Can We Teach Computers
To Write Fast Libraries?

Markus Püschel

… and the Spiral team (only part shown)

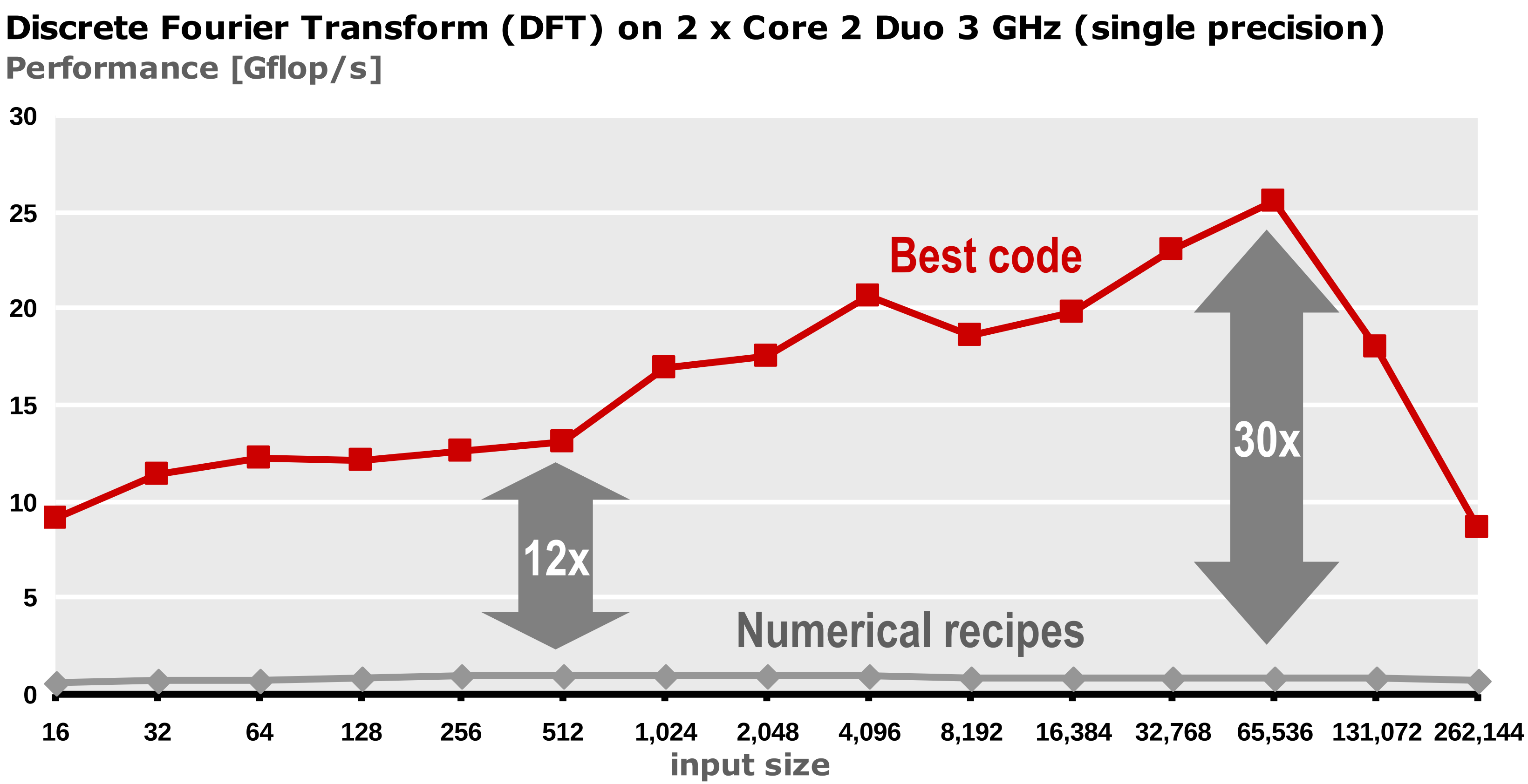
Collaborators:
Franz Franchetti
Yevgen Voronenko

Electrical and
Computer Engineering
Carnegie Mellon University

This work was supported by
DARPA DESA program, NSF-NGS/ITR, NSF-ACR, NSF-CPA, and Intel

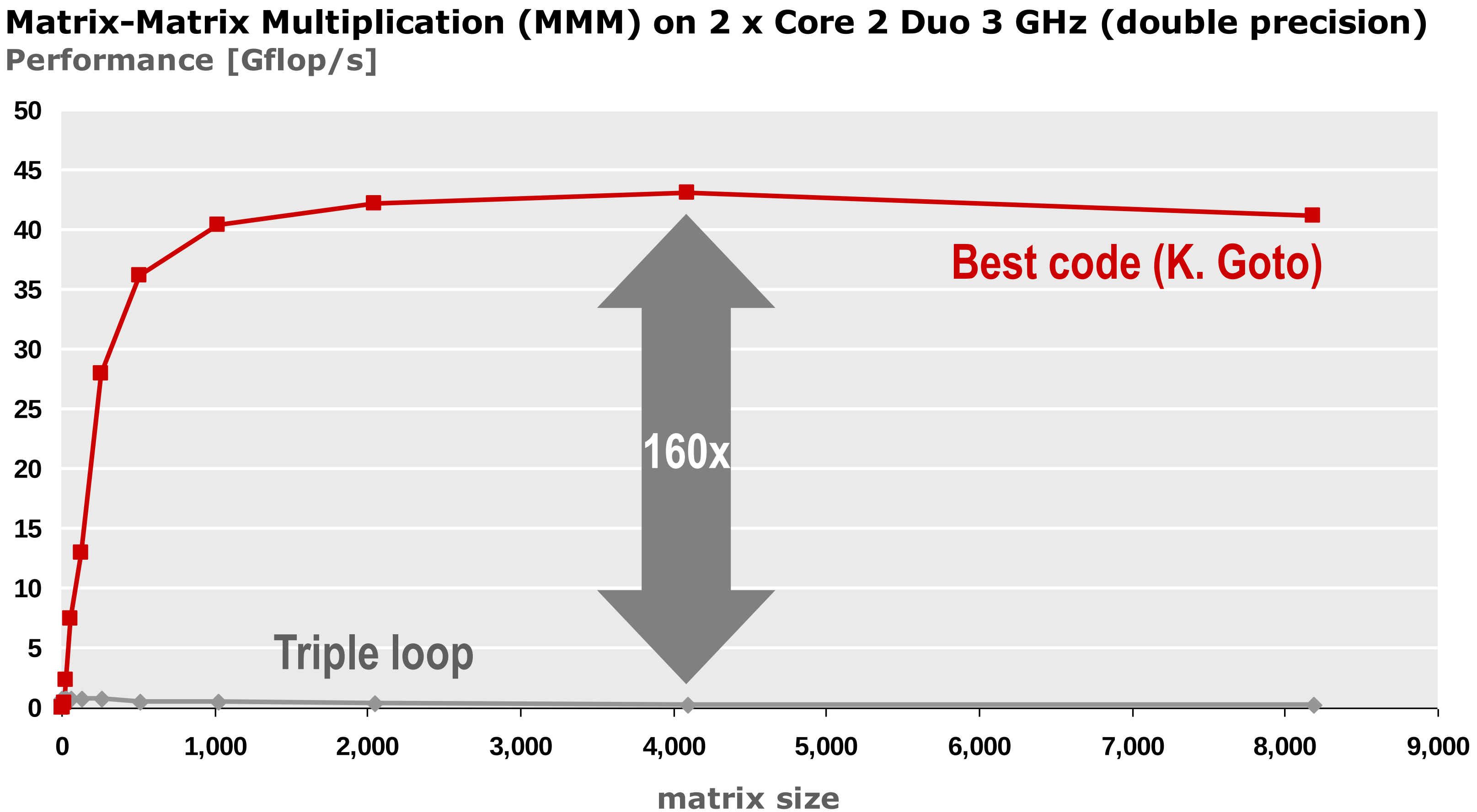
Hiroshige, Temple in park near Osaka, ca. 1853-59

