimization

+ Search or Learning for Choices

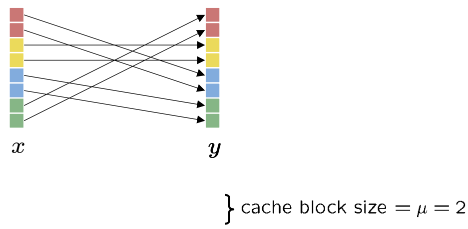
SPL to Shared Memory Code: Basic Idea

"Good" SPL structures

$$y = \left(I_p \otimes A\right) x$$



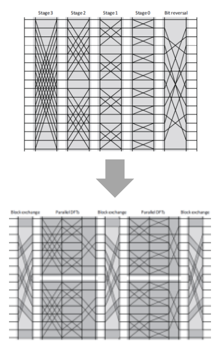
$$y = \left(P \otimes I_\mu\right) x$$



Rewriting: Bad structures → good structures

Example: SMP Parallelization

Franchetti, Voronenko & P, SC 2006



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

$$\ldots$$

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

$$\ldots$$

$$\to \left((L_m^{mp} \otimes I_{n/p\mu}) \otimes I_\mu\right)\left(I_p \otimes (\mathbf{DFT}_m \otimes I_{n/p})\right)\left((L_p^{mp} \otimes I_{n/p\mu}) \otimes I_\mu\right)$$

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

load-balanced, no false sharing

One rewriting system for every platform paradigm:
SIMD, distributed memory parallelism, FPGA, …

Same Approach for Different Paradigms

Threading:

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

Vectorization:

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

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

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

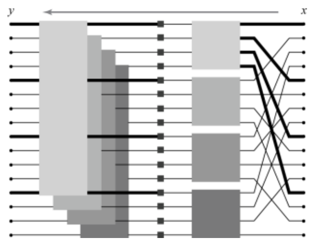
```
DFT(n, x, y) {
  …
  for(i = …) {
    DFT_strided(m, x+mi, y+i, 1, k)
  }
  …
}
```

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

Challenge: Recursive Steps Needed

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

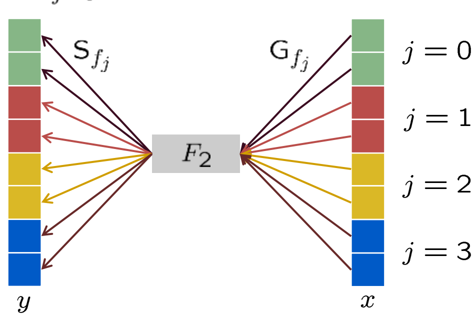
Four additional matrix constructs: Σ, G, S, Perm

- *Σ (sum)* — *matrix 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, thesis 2008

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

$$(\{\mathbf{DFT}_{n/k}\} \otimes I_k) T^n_k (I_{n/k} \otimes \{\mathbf{DFT}_k\}) L^n_{n/k}$$

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

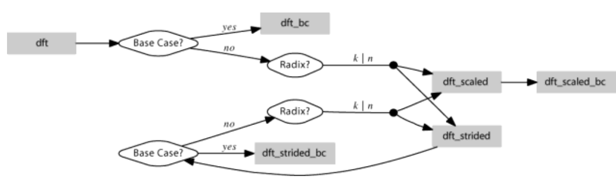
$$\sum_{i=0}^{k-1} \mathsf{S}_{h_{i,k}} \{\mathbf{DFT}_{n/k}\}\, \mathsf{diag}(f \circ h_{i,k})\, \mathsf{G}_{h_{i,k}} \quad \sum_{j=0}^{n/k-1} \mathsf{S}_{h_{jk,1}} \{\mathbf{DFT}_k\}\, \mathsf{G}_{h_{j,n/k}}$$

