How to Write Fast Code

18-645, spring 2008

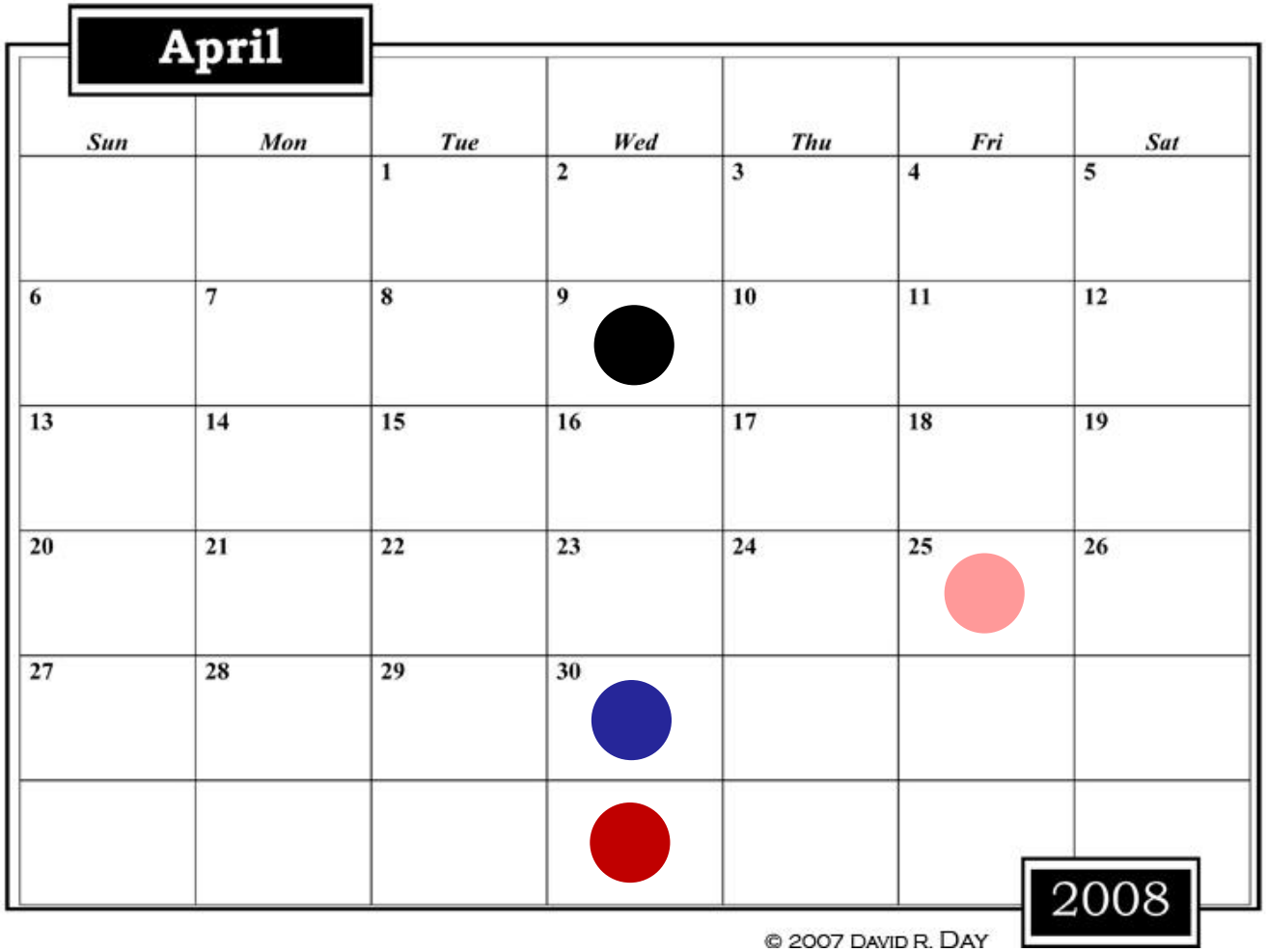
23rd Lecture, Apr. 9th

Instructor: Markus Püschel

TAs: Srinivas Chellappa (Vas) and Frédéric de Mesmay (Fred)

Research Project

- Current status

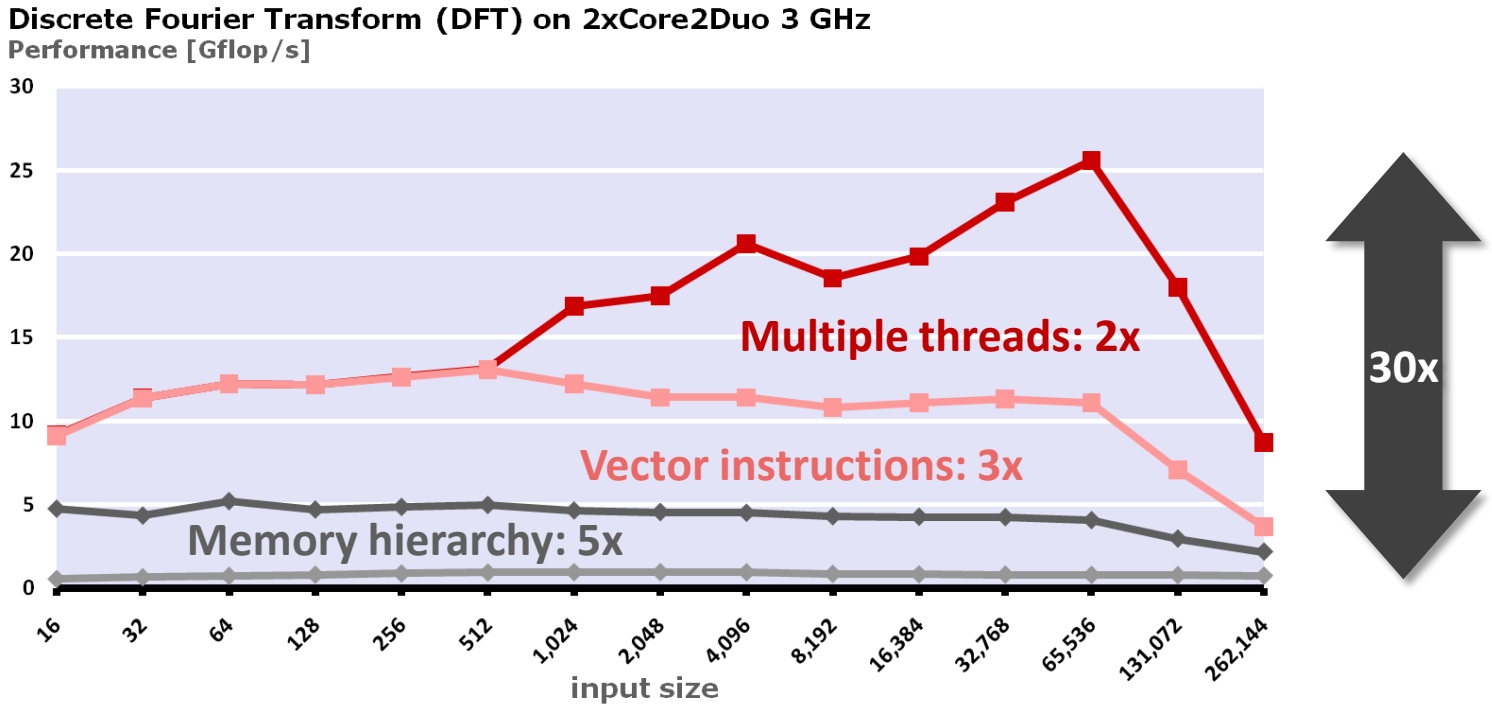


- Today
- Papers due
- Last class: poster session 5:30 – 8:30
- Final papers due

Today

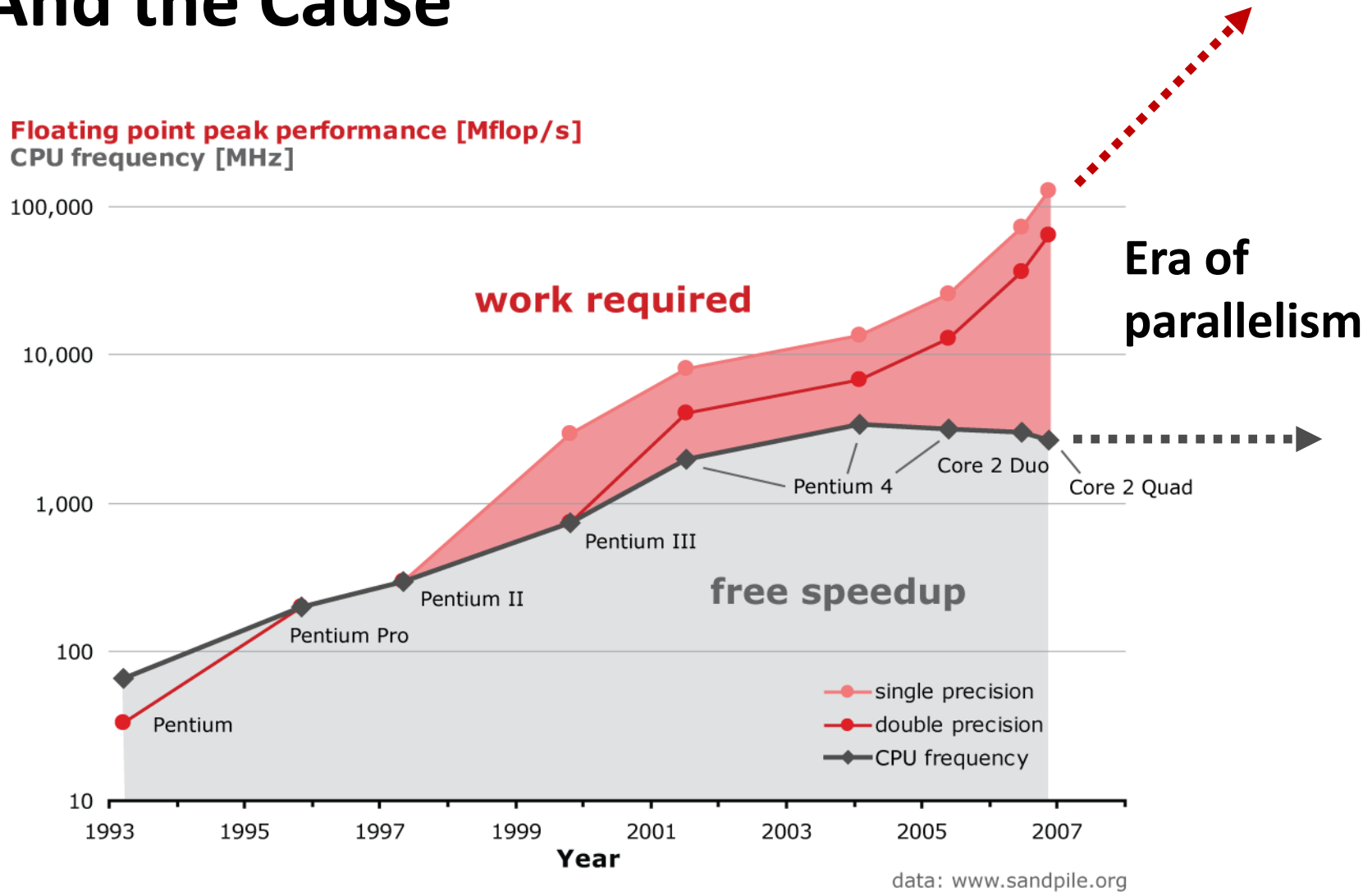
- **Spiral: Can we teach computers to write fast libraries?**