The Problem: Example DFT



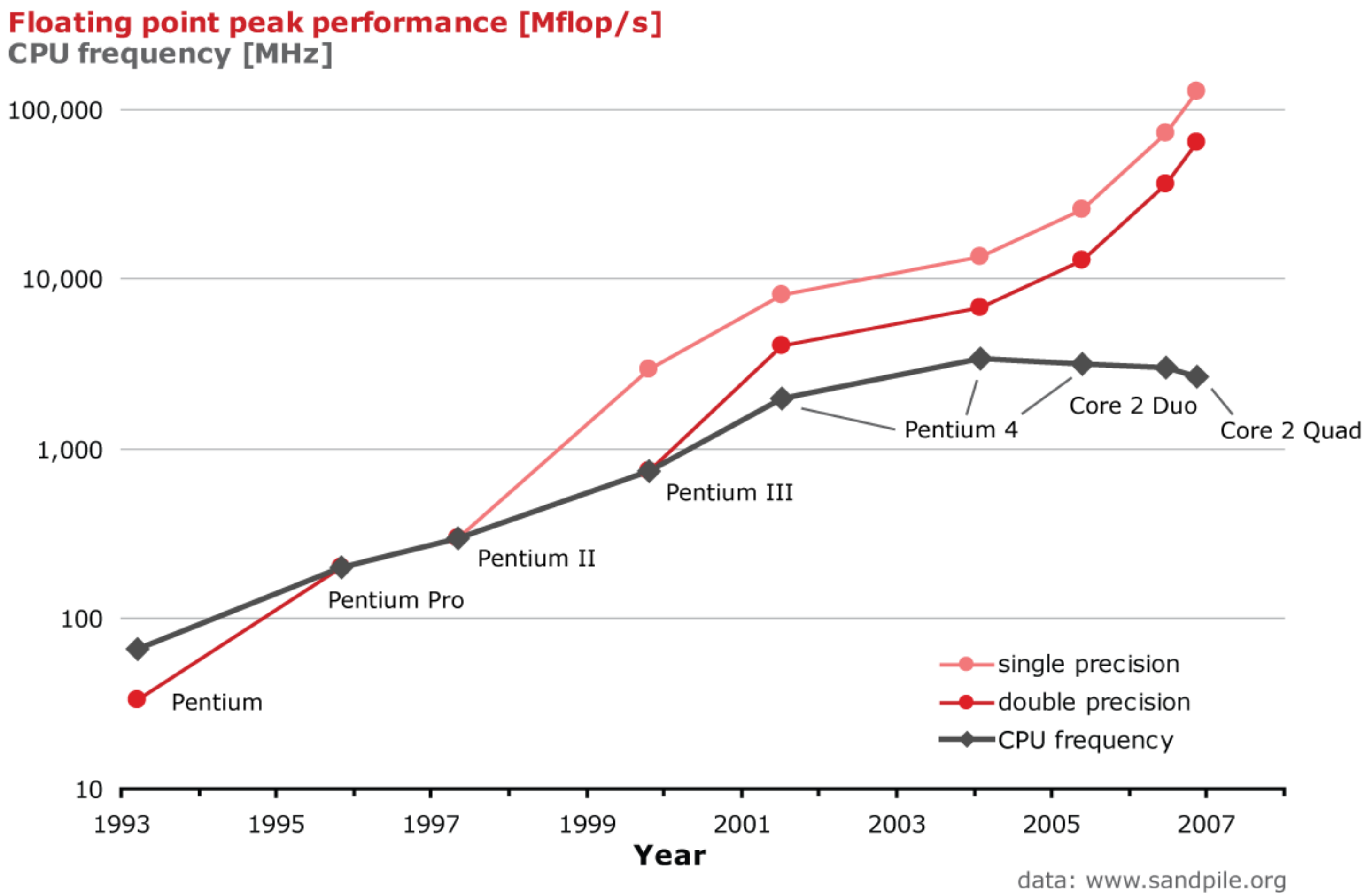
- Standard desktop computer, vendor compiler, using optimization flags
- All implementations have roughly the same operations count (~ $4n\log_2(n)$)
- *Maybe the DFT is just difficult?*