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

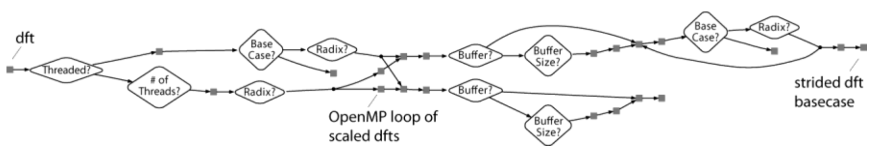
Repeat until closure

Recursion Step Closure: Examples

DFT: scalar code



DFT: full-fledged (vectorized and parallel code)



Generating Dozens of "FFTWs"

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

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

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

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

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

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

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

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

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

$$\mathbf{DFT}_p \rightarrow R^T_p(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1})D_p(\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1})R_p, \quad p \text{ prime}$$

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

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

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

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

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

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

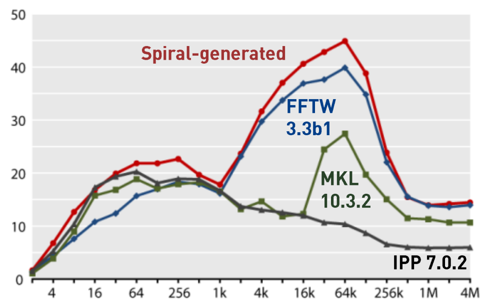
$$\mathbf{DCT\text{-}4}_2 \rightarrow \mathrm{J}_2 \mathrm{R}_{13\pi/8}$$

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

It Really Works

DFT on Sandybridge (3.3 GHz, 4 Cores, AVX)

Performance [Gflop/s]



$$\mathbf{DFT}_n \rightarrow (\mathbf{DFT}_k \otimes I_m)T^n_m(I_k \otimes \mathbf{DFT}_m)L^n_k$$

$$\mathbf{DFT}_n \rightarrow P^{\top}_{k/2,2m}\left(\mathbf{DFT}_{2m} \oplus \left(I_{k/2-1} \otimes_i C_{2m}\, \mathbf{rDFT}_{2m}(i/k)\right)\right)(\mathbf{RDFT}_k \otimes I_m)$$

$$\mathbf{RDFT}_n \rightarrow (P^{\top}_{k/2,m} \otimes I_2)\left(\mathbf{RDFT}_{2m} \oplus \left(I_{k/2-1} \otimes_i D_{2m}\, \mathbf{rDFT}_{2m}(i/k)\right)\right)(\mathbf{RDFT}_k \otimes I_m)$$

$$\mathbf{rDFT}_{2n}(u) \rightarrow L^{2n}_m\,(I_k \otimes_i \mathbf{rDFT}_{2m}((i+u)/k))\,(\mathbf{rDFT}_{2k}(u) \otimes I_m)$$

vectorized, threaded,
platform-tuned library
(5 MB source code)

Computer generated Functions for Intel IPP

3984 C functions

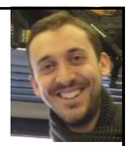
1M lines of code

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.