You Already Know the Problem …



- **A straightforward implementation of a numerical algorithm is typically *10-100x slower* than the fastest available**
- **Minimizing operations count ≠ minimizing runtime**
- **Developing high performance numerical code has become a nightmare**
  - Expertise in algorithms, programming, architecture
  - Optimization for the memory hierarchy, vectorization, parallelization

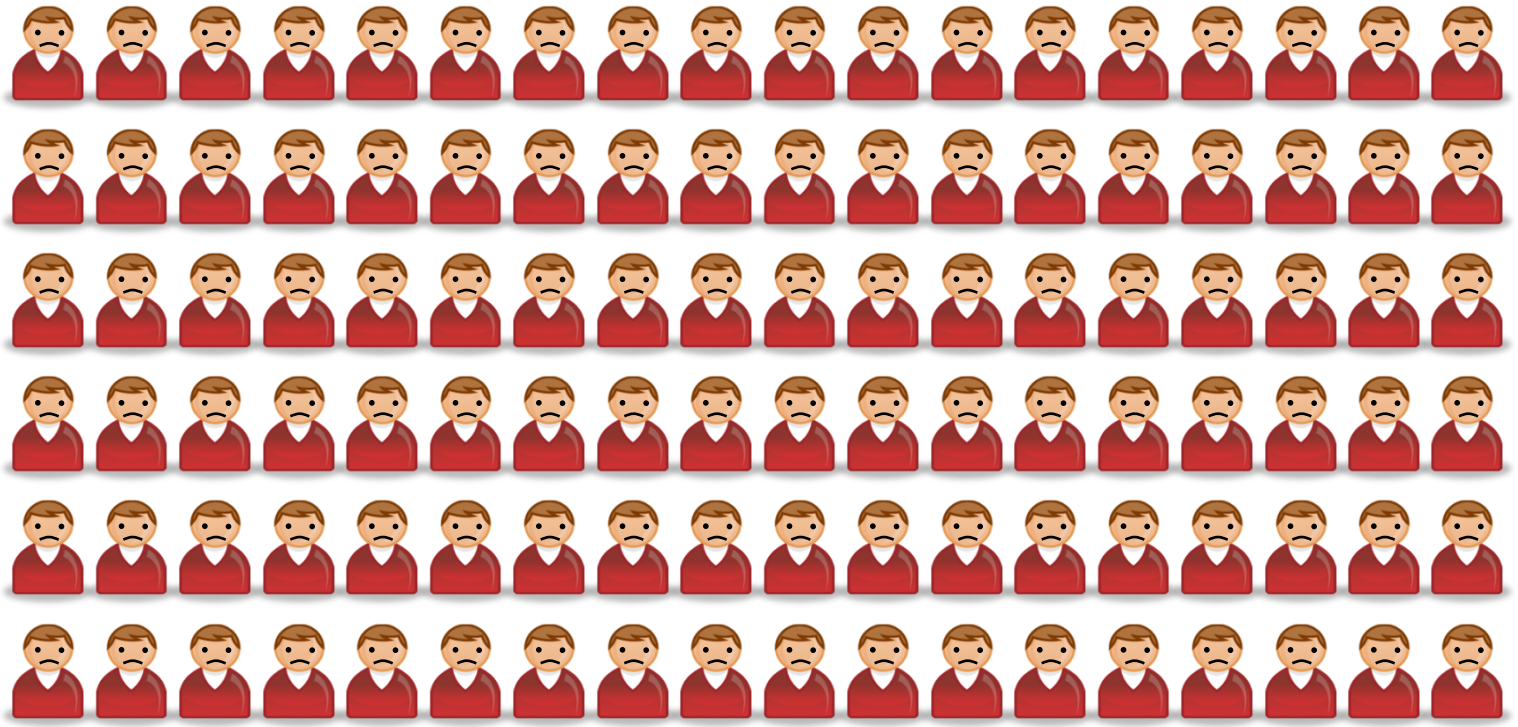
… And the Cause



High performance library development has become a nightmare

Current Solution

- **Legions of programmers implement and optimize the same functionality for every platform and whenever a new platform comes out.**



Better Solution: Automation

- **Automate (parts of) the implementation or optimization**

- **ATLAS:**
  - Automatic generation of BLAS
  - Optimization for the memory hierarchy
- **Sparsity/Bebop:**
  - Automatic platform adaptation of sparse MVM
  - Mainly register blocking
- **FFTW:**
  - Adaptive DFT library
  - Code generator for unrolled basic blocks
- **Spiral (today):**
  - Library generator for linear transforms
  - Complete automation (computer does everything)

www.spiral.net

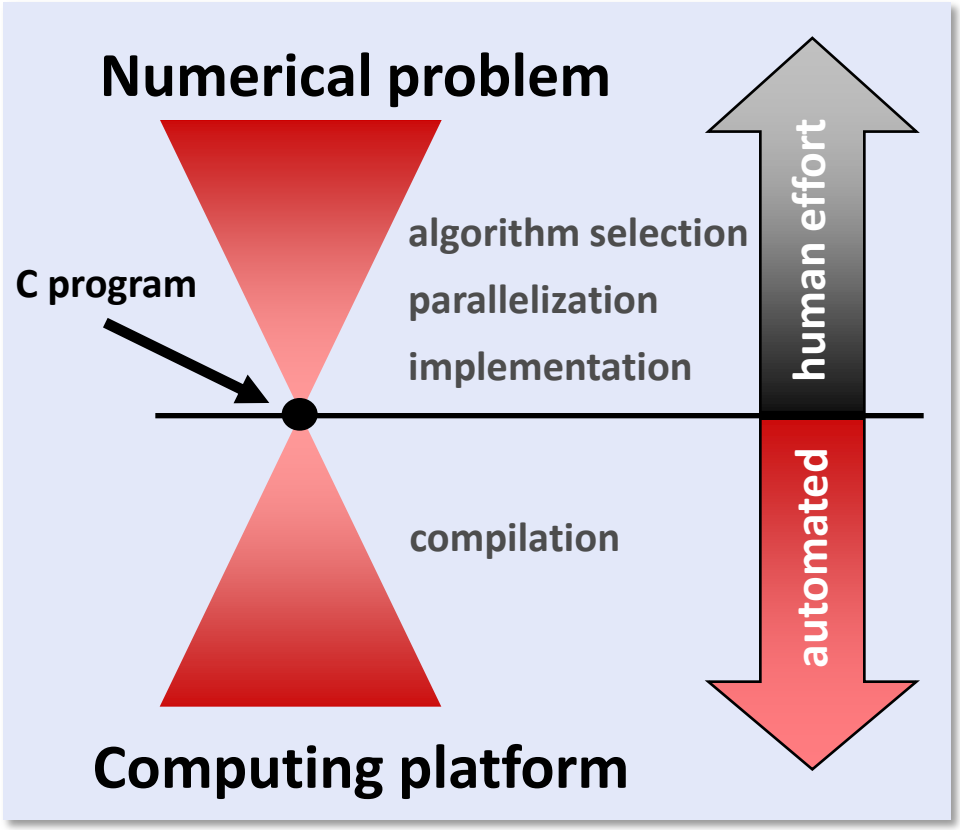
Organization

- **Spiral: Base system**
- Parallelism
- Libraries (general size)
- Beyond transforms
- Conclusions

Markus Püschel, 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 Nick Rizzolo, **SPIRAL: Code Generation for DSP Transforms,** Proceedings of the IEEE 93(2), 2005

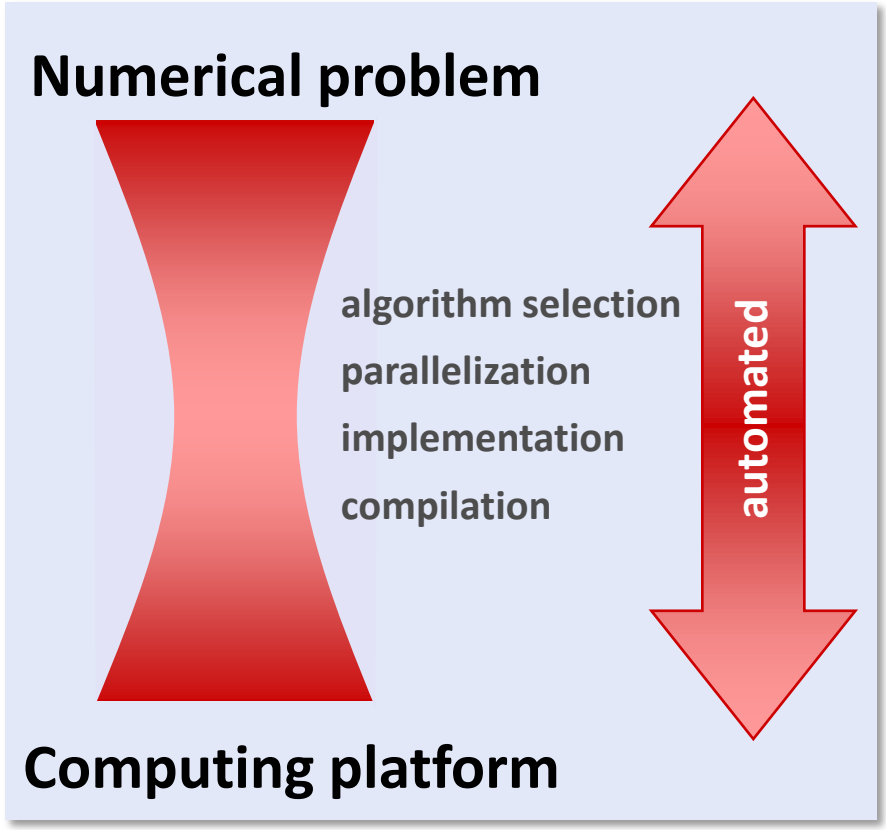
Vision Behind Spiral

Current



- C code a singularity: Compiler has no access to high level information

Future



- Challenge: conquer the high abstraction level for complete automation

Spiral

- **Library generator for linear transforms**
  (discrete Fourier/cosine/wavelet transforms, filters, ….) *and recently more …*

- **Wide range of platforms supported:**
  scalar, fixed point, vector, parallel, Verilog