The Problem: Example MMM

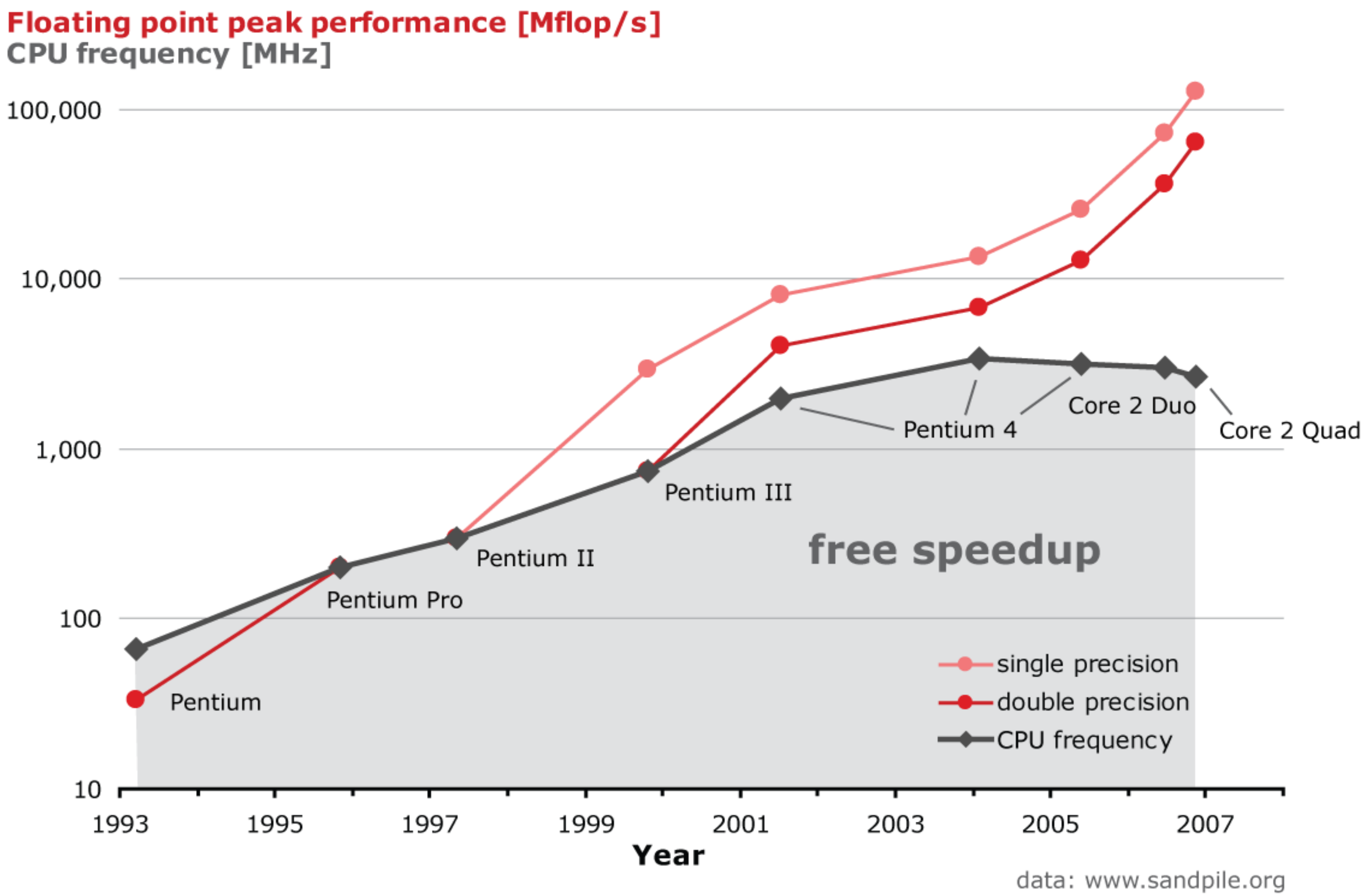


- Similar plots can be shown for all numerical kernels in linear algebra, signal processing, coding, crypto, …
- *What's going on?*

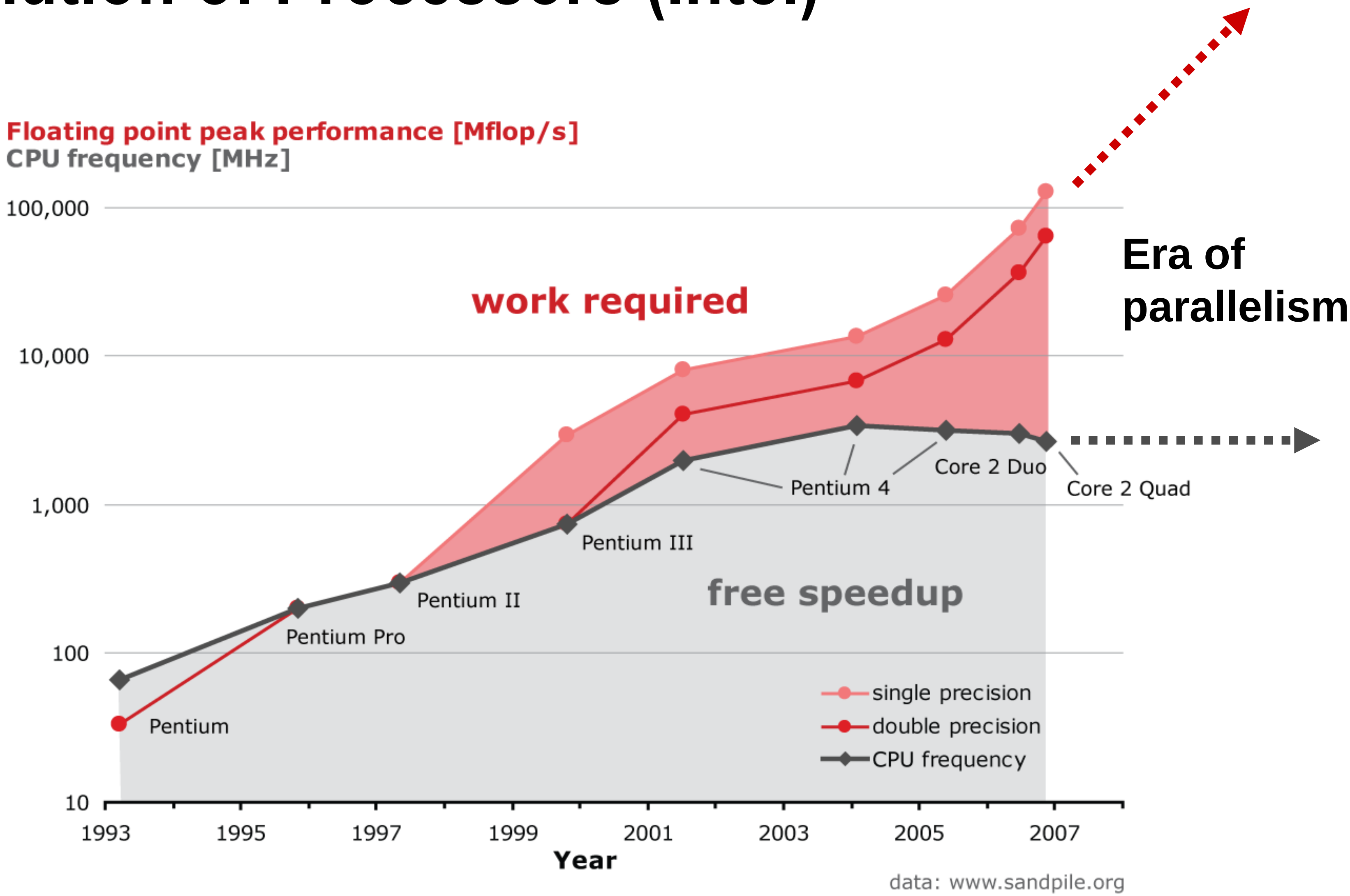
Evolution of Processors (Intel)