LGen: Generator for Basic Linear Algebra



Spampinato & P, CGO 2014

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

↓

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

↓

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

↓

C Program

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

vectorization

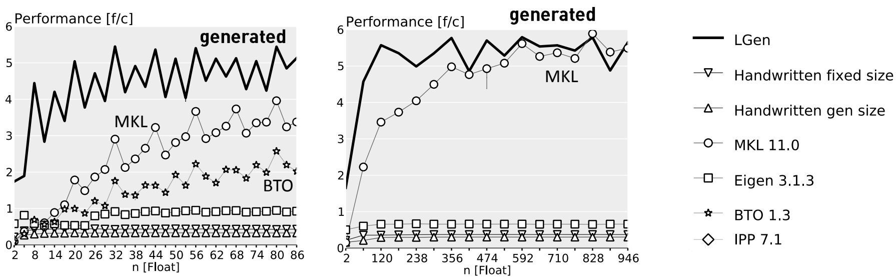
locality optimization

code style
code level optimization

LGen: Sample Results

$C = \alpha AB + \beta C$

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



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

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

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

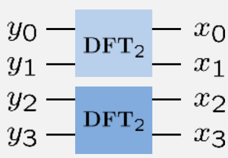
$B \in \mathbb{R}^{4\times n}$

PL Support: Example Code Style

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

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

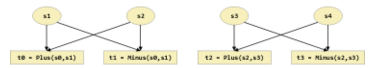
Data flow graph



Scala function

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

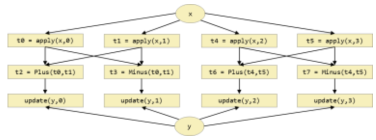
```
def f(x: Array[Rep[Double]],
      y: Array[Rep[Double]]) = {
      for (i <- 0 until 2) {
        y(2*i)   = x(i*2) + x(i*2+1)
        y(2*i+1) = x(i*2) - x(i*2+1)
      }
    }
```



scalarized

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

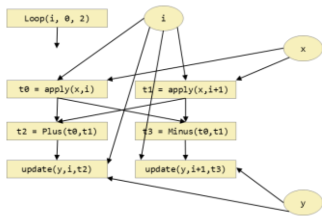
```
def f(x: Rep[Array[Double]],
      y: Rep[Array[Double]]) = {
      for (i <- 0 until 2) {
        y(2*i)   = x(i*2) + x(i*2+1)
        y(2*i+1) = x(i*2) - x(i*2+1)
      }
    }
```



unrolled, scalar repl.

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

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



looped, scalar repl.

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

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)
      }
}
```

Related Work

Program generators for performance

FFTW codelet generator (Frigo)
Flame (van de Geijn, Quintana-Orti, Bientinesi, …)
cvxgen (Mattingley, Boyd)
PetaBricks (Ansel, Amarasinghe, …)
Spiral

Autotuning

ATLAS/PhiPAC (Whaley, Bilmes, Demmel, Dongarra, …)
FFTW adaptive library (Frigo, Johnson)
OSKI (Vuduc et al.)
Adaptive sorting (Li et al.)

Environments for DSLs and program generation

see this workshop

Automatically from Math to Fast Code

Principles

Capturing algorithm knowledge:
Mathematical DSLs

Structural optimization:
Rewriting

$$
\begin{aligned}
\mathbf{DFT}_n &\rightarrow (\mathbf{DFT}_k \otimes \mathrm{I}_m)\, \mathrm{T}^n_m (\mathrm{I}_k \otimes \mathbf{DFT}_m)\, \mathrm{L}^n_k, \quad n = km \\
\mathbf{DFT}_n &\rightarrow P_n(\mathbf{DFT}_k \otimes \mathbf{DFT}_m) Q_n, \quad n = km,\ \gcd(k,m) = 1 \\
\mathbf{DCT\text{-}4}_n &\rightarrow S_n \mathbf{DCT\text{-}2}_n\, \mathrm{diag}_{0\le k<n}(1/(2\cos((2k+1)\pi/4n))) \\
\mathbf{IMDCT}_{2m} &\rightarrow (\mathrm{J}_m \oplus \mathrm{I}_m \oplus \mathrm{I}_m \oplus \mathrm{J}_m)\left(\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} \otimes \mathrm{I}_m\right) \oplus \left(\begin{bmatrix} -1 \\ -1 \end{bmatrix} \otimes \mathrm{I}_m\right)\right) \mathrm{J}_{2m}\, \mathbf{DCT\text{-}4}_{2m}
\end{aligned}
$$

Decision making:
Search and learning

$$
\underbrace{A_m \otimes \mathrm{I}_n}_{\mathrm{smp}(p,\mu)} \rightarrow \underbrace{\mathrm{L}^{mn}_m}_{\mathrm{smp}(p,\mu)} \left(\mathrm{I}_p \otimes_{\|} (\mathrm{I}_{n/p} \otimes A_m)\right) \underbrace{\mathrm{L}^{mn}_n}_{\mathrm{smp}(p,\mu)}
$$

Key Challenges

New domains (linear algebra, filters, …)

Programming language support (DSLs, staging)

More information: www.spiral.net