- **Research Goal: "Teach" computers to write fast libraries**
  - Complete automation of implementation and optimization
  - Conquer the "high" algorithm level for automation

- **When a new platform comes out:**
  Regenerate a retuned library

- **When a new platform paradigm comes out (e.g., vector or CMPs):**
  Update the tool rather than rewriting the library

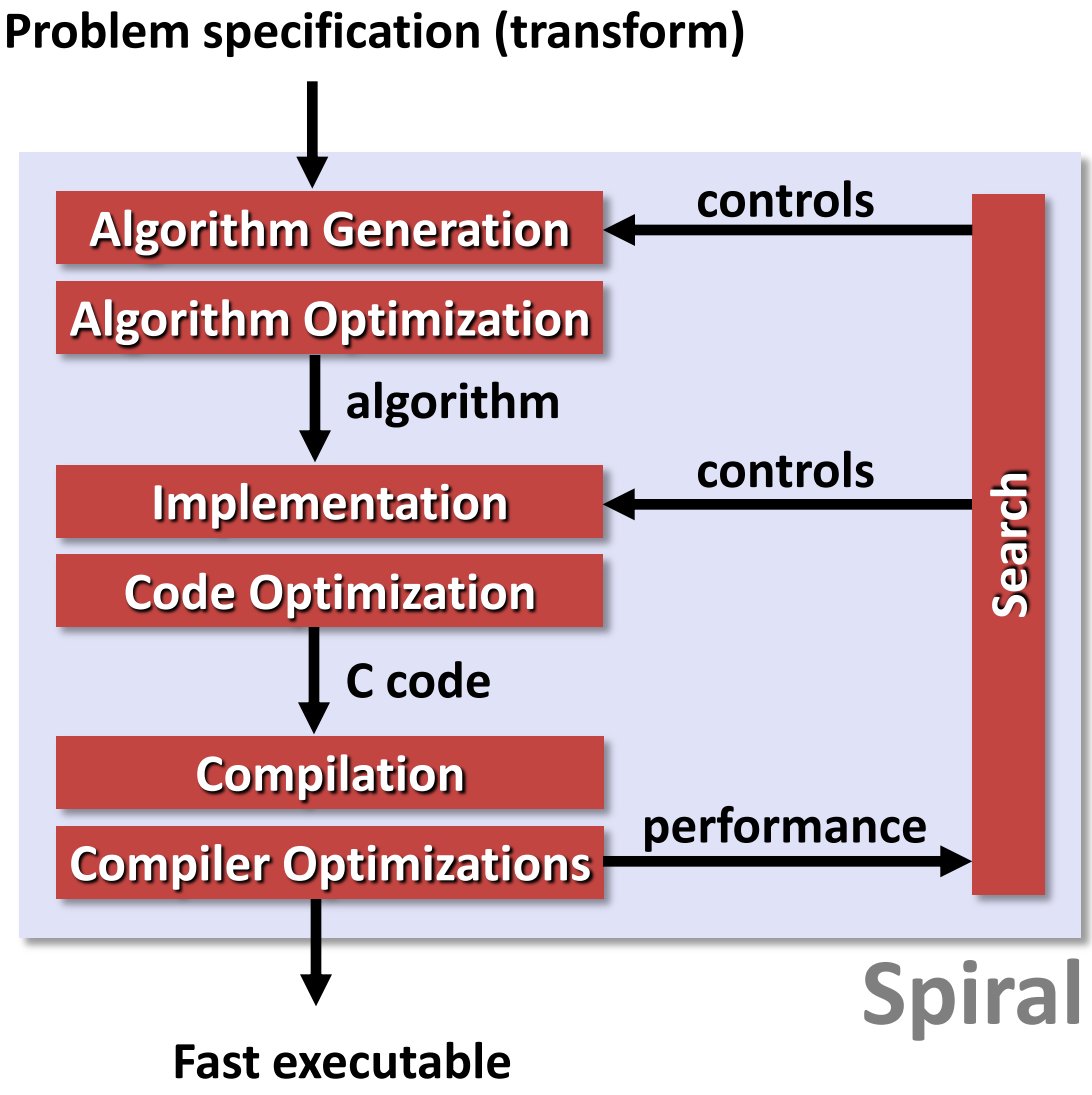
Intel has started to use Spiral
to generate parts of their MKL/IPP library

How Spiral Works

Spiral:
Complete automation of the implementation and optimization task

Key ideas:
Declarative, mathematical representation of algorithms

Rewriting systems to generate and optimize algorithms at a high level of abstraction



Web Interface (Beta Version, @spiral.net)

Program Generation Interface collapse

Target platform for optimization: 2x Intel Xeon 3.6 GHz with 2048K L2 cache

parameter	value	explanation
Transform	DCT2 (Discrete Cosine Transform, type 2)	The transform for which you want to request C code
Data type	double	The data type of input and output vector
Transform size	6	The size of the transform = the length of the input vector
Optimize for	runtime	What you want to optimize the code for
Search method	Dynamic Programming	The search method SPIRAL uses (Dynamic Programming is a good choice)
Compiler profile	gcc -O3	Compiler and compiler options used for compilation

Generate code

Browse Archive expand

Filter by Platform: All Platforms Selected

Filter by Transform: All Transforms Selected

Filter by Size: All Sizes Selected

Query Database

What is a (Linear) Transform?

- **Mathematically: Matrix-vector multiplication**

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

Output vector | **Transform = matrix** | **Input vector**

- **Example: Discrete Fourier transform (DFT)**

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

Transform Algorithms: Example 4-point FFT

Cooley/Tukey fast Fourier transform (FFT):

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

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

- Algorithms are divide-and-conquer: Breakdown rules
- Mathematical, declarative representation:
  SPL (signal processing language)
- SPL describes the structure of the dataflow

Examples: Transforms (currently ≈60)

$$
\begin{aligned}
\mathbf{DCT\text{-}2}_n &= \left[\cos(k(2\ell+1)\pi/2n)\right]_{0\le k,\ell<n},\\
\mathbf{DCT\text{-}3}_n &= \mathbf{DCT\text{-}2}_n^T \quad \text{(transpose)},\\
\mathbf{DCT\text{-}4}_n &= \left[\cos((2k+1)(2\ell+1)\pi/4n)\right]_{0\le k,\ell<n},\\
\mathbf{IMDCT}_n &= \left[\cos((2k+1)(2\ell+1+n)\pi/4n)\right]_{0\le k<2n,0\le \ell<n},\\
\mathbf{RDFT}_n &= [r_{k\ell}]_{0\le k,\ell<n},\quad r_{k\ell} = \begin{cases} \cos\frac{2\pi k\ell}{n}, & k \le \lfloor \frac{n}{2} \rfloor \\ -\sin\frac{2\pi k\ell}{n}, & k > \lfloor \frac{n}{2} \rfloor \end{cases},\\
\mathbf{WHT}_n &= \begin{bmatrix} \mathbf{WHT}_{n/2} & \mathbf{WHT}_{n/2} \\ \mathbf{WHT}_{n/2} & -\mathbf{WHT}_{n/2} \end{bmatrix}, \quad \mathbf{WHT}_2 = \mathbf{DFT}_2,\\
\mathbf{DHT} &= \left[\cos(2k\ell\pi/n) + \sin(2k\ell\pi/n)\right]_{0\le k,\ell<n}.
\end{aligned}
$$

Examples: Breakdown Rules (currently ≈220)

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

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

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

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

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

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

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

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

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

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

Base case rules

Combining these rules yields many algorithms for every given transform

Electrical & Computer ENGINEERING

Breakdown Rules (220)

BSkewDFT 4
- BSkewDFT_Base2
- BSkewDFT_Base4
- BSkewDFT_Fact
- BSkewDFT_Decomp

BSkewPRDFT 1
- BSkewPRDFT_Decomp

BSkewRDFT 1
- BSkewPRDFT_Decomp

Circulant 10
- RuleCirculant_Base
- RuleCirculant_toFilt
- RuleCirculant_Blocking
- RuleCirculant_BlockingDense

RuleCirculant_DiagonalizeStep
- RuleCirculant_DFT
- RuleCirculant_RDFT
- RuleCirculant_PRDFT
- RuleCirculant_PRDFT1
- RuleCirculant_DHT

CosDFT 2
- CosDFT_Base
- CosDFT_Trigonometric

DCT1 3
- DCT1_Base2
- DCT1_Base4
- DCT1_DCT1and3

DCT2 5
- DCT2_DCT2and4
- DCT2_toRDFT
- DCT2_toRDFT_odd
- DCT2_PrimePowerInduction
- DCT2_PrimeFactor

DCT3 5
- DCT3_Base2
- DCT3_Base3
- DCT3_Base5
- DCT3_Orth9
- DCT3_toSkewDCT3

DCT4 8
- DCT4_Base2
- DCT4_Base3
- DCT4_DCT2
- DCT4_DCT2t
- DCT4_DCT2andDST2
- DCT4_DST4andDST2
- DCT4_Iterative
- DCT4_BSkew_Decomp

DCT5 1
- DCT5_Rader

DFT 22
- DFT_Base
- DFT_Canonize
- DFT_CT
- DFT_CT_Mincost
- DFT_CT_DDL
- DFT_CosSinSplit
- DFT_Rader
- DFT_Bluestein
- DFT_GoodThomas
- DFT_PFA_SUMS
- DFT_PFA_RaderComb
- DFT_SplitRadix
- DFT_DCT1andDST1
- DFT_DFT1and3
- DFT_CT_Inplace

DRDFT 1
- DRDFT_tSPL_Pease

DSCirculant 2
- RuleDSCirculant_Base

DST1_Base2
DST1_DST3and1

DST2 3
- DST2_Base2
- DST2_toDCT2
- DST2_DST2and4

DST4 2
- DST4_Base
- DST4_toDCT4

DTT 15
- DTT_C3_Base2

Cooley-Tukey FFT

Filt 10
- RuleFilt_Base
- RuleFilt_Nest
- RuleFilt_OverlapSave

RuleFilt_OverlapSaveFreq
- RuleFilt_OverlapAdd
- RuleFilt_OverlapAdd2

RuleFilt_KaratsubaSimple
- RuleFilt_Circulant
- RuleFilt_Blocking
- RuleFilt_KaratsubaFast

IPRDFT 5
- IPRDFT1_Base1
- IPRDFT1_Base2
- IPRDFT1_CT
- IPRDFT1_Complex
- IPRDFT1_CT_Radix2

IPRDFT2 4
- IPRDFT2_Base1
- IPRDFT2_Base2
- IPRDFT2_CT

PDHT2 2
- PDHT2_Base2
- PDHT2_CT

PDHT3 4
- PDHT3_Base2
- PDHT3_CT
- PDHT3_Trig
- PDHT3_CT_Radix2

PDHT4 3
- PDHT4_Base2
- PDHT4_CT
- PDHT4_Trig

PDST4 2
- PDST4_Base2
- PDST4_CT

PRDFT 10
- PRDFT1_Base1
- PRDFT1_Base2
- PRDFT1_CT
- PRDFT1_Complex
- PRDFT1_Complex_T
- PRDFT1_Trig
- PRDFT1_PF