Evolution of Processors (Intel)

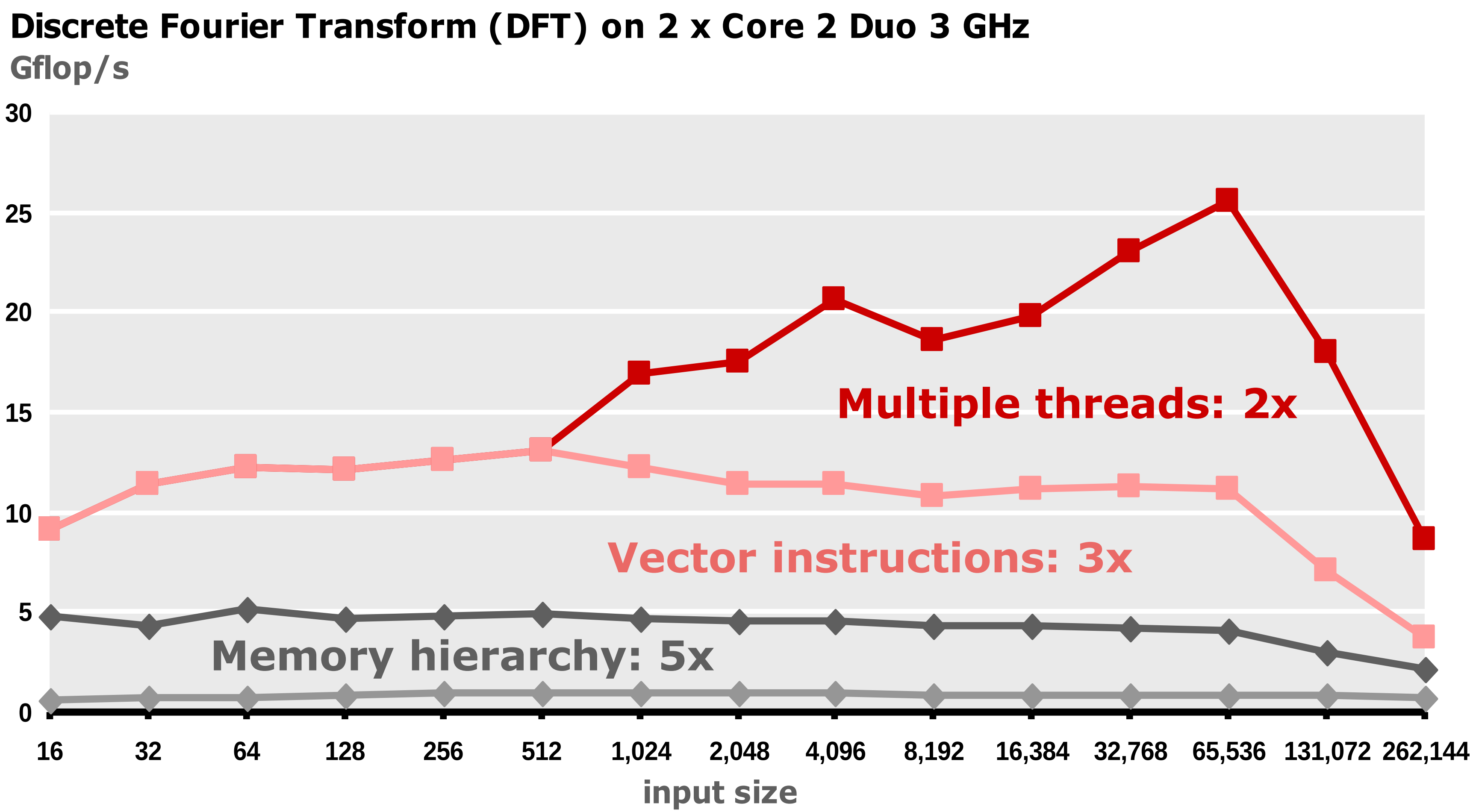


Evolution of Processors (Intel)



High performance library development becomes increasingly difficult

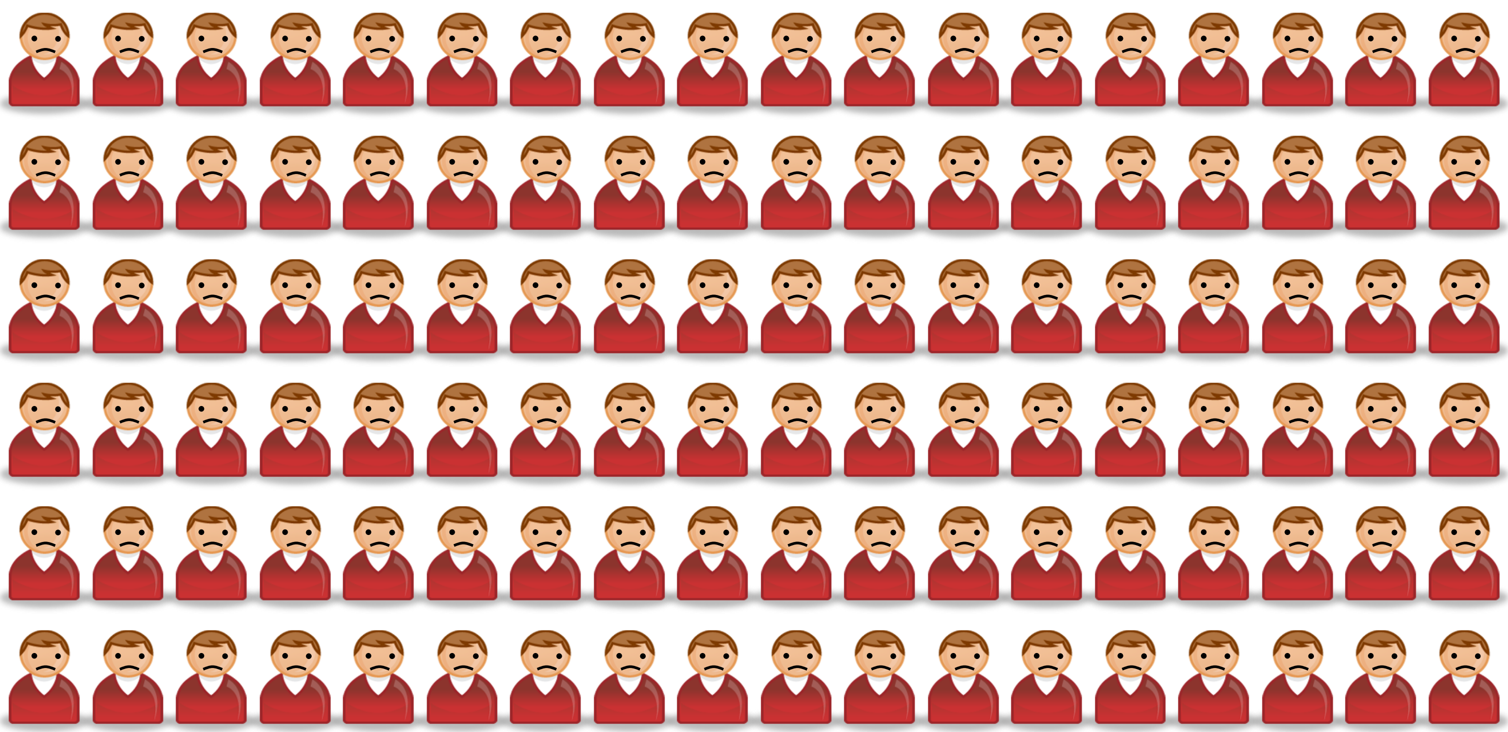
DFT Plot: Analysis



High performance library development has become a nightmare

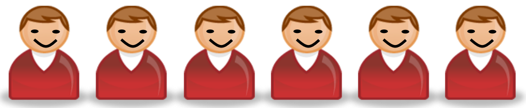
Current Solution

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

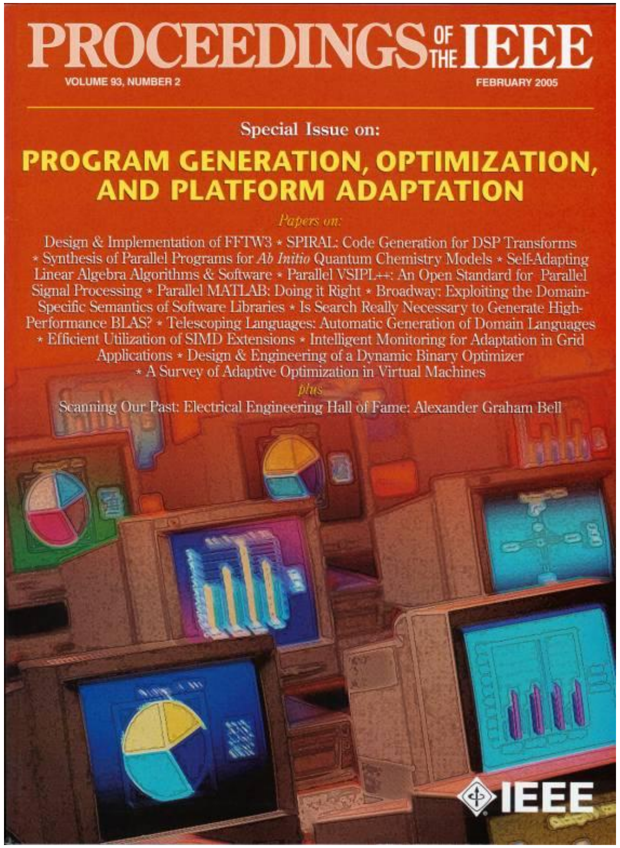


Better Solution: Automatic Performance Tuning

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



- **Research efforts**
  - Linear algebra: Phipac/ATLAS (UTK), Sparsity/Bebop (Berkeley), Flame (UT Austin)
  - Tensor computations (Ohio State)
  - PDE/finite elements: Fenics
  - Adaptive sorting (UIUC)
  - Fourier transform: FFTW (MIT)
  - Linear transforms: Spiral
  - …others
  - New compiler techniques