PolyDTT 4
- PolyDTT_ToNormal
- PolyDTT_Base2
- PolyDTT_SkewBase2
- PolyDTT_SkewDST3_CT

RDFT 4
- RDFT_Base
- RDFT_Trigonometric
- RDFT_CT_Radix2
- RDFT_toDCT2

RHT 2
- RHT_Base
- RHT_CooleyTukey

SinDFT 2
- SinDFT_Base
- SinDFT_Trigonometric

SkewDFT 2
- SkewDFT_Base2
- SkewDFT_Fact

SkewDTT 3
- SkewDTT_Base2
- SkewDCT3_VarSteidl

Breakdown rules in Spiral:

- "Teaches" Spiral about existing algorithm knowledge (≈200 journal papers)
- Includes many new ones (algebraic theory)

DFT4 3
- DFT4_Base
- DFT4_CT
- DFT4_PRDFT4

DFTBR 1
- DFTBR_HW

DHT 3
- DHT_Base
- DHT_DCT1andDST1
- DHT_Radix2

DWT_Base2
DWT_Base4
DWT_Base8
DWT_Lifting

DWTper 4
- DWTper_Mallat_2
- DWTper_Mallat
- DWTper_Polyphase
- DWTper_Lifting

MDDFT_Tensor
MDDFT_RowCol
MDDFT_Dimless
MDDFT_tSPL_RowCol

PDCT4 2
- PDCT4_Base2
- PDCT4_CT

PDHT 3
- PDHT1_Base2
- PDHT1_CT
- PDHT1_CT_Radix2

PRDFT4_Base1
PRDFT4_Base2
PRDFT4_CT
PRDFT4_Trig

PolyBDFT 5
- PolyBDFT_Base2
- PolyBDFT_Base4
- PolyBDFT_Fact
- PolyBDFT_Trig
- PolyBDFT_Decomp

WHT 8
- WHT_Base
- WHT_GeneralSplit
- WHT_BinSplit
- WHT_DDL
- WHT_Dirsum
- WHT_tSPL_BinSplit
- WHT_tSPL_STerm
- WHT_tSPL_Pease

SPL to Sequential Code

SPL construct	code
$y = (A_n B_n)x$	`t[0:1:n-1] = B(x[0:1:n-1]);` `y[0:1:n-1] = A(t[0:1:n-1];)`

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

$y = (A_m \otimes I_n)x$	`for (i=0;i<m;i++)`
	`  y[i:n:i+m-1] =`
	`  A(x[i:n:i+m-1]);`

$y = \left(\oplus_{i=0}^{m-1} A_n^i\right) x$	`for (i=0;i<m;i++)`
	`  y[i*n:1:i*n+n-1] =`
	`  A(i, x[i*n:1:i*n+n-1]);`

$y = D_{m,n}x$	`for (i=0;i<m*n;i++)` `  y[i] = Dmn[i]*x[i];`

$y = L_m^{mn}x$	`for (i=0;i<m;i++)`
	`  for (j=0;j<n;j++)`
	`  y[i+m*j]=x[n*i+j];`

Example: tensor product

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

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

Program Generation in Spiral (Sketched)

Transform
user specified

$$\mathrm{DFT}_8$$

Fast algorithm in SPL
many choices

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

∑-SPL:

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

C Code:

```
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;
  f8 = 0.3826834323650898 * f2;
  y[1] = f7 + f8;
  f10 = 0.3826834323650898 * f0;
  f11 = (-0.9238795325112867) * f2;
  y[3] = f10 + f11;
}
```

Optimization at all abstraction levels

- **parallelization vectorization**
- **loop optimizations**
- **constant folding scalar replacem. scheduling**
- **……**

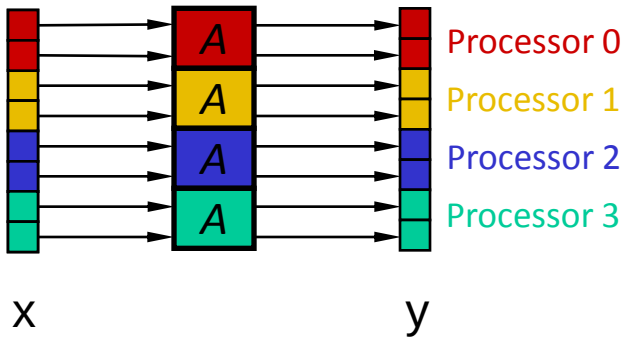
Organization

- Spiral: Base system
- **Parallelism**
- Libraries (general size)
- Beyond transforms
- Conclusions

SPL to Shared Memory Code: Basic Idea [SC 06]

- **Governing construct: tensor product**

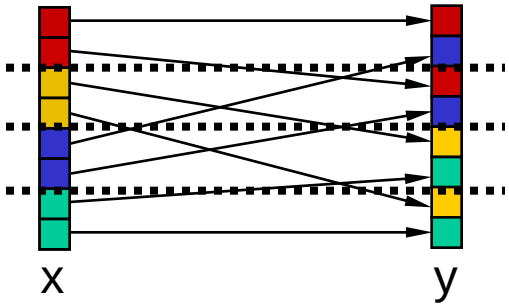
$$y = \left(\mathrm{I}_p \otimes A \right) x$$



p-way embarrassingly parallel, load-balanced

- **Problematic construct: permutations produce false sharing**

$$y = \mathrm{L}^{8}_{4}\, x$$



Task:* *Rewrite formulas to
extract tensor product + keep contiguous blocks

Step 1: Shared Memory Tags

- **Identify crucial hardware parameters**
  - Number of processors: p
  - Cache line size: μ

- **Introduce them as tags in SPL**

$$\overbrace{A}^{\mathsf{smp}(p,\mu)}$$

This means: formula A is to be optimized for p processors and cache line size μ

Step 2: Identify "Good" Formulas

- **Load balanced, avoiding false sharing**

$$y = \left(\mathrm{I}_p \otimes A\right) x \quad \text{with} \quad A \in \mathbb{C}^{m\mu \times m\mu}$$

$$y = \left(\bigoplus_{i=0}^{p-1} A_i\right) x \quad \text{with} \quad A_i \in \mathbb{C}^{m\mu \times m\mu}$$

$$y = \left(P \otimes \mathrm{I}_\mu\right) x \quad \text{with } P \text{ a permutation matrix}$$

Blackboard

- **Tagged operators (no further rewriting necessary)**

$$\mathrm{I}_p \otimes_{\|} A, \qquad \bigoplus_{i=0}^{p-1}{}_{\|} A_i, \qquad P \overline{\otimes} \mathrm{I}_\mu$$

- **Definition: A formula is fully optimized for (p, μ) if it is one of the above or of the form**

$$\mathrm{I}_m \otimes A \quad \text{or} \quad AB$$

where A and B are fully optimized.

Step 3: Identify Rewriting Rules

- **Goal: Transform formulas into fully optimized formulas**
  - Formulas rewritten, tags propagated
  - There may be choices

$$\underbrace{AB}_{\text{smp}(p,\mu)} \rightarrow \underbrace{A}_{\text{smp}(p,\mu)}\ \underbrace{B}_{\text{smp}(p,\mu)}$$

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

$$\underbrace{\mathrm{L}^{mn}_m}_{\text{smp}(p,\mu)} \rightarrow \begin{cases} \underbrace{\left(\mathrm{I}_p \otimes \mathrm{L}^{mn/p}_{m/p}\right)}_{\text{smp}(p,\mu)}\underbrace{\left(\mathrm{L}^{pn}_p \otimes \mathrm{I}_{m/p}\right)}_{\text{smp}(p,\mu)} \\ \underbrace{\left(\mathrm{L}^{pm}_m \otimes \mathrm{I}_{n/p}\right)}_{\text{smp}(p,\mu)}\underbrace{\left(\mathrm{I}_p \otimes \mathrm{L}^{mn/p}_m\right)}_{\text{smp}(p,\mu)} \end{cases}$$

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

$$\underbrace{(P \otimes \mathrm{I}_n)}_{\text{smp}(p,\mu)} \rightarrow \left(P \otimes \mathrm{I}_{n/\mu}\right) \overline{\otimes}\, \mathrm{I}_\mu$$

Simple Rewriting Example

$$A_m \otimes \mathrm{I}_n$$

Loop splitting + loop exchange

$$\left(\mathrm{L}^{mp}_{m} \otimes \mathrm{I}_{n/p}\right)\left(\mathrm{I}_p \otimes_{||} (A_m \otimes \mathrm{I}_{n/p})\right)\left(\mathrm{L}^{mp}_{p} \otimes \mathrm{I}_{n/p}\right)$$

fully optimized

```
parallel for (i=0; i<p; i++)
   for (j=0; j<n/p; j++)
      y[i*n/p+j:n:i*n/p+j+m-1] =
         A(x[i*n/p+j:n:i*n/p+j+m-1]);
```

Parallelization by Rewriting

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

$$\cdots$$

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

$$\cdots$$

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

Fully optimized (load-balanced, no false sharing) in the sense of our definition

Same Approach for Other Parallel Paradigms

Message Passing: [ISPA 06]

$$\underbrace{\mathrm{DFT}_{mn}}_{\mathrm{par}(p)} \rightarrow \underbrace{(\mathrm{DFT}_m \otimes \mathrm{I}_n)}_{\mathrm{par}(p\leftarrow q)} \underbrace{\mathsf{T}^{mn}_n}_{\mathrm{par}(q)} \underbrace{(\mathrm{I}_m \otimes \mathrm{DFT}_n)}_{\mathrm{par}(q)} \underbrace{\mathsf{L}^{mn}_m}_{\mathrm{par}(q\leftarrow p)}$$

$$\cdots$$

$$\rightarrow \underbrace{\left(\mathrm{I}_p \otimes \mathsf{L}^{mn/p}_{m/p}\right)}_{\mathrm{par}(p)} \underbrace{\left(\mathsf{L}^{p^2}_p \otimes \mathrm{I}_{mn/p^2}\right)}_{\mathrm{par}(p\leftarrow q)} \underbrace{\left(\mathrm{I}_q \otimes (\mathrm{I}_{p/q} \otimes \mathsf{L}^n_p \otimes \mathrm{I}_{m/p})\right)}_{\mathrm{par}(q)} \underbrace{\left(\mathrm{I}_q \otimes (\mathrm{I}_{n/q} \otimes \mathrm{DFT}_m)\right)}_{\mathrm{par}(q)}$$

$$\underbrace{\left(\mathrm{I}_q \otimes \mathsf{L}^{mn/q}_{m/q}\right)}_{\mathrm{par}(q)} \underbrace{\left(\mathsf{L}^{q^2}_q \otimes \mathrm{I}_{mn/q^2}\right)}_{\mathrm{par}(q)} \underbrace{\left(\mathrm{I}_q \otimes (\mathsf{L}^n_q \otimes \mathrm{I}_{m/q})\right)}_{\mathrm{par}(q)} \underbrace{\mathsf{T}^{mn}_n}_{\mathrm{par}(q)} \underbrace{\left(\mathrm{I}_q \otimes (\mathrm{I}_{m/q} \otimes \mathrm{DFT}_n)\right)}_{\mathrm{par}(q)}$$

$$\underbrace{\left(\mathrm{I}_q \otimes (\mathrm{I}_{p/q} \otimes \mathsf{L}^{mn/p}_{m/p})\right)}_{\mathrm{par}(q)} \underbrace{\left(\mathsf{L}^{p^2}_p \otimes \mathrm{I}_{mn/p^2}\right)}_{\mathrm{par}(q\leftarrow p)} \underbrace{\left(\mathrm{I}_p \otimes (\mathsf{L}^n_p \otimes \mathrm{I}_{m/p})\right)}_{\mathrm{par}(p)}$$

With Bonelli, Lorenz, Ueberhuber, TU Vienna

Vectorization: [IPDPS 02, VecPar 06]

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

$$\cdots$$

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

$$\cdots$$

$$\rightarrow (\mathrm{I}_{mn/\nu} \otimes \underbrace{\mathsf{L}^{2\nu}_{\nu}}_{\mathrm{sse}})(\overline{\mathrm{DFT}_m \otimes \mathrm{I}_{n/\nu}} \vec{\otimes} \mathrm{I}_\nu)\underbrace{\overleftarrow{\overline{(\mathsf{T}^{mn}_n)}}^{\nu}}_{\mathrm{sse}}$$

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

$$\left((\mathsf{L}^{mn}_m \otimes \mathrm{I}_2) \vec{\otimes} \mathrm{I}_\nu\right)(\mathrm{I}_{mn/\nu} \otimes \underbrace{\mathsf{L}^{2\nu}_2}_{\mathrm{sse}})$$

Cg/OpenGL for GPUs:

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

$$\cdots$$

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

$$(\mathsf{L}^{r^n/2}_r \vec{\otimes} \mathrm{I}_2)(\mathrm{I}_{r^{n-1}/2} \otimes_\times \underbrace{\mathsf{L}^{2r}_r}_{\mathrm{shd}(t,c)})(\mathsf{R}^{r^{n-1}}_r \vec{\otimes} \mathrm{I}_r)$$

With Shen, TU Denmark

Verilog for FPGAs: [DAC 05]

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

$$\cdots$$

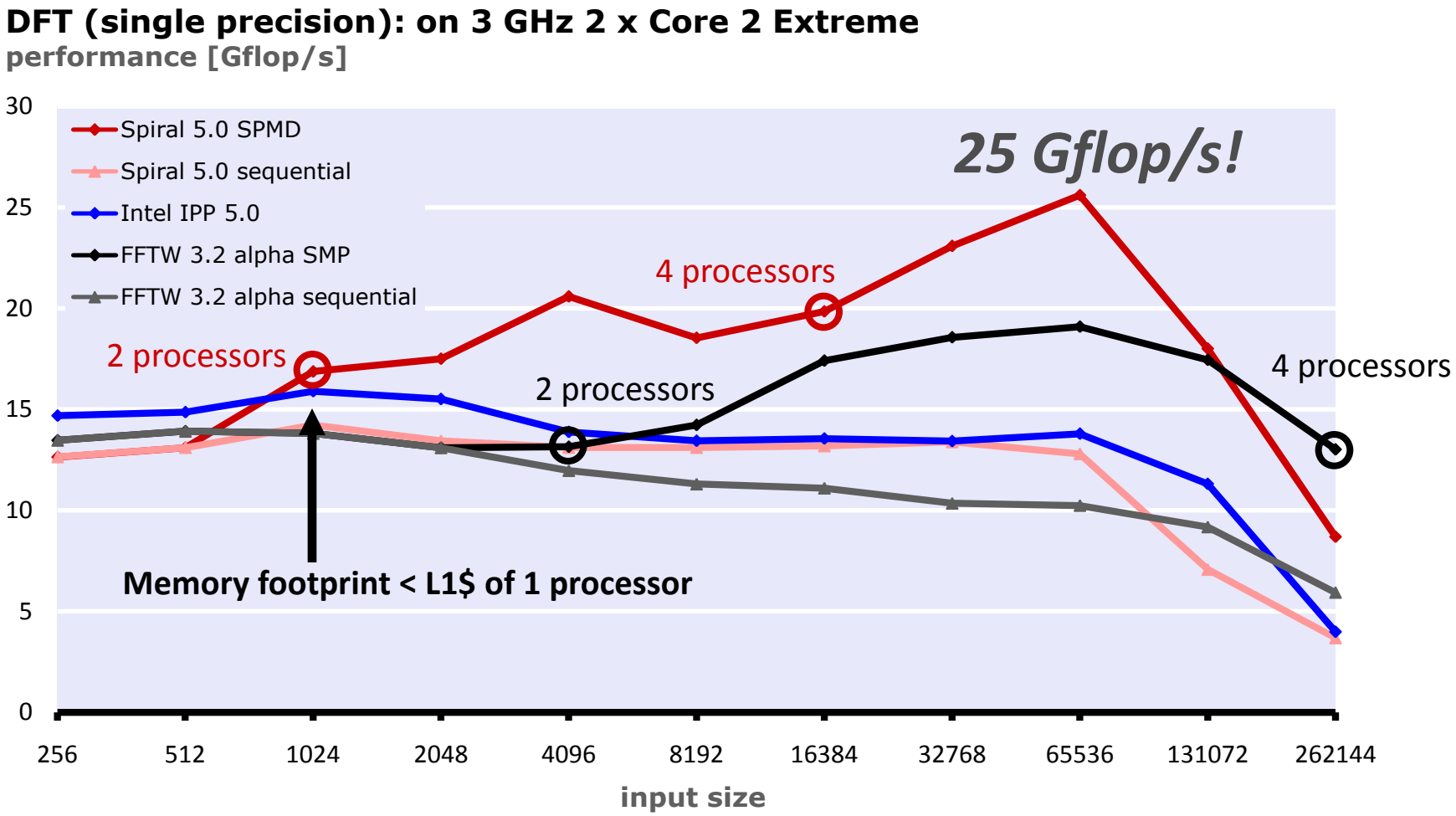
$$\rightarrow \left[\prod_{i=0}^{k-1} \underbrace{\mathsf{L}^{r^k}_r}_{\mathrm{stream}(r^s)} \underbrace{\left(\mathrm{I}_{r^{k-1}} \otimes \mathbf{DFT}_r\right)}_{\mathrm{stream}(r^s)} \underbrace{\left(\mathsf{L}^{r^k}_{r^{k-i-1}}(\mathrm{I}_{r^i} \otimes \mathsf{T}^{r^{k-i}}_{r^{k-i-1}})\,\mathsf{L}^{r^k}_{r^{i+1}}\right)}_{\mathrm{stream}(r^s)}\right]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

$$\cdots$$

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

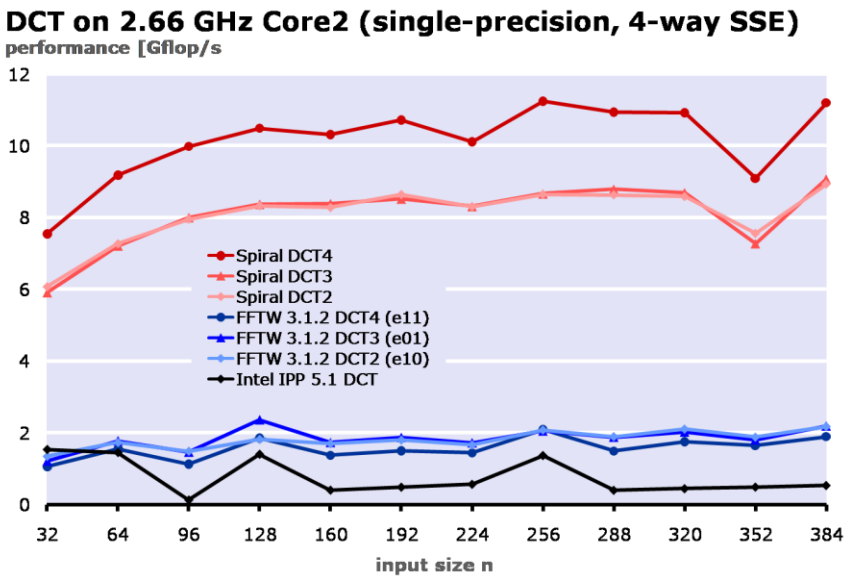
With Milder, Hoe, CMU

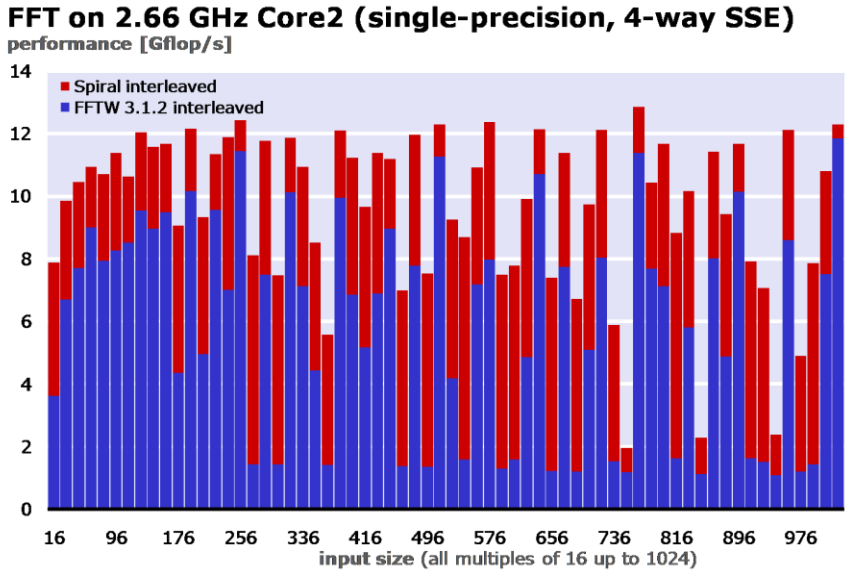
Benchmark: Vector and SMP



4-way vectorized + up to 4-threaded + adapted to the memory hierarchy

Other Transforms





Less popular transforms or sizes often means suboptimal code

Organization

- Spiral: Base system
- Parallelism
- **Libraries (general size)**
- Beyond transforms
- Conclusions

Ultimate Challenge: General Size Libraries

Spiral so far:

Fixed size code

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

- Algorithm = recursion fixed at code generation time

Challenge:

Recursive general size library (FFTW, MKL)

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

- Algorithm = recursion fixed at time of use
- Requires infrastructure (search, precomputing, base cases = codelets)
- Many other challenges

$$(\mathrm{DFT}_k \otimes \mathrm{I}_m)\, T^n_m (\mathrm{I}_k \otimes \mathrm{DFT}_m)\, L^n_k$$

Possible? → **Vectorized, parallel, general-size, adaptive library**

Challenge: Recursion Steps

- **Cooley-Tukey FFT**

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

- **Implementation in FFTW 2.x**

```
DFT(int n, complex *Y, complex *X) {
  k = choose_factor(n);

  for i=0 to k-1
    DFT_strided(n/k, k, 1, Y + (n/k)*i, T + (n/k)*i)

  for i=0 to n/k-1
    DFT_scaled(k, n/k, precomputed_f, Y + i, Y + i)
}

DFT_strided(int n, int is, int os, complex *Y, complex *X) {…}

DFT_scaled(int n, int s, complex *F, complex *Y, complex *X) {…}
```

2 additional functions (recursion steps) needed

How to discover automatically?