Proceedings of the IEEE special issue, Feb. 2005

Promising new area but more work needed
In particular for parallelism …

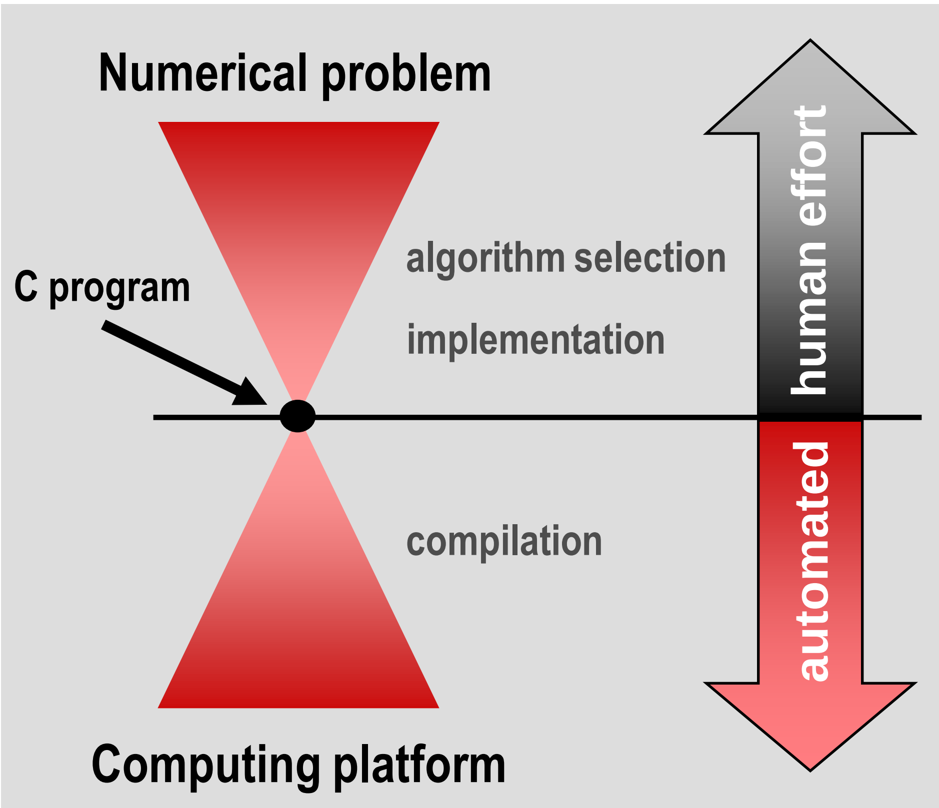
Organization

- **Spiral: Brief overview**
- Parallelization in Spiral
- Results
- Conclusion

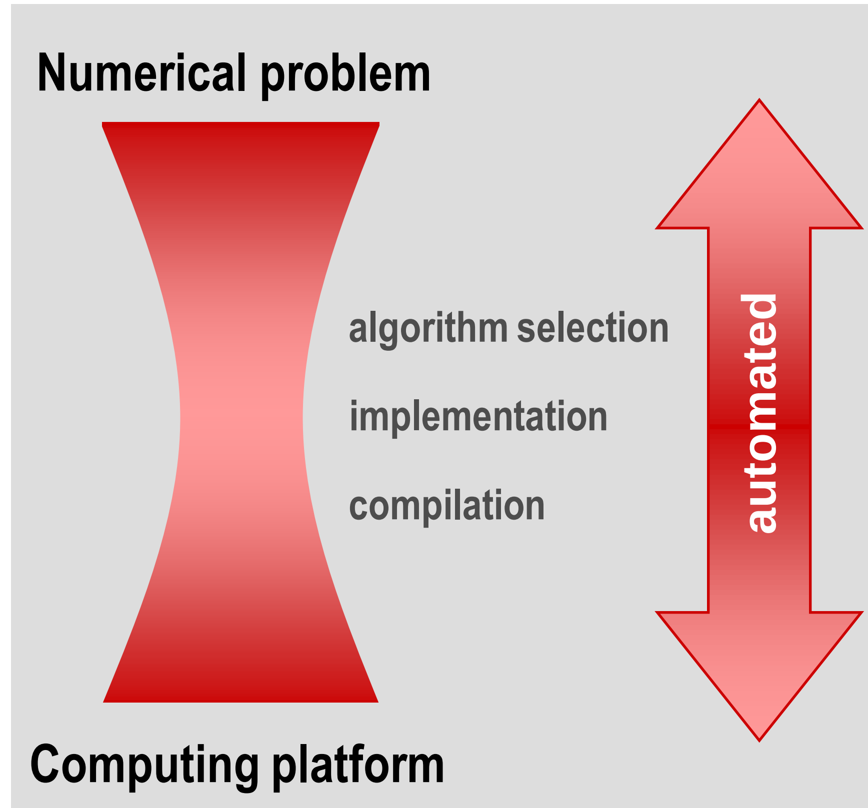
Markus Püschel, José M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan Singer, Jianxin Xiong, Franz Franchetti, Aca Gacic, Yevgen Voronenko, Kang Chen, Robert W. Johnson, and Nick Rizzolo, **SPIRAL: Code Generation for DSP Transforms,** Proceedings of the IEEE 93(2), 2005

Vision Behind Spiral

Current



Future



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

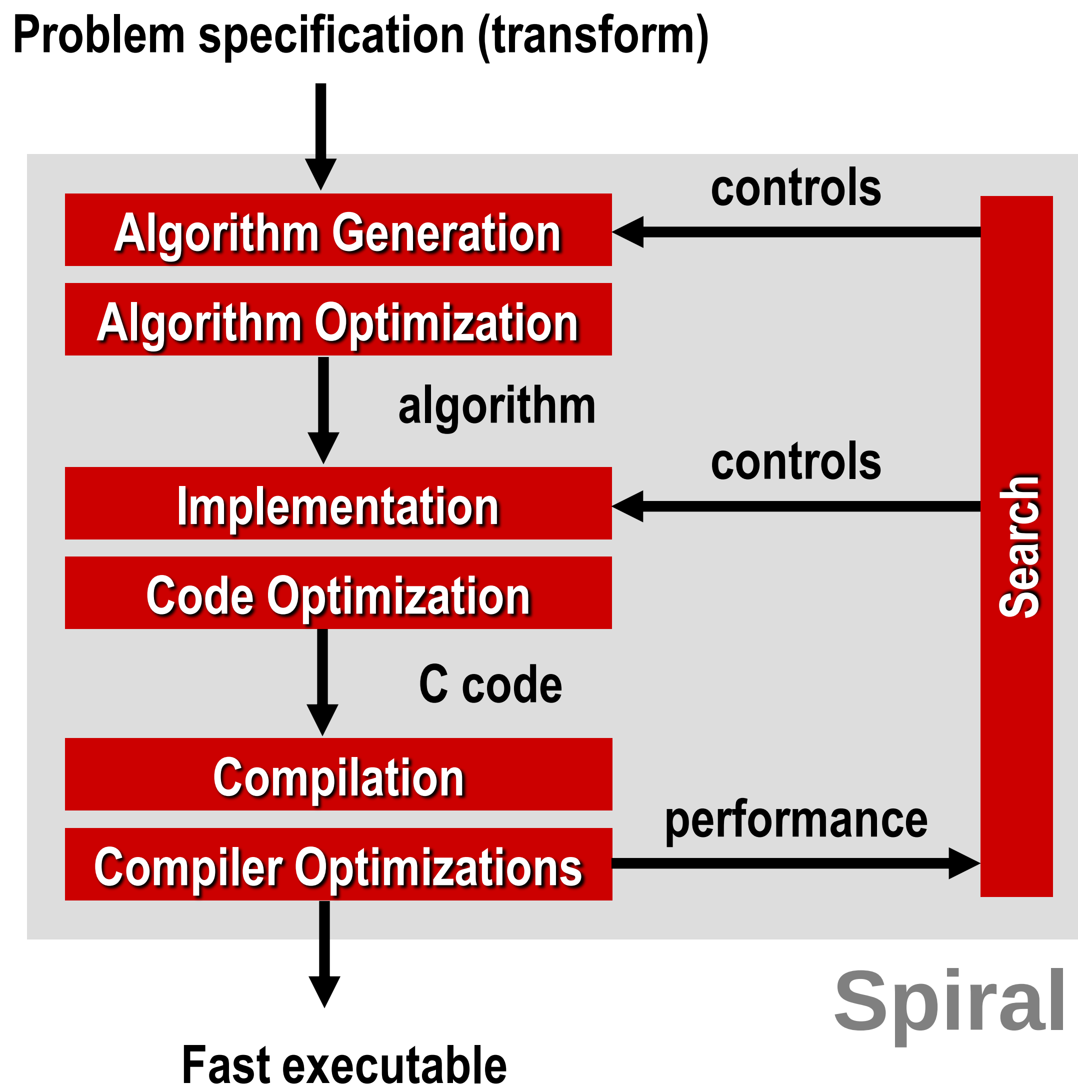
- Challenge: conquer the high abstraction level for complete automation