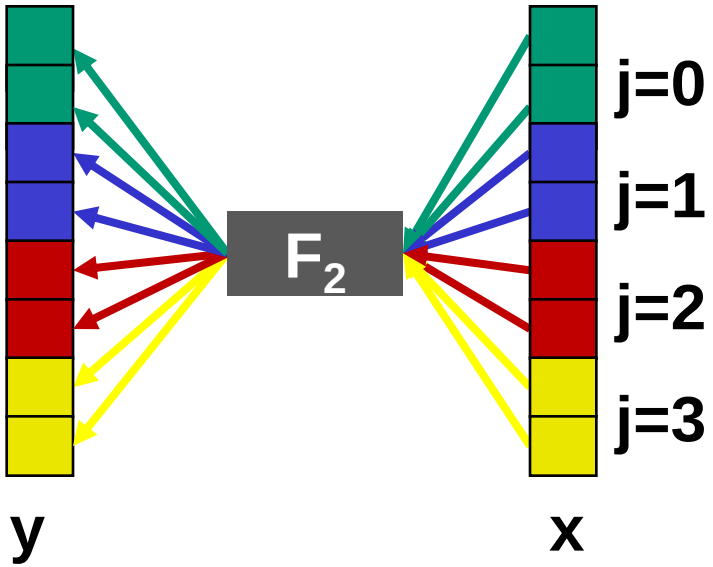
Σ–SPL : Basic Idea

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

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



Find Recursion Step Closure

- **Input:** transform T and a breakdown rule
- **Output:** recursion step closure + implementation of each recursion step
- **Algorithm:**

1. Apply the breakdown rule to T

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

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

2. Convert to Σ-SPL

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

3. Apply loop merging + index simplification rules.

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

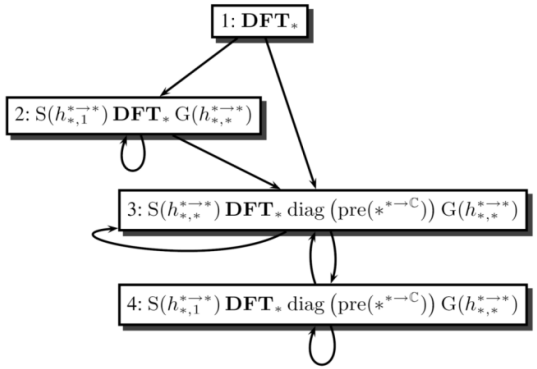
4. Extract required recursion steps

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

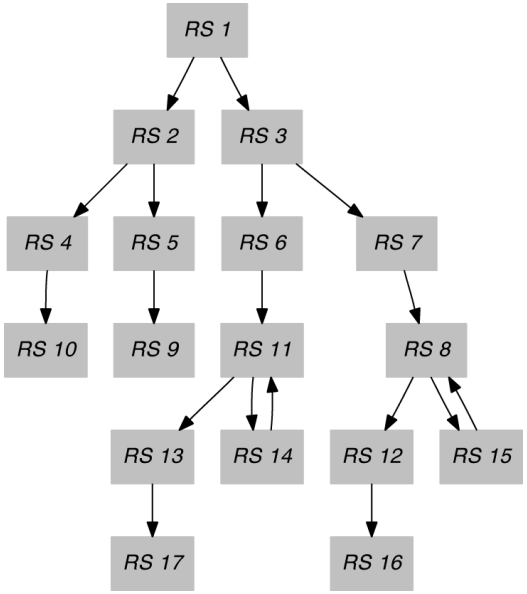
5. Repeat until closure is reached

Recursion Step Closure: Example

DFT: scalar code



DCT4: vector code



1 : $\mathrm{Vec}_2(\mathbf{DCT\text{-}4}_{u_1})$

2 : $\mathrm{Vec}_2(\mathrm{GT}(\mathrm{diag}\,(N_{2u_8})\,\mathbf{RDFT\text{-}3}^{\top}_{2u_8}\,\mathrm{rcdiag}(\mathrm{pre}(u_4{}^{\mathbb{Z}\times 2u_8\to\mathbb{R}})),\ h^{2u_8\to u_6}_{0,\,1,u_7}\circ \ell^{2u_8}_{u_8},\ r^{2u_8\to u_9}_{0,\,u_{11},1,\,u_{12}},\ \{u_{13}\}))$

3 : $\mathrm{Vec}_2(\mathrm{GT}(\mathbf{RDFT\text{-}3}_{u_1}\,\mathrm{diag}\,(N_{u_1}),\ r^{u_1\to u_3}_{0,\,u_5,1,\,u_6},\ h^{u_1\to u_8}_{0,\,u_9,1},\ \{u_{10}\}))$

4 : $\mathrm{VJam}_2(\mathrm{GT}(\mathrm{diag}\,(N_{2u_9})\,\mathbf{RDFT\text{-}3}^{\top}_{2u_9}\,\mathrm{rcdiag}(\mathrm{pre}(u_4{}^{\mathbb{Z}\times\mathbb{Z}\times 2u_9\to\mathbb{R}})),\ h^{2u_9\to u_6}_{0,\,1,u_7,u_8}\circ \ell^{2u_9}_{u_9},\ r^{2u_9\to u_{10}}_{0,\,u_{12},1,2,\,u_{13}},\ \{2,u_{14}\}))$

5 : $\mathrm{GT}(\mathrm{diag}\,(N_{2u_9})\,\mathbf{RDFT\text{-}3}^{\top}_{2u_9}\,\mathrm{rcdiag}(\mathrm{pre}(u_4{}^{\mathbb{Z}\times 2u_9\to\mathbb{R}})),\ h^{2u_9\to u_6}_{u_7,\,1,u_8}\circ \ell^{2u_9}_{u_9},\ r^{2u_9\to u_{10}}_{u_{12},\,u_{13},1,\,u_{14}},\ \{u_{15}\})$

6 : $\mathrm{VJam}_2(\mathrm{GT}(\mathbf{RDFT\text{-}3}_{u_1}\,\mathrm{diag}\,(N_{u_1}),\ r^{u_1\to u_3}_{0,\,u_5,1,2,\,u_6},\ h^{u_1\to u_8}_{0,\,u_9,1,2},\ \{2,u_{10}\}))$

7 : $\mathrm{GT}(\mathbf{RDFT\text{-}3}_{u_1}\,\mathrm{diag}\,(N_{u_1}),\ r^{u_1\to u_3}_{u_5,\,u_6,1,\,u_7},\ h^{u_1\to u_9}_{u_{10},\,u_{11},1},\ \{u_{12}\})$

8 : $\mathrm{S}(h^{u_1\to u_2}_{u_3,\,u_4})\,\mathbf{RDFT\text{-}3}_{u_1}\,\mathrm{diag}\,(N_{u_1})\,\mathrm{G}(r^{u_1\to u_7}_{u_9,\,u_{10},\,u_{11}})$

9 : $\mathrm{S}(r^{2u_{13}\to u_1}_{u_3,\,u_4,\,u_5})\mathrm{diag}\,(N_{2u_{13}})\,\mathbf{RDFT\text{-}3}^{\top}_{2u_{13}}\,\mathrm{rcdiag}(\mathrm{pre}(u_9{}^{2u_{13}\to\mathbb{R}}))\mathrm{G}(h^{2u_{13}\to u_{11}}_{u_{12},\,1}\circ \ell^{2u_{13}}_{u_{13}})$

10 : $\mathrm{VJam}_2(\mathrm{GT}(\mathrm{diag}\,(N_{2u_9})\,\mathbf{RDFT\text{-}3}^{\top}_{2u_9}\,\mathrm{rcdiag}(\mathrm{pre}(u_4{}^{\mathbb{Z}\times 2u_9\to\mathbb{R}})),\ h^{2u_9\to u_6}_{u_7,\,1,u_8}\circ \ell^{2u_9}_{u_9},\ r^{2u_9\to u_{10}}_{u_{12},\,u_{13},1,\,u_{14}},\ \{2\}))$

11 : $\mathrm{VJam}_2(\mathrm{GT}(\mathbf{RDFT\text{-}3}_{u_1}\,\mathrm{diag}\,(N_{u_1}),\ r^{u_1\to u_3}_{u_5,\,u_6,1,\,u_7},\ h^{u_1\to u_9}_{u_{10},\,u_{11},1},\ \{2\}))$

12 : $\mathrm{GT}(\mathrm{diag}\,(C_{u_1})\,\mathbf{rDFT}_{2u_1}(\lambda\text{-wrap}(\lambda_1{}^{\mathbb{Z}\times\mathbb{Z}\to\mathbb{R}})),\ h^{2u_1\to u_4}_{0,\,1,u_5},\ h^{2u_{10}\to u_7}_{u_8,\,u_9}\circ (r^{u_1\to u_{10}}_{0,\,u_{12},1,\,u_{13}}\otimes \imath_2),\ \{u_{14}\})$

13 : $\mathrm{VJam}_2(\mathrm{GT}(\mathrm{diag}\,(C_{u_1})\,\mathbf{rDFT}_{2u_1}(\lambda\text{-wrap}(\lambda_1{}^{\mathbb{Z}\times\mathbb{Z}\times\mathbb{Z}\to\mathbb{R}})),\ h^{2u_1\to u_4}_{u_5,\,u_6,1,u_7},\ h^{2u_{12}\to u_9}_{u_{10},\,u_{11},1}\circ (r^{u_1\to u_{12}}_{0,\,u_{14},0,1,\,u_{15}}\otimes \imath_2),\ \{2,u_{16}\}))$

14 : $\mathrm{VJam}_2(\mathrm{GT}(\mathbf{RDFT\text{-}3}_{u_1}\,\mathrm{diag}\,(N_{u_1}),\ r^{u_1\to u_3}_{u_5,\,u_6,1,u_7,\,u_8},\ h^{u_1\to u_{10}}_{u_{11},\,u_{12},1,u_{13}},\ \{2,u_{14}\}))$

15 : $\mathrm{GT}(\mathbf{RDFT\text{-}3}_{u_1}\,\mathrm{diag}\,(N_{u_1}),\ r^{u_1\to u_3}_{u_5,\,u_6,u_7,\,u_8},\ h^{u_1\to u_{10}}_{0,\,u_{11},1},\ \{u_{12}\})$

16 : $\mathrm{S}(h^{2u_5\to u_2}_{u_3,\,u_4}\circ (r^{u_6\to u_5}_{u_7,\,u_8,\,u_9}\otimes \imath_2))\mathrm{diag}\,(C_{u_6})\,\mathbf{rDFT}_{2u_6}(\lambda\text{-wrap}(\lambda_1{}^{\mathbb{Z}\to\mathbb{R}}))\mathrm{G}(h^{2u_6\to u_{13}}_{u_{14},\,1})$

17 : $\mathrm{VJam}_2(\mathrm{GT}(\mathrm{diag}\,(C_{u_1})\,\mathbf{rDFT}_{2u_1}(\lambda\text{-wrap}(\lambda_1{}^{\mathbb{Z}\times\mathbb{Z}\to\mathbb{R}})),\ h^{2u_1\to u_4}_{u_5,\,u_6,1},\ h^{2u_{11}\to u_8}_{u_9,\,u_{10},1}\circ (r^{u_1\to u_{11}}_{u_{13},\,u_{14},\,u_{15}}\otimes \imath_2),\ \{2\}))$

Generated Full Libraries

Input: breakdown rules (a few are not shown)

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

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

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

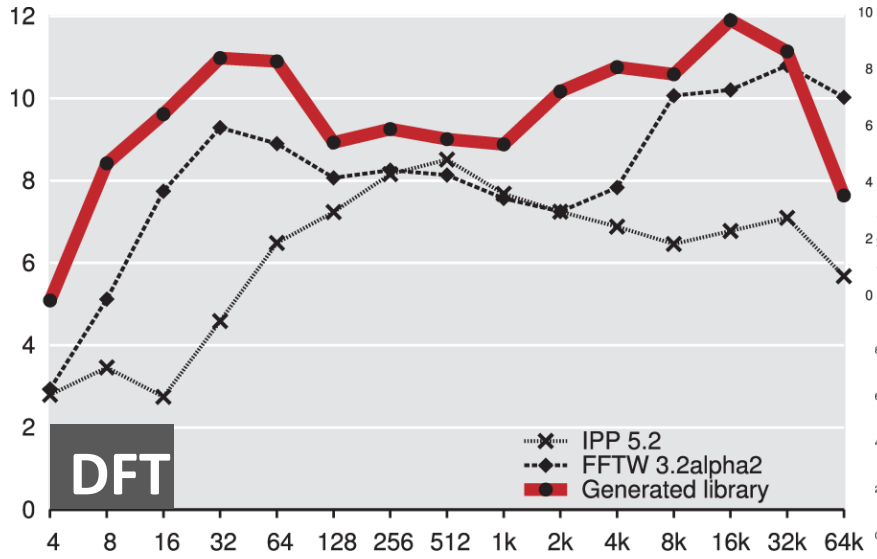
$$\mathbf{RDFT\text{-}3}_n = (Q_{k/2,m}^{\top} \otimes I_2) \left(I_k \otimes_i \mathbf{rDFT}_{2m})(i+1/2)/k)\right) \left(\mathbf{RDFT\text{-}3}_k \otimes I_m\right), \quad k \text{ even},$$

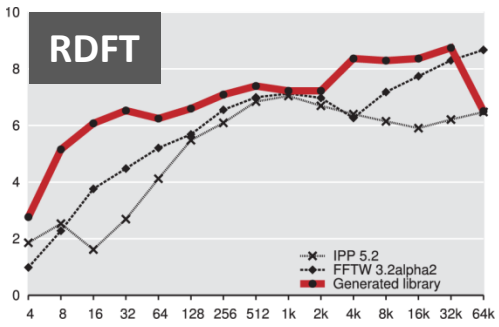
$$\mathbf{DCT\text{-}2}_n = P_{k/2,2m}^{\top} \left(\mathbf{DCT\text{-}2}_{2m}\, K_2^{2m} \oplus \left(I_{k/2-1} \otimes N_{2m}\, \mathbf{RDFT\text{-}3}_{2m}^{\top}\right)\right) B_n (L_{k/2}^{n/2} \otimes I_2)(I_m \otimes \mathbf{RDFT}'_k) Q_{m/2,k},$$

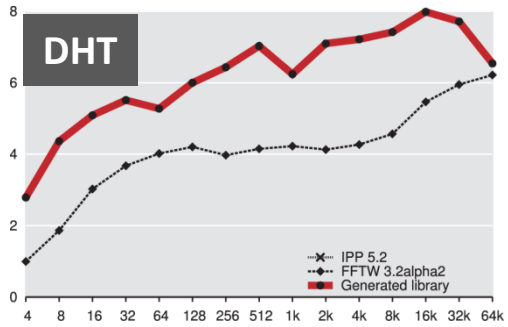
$$\mathbf{DCT\text{-}3}_n = \mathbf{DCT\text{-}2}_n^{\top},$$

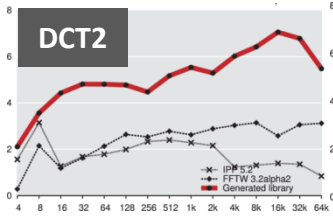
$$\mathbf{DCT\text{-}4}_n = Q_{k/2,2m}^{\top} \left(I_{k/2} \otimes N_{2m}\, \mathbf{RDFT\text{-}3}_{2m}^{\top}\right) B'_n (L_{k/2}^{n/2} \otimes I_2)(I_m \otimes \mathbf{RDFT\text{-}3}_k) Q_{m/2,k}.$$

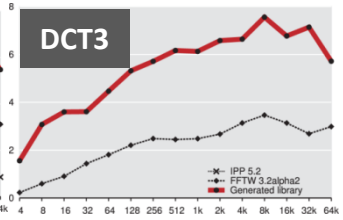
Generated Full Libraries

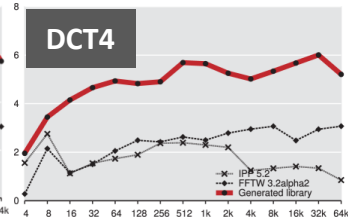




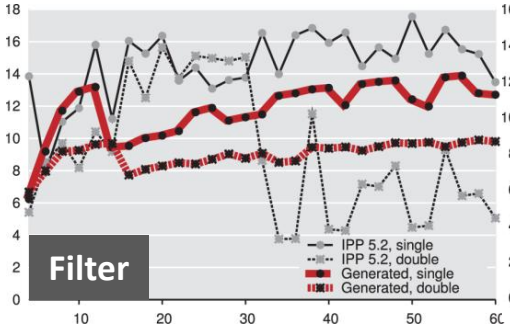


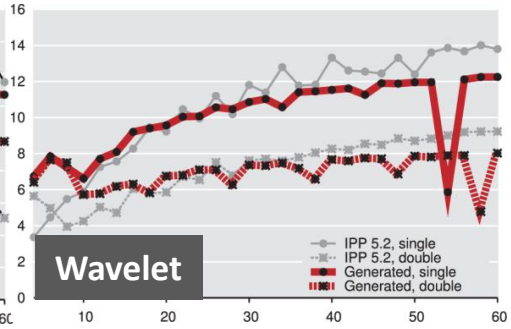






- 2-way vectorized, 2-threaded
- Most are faster than hand-written libs
- Recursion steps: 4–17
- Code size: 8–120 kloc or 0.5–5 MB
- Generation time: 1–3 hours





Total: 300 kloc or 13.3 MB of code generated in < 20 hours

from a few breakdown rules

Organization

- Spiral: Base system
- Parallelism
- Libraries (general size)
- **Beyond transforms**
- Conclusions

Going Beyond Transforms

- **Transform =**
  linear operator with one vector input and one vector output

- **Key ideas:**
  - Generalize to (possibly nonlinear) operators with several inputs and several outputs
  - Generalize SPL (including tensor product) to OL (operator language)
  - Generalize rewriting systems for parallelizations

Basic operators

name	definition
stride	$L^{mn}_n : \mathbb{R}^{mn} \to \mathbb{R}^{mn};\ \mathbf{M} \mapsto \mathbf{M}^\mathsf{T}$
vector sum	$\Sigma_m : \mathbb{R}^m \to \mathbb{R};\ \mathbf{x} \mapsto \sum_{i=0}^{m-1} x_i$
point-wise multiplication	$(\cdot)_m : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^m;\ (\mathbf{x}, \mathbf{y}) \mapsto (x_0 y_0,\ \ldots,\ x_{m-1} y_{m-1})$
Kronecker product	$(\otimes)_{m \times n} : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^{mn};\ (\mathbf{x}, \mathbf{y}) \mapsto x_0 \mathbf{y} \oplus \ldots \oplus x_{m-1} \mathbf{y}$

Operations (higher-order operators)

name	definition
composition	$(\mathcal{B} \circ \mathcal{A})(\mathbf{x}, \mathbf{y}) = \mathcal{B}(\mathcal{A}(\mathbf{x}, \mathbf{y}))$
Cartesian product	$(\mathcal{A} \times \mathcal{B})(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{t}) = \mathcal{A}(\mathbf{x}, \mathbf{y}) \times \mathcal{B}(\mathbf{z}, \mathbf{t})$
tensor product	$(\mathcal{A} \otimes \mathcal{B})(\mathbf{x}, \mathbf{y}) = \sum_{0 \le i < p, 0 \le j < r} \mathcal{A}(\mathbf{e}_{p,i}, \mathbf{e}_{r,j}) \otimes \mathcal{B}(\mathbf{x}^i, \mathbf{y}^j)$

Example: MMM

Breakdown rules capture various forms of blocking:

breakdown rule	description
$\mathrm{MMM}_{1,1,1} \rightarrow (\cdot)_1$	base case
$\mathrm{MMM}_{m,n,k} \rightarrow (\otimes)_{m/m_b \times 1} \otimes \mathrm{MMM}_{m_b,n,k}$	horizontal blocking
$\mathrm{MMM}_{m,n,k} \rightarrow \mathrm{MMM}_{m,nb,k} \otimes (\otimes)_{1 \times n/nb}$	interleaved blocking
$\mathrm{MMM}_{m,n,k} \rightarrow ((\Sigma_{k/k_b} \circ (\cdot)_{k/k_b}) \otimes \mathrm{MMM}_{m,n,k_b}) \circ ((L^{mk/k_b}_{k/k_b} \otimes I_{k_b}) \times I_{kn})$	accumulative blocking
$\mathrm{MMM}_{m,n,k} \rightarrow (L^{mn/n_b}_{m} \otimes I_{n_b}) \circ ((\otimes)_{1 \times n/n_b} \otimes \mathrm{MMM}_{m,n_b,k}) \circ (I_{km} \times (L^{kn/n_b}_{n/n_b} \otimes I_{n_b}))$	vertical blocking