Spiral

- Library generator for linear transforms (DFT, DCT, DWT, 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

Basic ideas:
Declarative 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

$$x \mapsto y = T \cdot x$$

input vector (signal)

output vector (signal)

transform = matrix

- Example: Discrete Fourier transform (DFT)

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

Transform Algorithms: Example 4-point FFT

Cooley/Tukey fast Fourier transform (FFT):

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

Fourier transform

Diagonal matrix (twiddles)

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

Kronecker product

Identity

Permutation

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

Examples: Transforms (currently 55)

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

Examples: Breakdown Rules (currently ≈220)

$$
\begin{aligned}
\mathbf{DFT}_n &\rightarrow (\mathbf{DFT}_k \otimes \mathrm{I}_m)\, \mathsf{T}^n_m (\mathrm{I}_k \otimes \mathbf{DFT}_m)\, \mathsf{L}^n_k, \quad n = km \\
\mathbf{DFT}_n &\rightarrow P_n(\mathbf{DFT}_k \otimes \mathbf{DFT}_m) Q_n, \quad n = km,\ \mathsf{gcd}(k,m) = 1 \\
\mathbf{DFT}_p &\rightarrow R_p^T (\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1}) D_p (\mathrm{I}_1 \oplus \mathbf{DFT}_{p-1}) R_p, \quad p \text{ prime} \\
\mathbf{DCT\text{-}3}_n &\rightarrow (\mathrm{I}_m \oplus \mathsf{J}_m)\, \mathsf{L}^n_m (\mathbf{DCT\text{-}3}_m(1/4) \oplus \mathbf{DCT\text{-}3}_m(3/4)) \\
& \qquad \cdot (\mathsf{F}_2 \otimes \mathrm{I}_m) \begin{bmatrix} \mathrm{I}_m & 0 \oplus -\mathsf{J}_{m-1} \\ & \frac{1}{\sqrt{2}}(\mathrm{I}_1 \oplus 2\,\mathrm{I}_m) \end{bmatrix}, \quad n = 2m \\
\mathbf{DCT\text{-}4}_n &\rightarrow S_n \mathbf{DCT\text{-}2}_n\, \mathsf{diag}_{0 \le k < n}(1/(2\cos((2k+1)\pi/4n))) \\
\mathbf{IMDCT}_{2m} &\rightarrow (\mathsf{J}_m \oplus \mathrm{I}_m \oplus \mathrm{I}_m \oplus \mathsf{J}_m) \left(\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} \otimes \mathrm{I}_m \right) \oplus \left(\begin{bmatrix} -1 \\ -1 \end{bmatrix} \otimes \mathrm{I}_m \right) \right) \mathsf{J}_{2m}\, \mathbf{DCT\text{-}4}_{2m} \\
\mathbf{WHT}_{2^k} &\rightarrow \prod_{i=1}^{t} (\mathrm{I}_{2^{k_1+\cdots+k_{i-1}}} \otimes \mathbf{WHT}_{2^{k_i}} \otimes \mathrm{I}_{2^{k_{i+1}+\cdots+k_t}}), \quad k = k_1 + \cdots + k_t \\
\mathbf{DFT}_2 &\rightarrow \mathsf{F}_2 \\
\mathbf{DCT\text{-}2}_2 &\rightarrow \mathsf{diag}(1, 1/\sqrt{2})\, \mathsf{F}_2 \\
\mathbf{DCT\text{-}4}_2 &\rightarrow \mathsf{J}_2\, \mathsf{R}_{13\pi/8}
\end{aligned}
$$

Base case rules

Combining these rules yields many algorithms for every given transform

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_Canonicize
- 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

DFT4 3
- DFT4_Base
- DFT4_CT
- DFT4_PRDFT4

DFTBR 1
- DFTBR_HW

DHT 3
- DHT_Base
- DHT_DCT1andDST1
- DHT_Radix2

Cooley-Tukey FFT

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
- DWT_Base2
- DWT_Base4
- DWT_Base8
- DWT_Lifting

DWTper 4
- DWTper_Mallat_2
- DWTper_Mallat
- DWTper_Polyphase
- DWTper_Lifting

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
- IPRDFT2_CT_Radix2

IPRDFT3 3
- MDDFT_Dimless
- MDDFT_tSPL_RowCol

PDCT4 2
- PDCT4_Base2
- PDCT4_CT

PDHT 3
- PDHT1_Base2
- PDHT1_CT
- PDHT1_CT_Radix2

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
- PRDFT1_CT_Radix2
- PRDFT4_Base1
- PRDFT4_Base2
- PRDFT4_CT
- PRDFT4_Trig

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

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

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

Breakdown rules in Spiral:

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

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]);`

$y = \left(\bigoplus_{i=0}^{m-1} A_n^i\right) x$	`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

$$(\mathrm{DFT}_2 \otimes I_4)\, T^8_4\, (I_2 \otimes ((\mathrm{DFT}_2 \otimes I_2) \cdot T^4_2\, (I_2 \otimes \mathrm{DFT}_2)\, L^4_2))\, L^8_2$$

∑-SPL:
[PLDI 05]

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

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
  scheduling
  ......