Parallelization through rewriting

$$\underbrace{\mathrm{MMM}_{m,n,k}}_{\mathrm{smp}(p,\mu)}$$

$$\rightarrow \underbrace{\left(\mathrm{I}_m \otimes \mathrm{L}^n_p\right) \circ \left(\mathrm{MMM}_{m,n/p,k} \otimes (\otimes)_{1 \times p \rightarrow p}\right) \circ \left(\mathrm{I}_{km} \times (\mathrm{I}_k \otimes \mathrm{L}^n_{n/p})\right)}_{\mathrm{smp}(p,\mu)}$$

$$\rightarrow \underbrace{\left(\mathrm{I}_m \otimes \mathrm{L}^n_p\right)}_{\mathrm{smp}(p,\mu)} \circ \underbrace{\left(\mathrm{MMM}_{m,n/p,k} \otimes (\otimes)_{1 \times p \rightarrow p}\right)}_{\mathrm{smp}(p,\mu)} \circ \underbrace{\left(\mathrm{I}_{km} \times (\mathrm{I}_k \otimes \mathrm{L}^n_{n/p})\right)}_{\mathrm{smp}(p,\mu)}$$

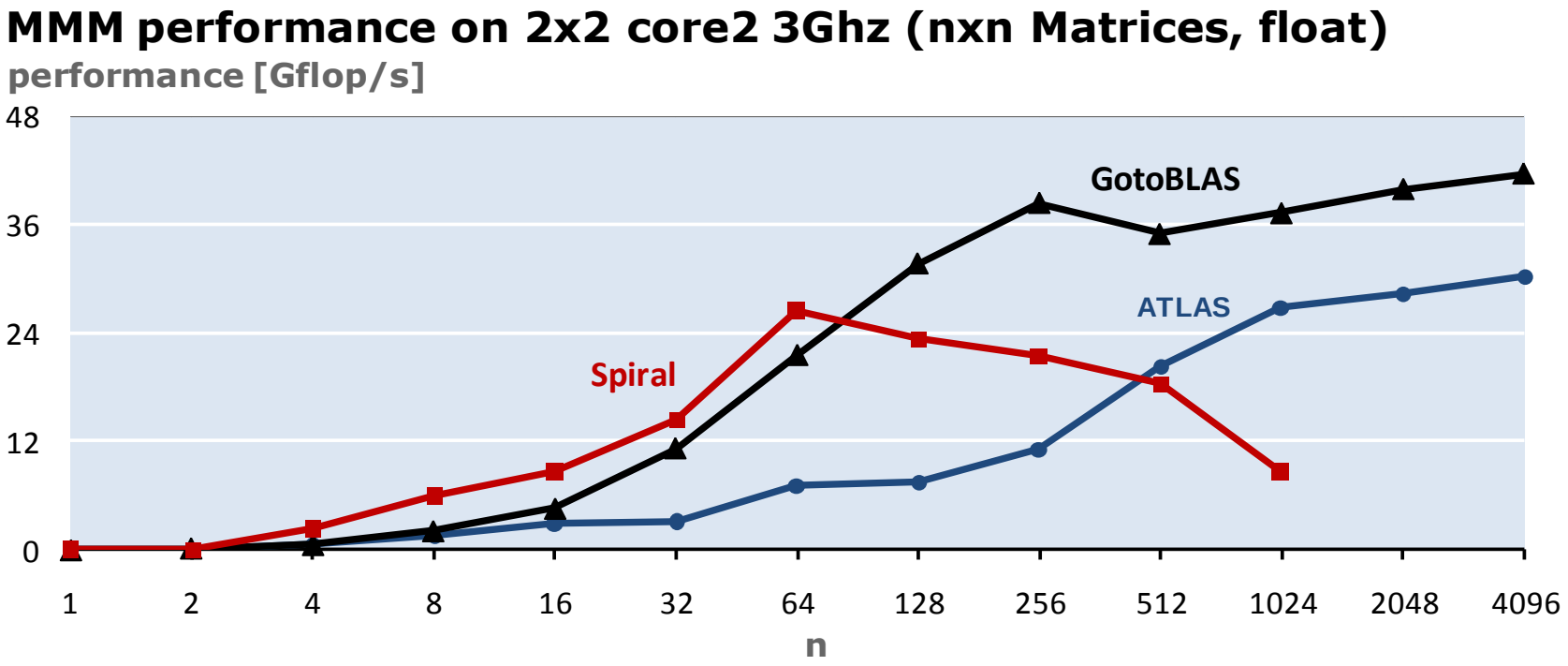
$$\rightarrow \underbrace{\left(\mathrm{I}_m \otimes \mathrm{L}^n_p\right)}_{\mathrm{smp}(p,\mu)} \circ \underbrace{\mathrm{L}^{mn}_{m/pn}}_{\mathrm{smp}(p,\mu)} \circ \left((\otimes)_{1 \times p \rightarrow p} \otimes_{\|} \mathrm{MMM}_{m/p,n,k}\right) \circ \underbrace{\left(\mathrm{I}_{km} \times \mathrm{L}^{kn}_p\right)}_{\mathrm{smp}(p,\mu)} \circ \underbrace{\left(\mathrm{I}_{km} \times (\mathrm{I}_k \otimes \mathrm{L}^n_{n/p})\right)}_{\mathrm{smp}(p,\mu)}$$

$$\rightarrow \left((\mathrm{L}^{mp}_m \otimes \mathrm{I}_{n/(p\mu)}) \bar{\otimes} \mathrm{I}_\mu\right) \circ \left((\otimes)_{1 \times p \rightarrow p} \otimes_{\|} \mathrm{MMM}_{m,n/p,k}\right) \circ \left((\mathrm{I}_{km/\mu} \bar{\otimes} \mathrm{I}_\mu) \times ((\mathrm{L}^{kp}_p \otimes \mathrm{I}_{n/(p\mu)}) \bar{\otimes} \mathrm{I}_\mu)\right)$$

Load-balanced

No false sharing

First Results



- **Other numerical problems we currently study**
  - SAR imaging
  - Viterbi decoding
  - JPEG 2000 decoding

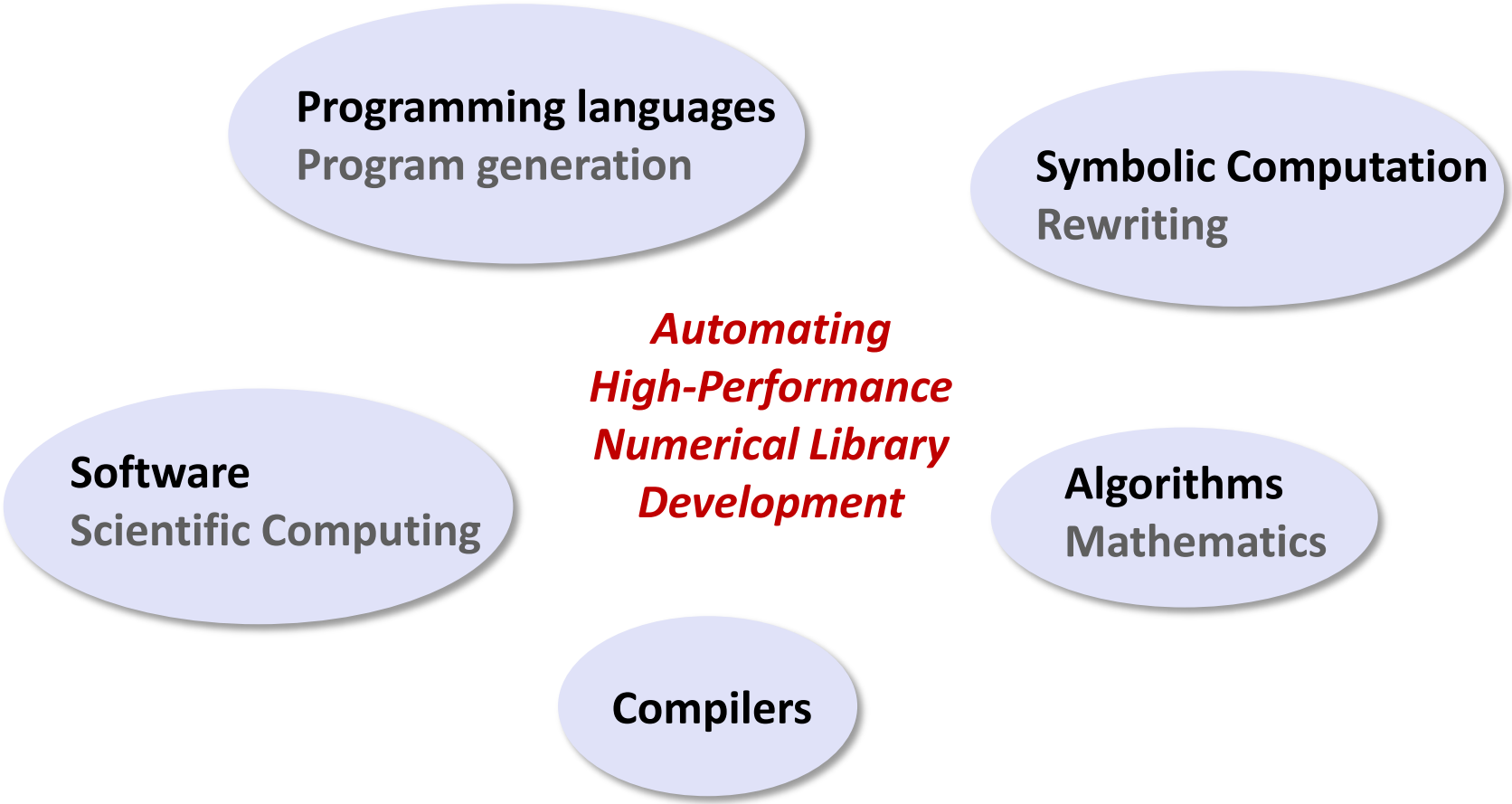
Organization

- Spiral: Base system
- Parallelism
- Libraries (general size)
- Beyond transforms
- **Conclusions**

Summary

- **Spiral: The computer writes fast transform libraries**
  - Complete automation
  - Including memory hierarchy optimizations, vectorization, parallelization

- **What we have learned**
  - Declarative representation of algorithms (mathematical domain-specific language)
  - Optimization at a high, “right” level of abstraction using rewriting
  - It makes sense to use math to represent and optimize math functionality
  - It makes sense to “teach” the computer algorithms and math (does not become obsolete)
  - Domain-specific is necessary to get fastest code
  - One needs techniques from different disciplines …..


Programming languages
Program generation
Symbolic Computation
Rewriting
Automating
High-Performance
Numerical Library
Development
Software
Scientific Computing
Algorithms
Mathematics
Compilers