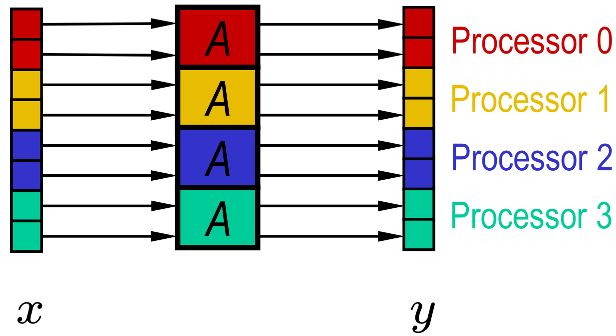
Organization

- Spiral: Brief overview
- **Parallelization in Spiral**
- Results
- Conclusion

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

- Governing construct: tensor product

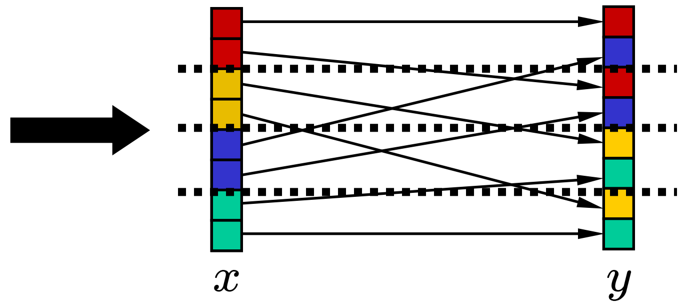
$$y = \left(\mathrm{I}_p \otimes A \right) x$$



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**

$$\underbrace{A}_{\mathrm{smp}(p,\mu)}$$

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

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

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

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

- Tagged operators (no further rewriting necessary)

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

- **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

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

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

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

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

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

Simple Rewriting Example

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

Loop splitting + loop exchange

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

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

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

$$\dots$$

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

$$\dots$$

$$\rightarrow \underline{\Big((\mathsf{L}_m^{mp} \otimes \mathrm{I}_{n/p\mu}) \otimes_\mu \mathrm{I}_\mu\Big)}\underline{\Big(\mathrm{I}_p \otimes_{\|} (\mathbf{DFT}_m \otimes \mathrm{I}_{n/p})\Big)}\underline{\Big((\mathsf{L}_p^{mp} \otimes \mathrm{I}_{n/p\mu}) \otimes_\mu \mathrm{I}_\mu\Big)}$$

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

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

Same Approach for Other Parallel Paradigms

Message Passing: [ISPA 06]

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

$$\dots$$

$$\dots$$

$$\dots$$

$$\to \underbrace{(\mathrm{I}_p\otimes \mathrm{L}^{mn/p}_{m/p})}_{\mathrm{par}(p)} \underbrace{(\mathrm{L}^{p^2}_p\otimes \mathrm{I}_{mn/p^2})}_{\mathrm{par}(p\leftarrow q)} \underbrace{(\mathrm{I}_q\otimes(\mathrm{I}_{p/q}\otimes\mathrm{L}^n_p\otimes\mathrm{I}_{m/p}))}_{\mathrm{par}(q)} \underbrace{(\mathrm{I}_q\otimes(\mathrm{I}_{n/q}\otimes\mathrm{DFT}_m))}_{\mathrm{par}(q)}$$
$$\underbrace{(\mathrm{I}_q\otimes \mathrm{L}^{mn/q}_{m/q})}_{\mathrm{par}(q)} \underbrace{(\mathrm{L}^{q^2}_q\otimes \mathrm{I}_{mn/q^2})}_{\mathrm{par}(q)} \underbrace{(\mathrm{I}_q\otimes(\mathrm{L}^n_q\otimes\mathrm{I}_{m/q}))}_{\mathrm{par}(q)} \underbrace{\mathrm{T}^{mn}_n}_{\mathrm{par}(q)} \underbrace{(\mathrm{I}_q\otimes(\mathrm{I}_{m/q}\otimes\mathrm{DFT}_n))}_{\mathrm{par}(q)}$$
$$\underbrace{(\mathrm{I}_q\otimes(\mathrm{I}_{p/q}\otimes \mathrm{L}^{mn/p}_{m/p}))}_{\mathrm{par}(q)} \underbrace{(\mathrm{L}^{p^2}_p\otimes \mathrm{I}_{mn/p^2})}_{\mathrm{par}(q\leftarrow p)} \underbrace{(\mathrm{I}_p\otimes(\mathrm{L}^n_p\otimes\mathrm{I}_{m/p}))}_{\mathrm{par}(p)}$$

With Bonelli, Lorenz, Ueberhuber, TU Vienna

Vectorization: [IPDPS 02, VecPar 06]

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

$$\dots$$

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

$$\dots$$

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

Cg/OpenGL for GPUs:

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

$$\dots$$

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

With Shen, TU Denmark

Verilog for FPGAs: [DAC 05]

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

$$\dots$$

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

$$\dots$$

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

With Milder, Hoe, CMU

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

Cooley-Tukey FFT in OL: $\mathrm{DFT} \to (\mathrm{DFT} \otimes \mathrm{I}) \circ D \circ (\mathrm{I} \otimes \mathrm{DFT}) \circ \mathrm{L}\,.$

Viterbi in OL: $\mathrm{Vit} \to \pi \circ \left(\prod \left(\mathrm{I} \otimes \mathrm{V}\right) \circ \left(\mathrm{L} \times \mathrm{I}\right)\right) \circ \left(\mathrm{C} \times \mathrm{C} \times \mathrm{I}\right)$

Mat-Mat-Mult: $\mathrm{MMM} \to \mathrm{I} \otimes \mathrm{MMM}$

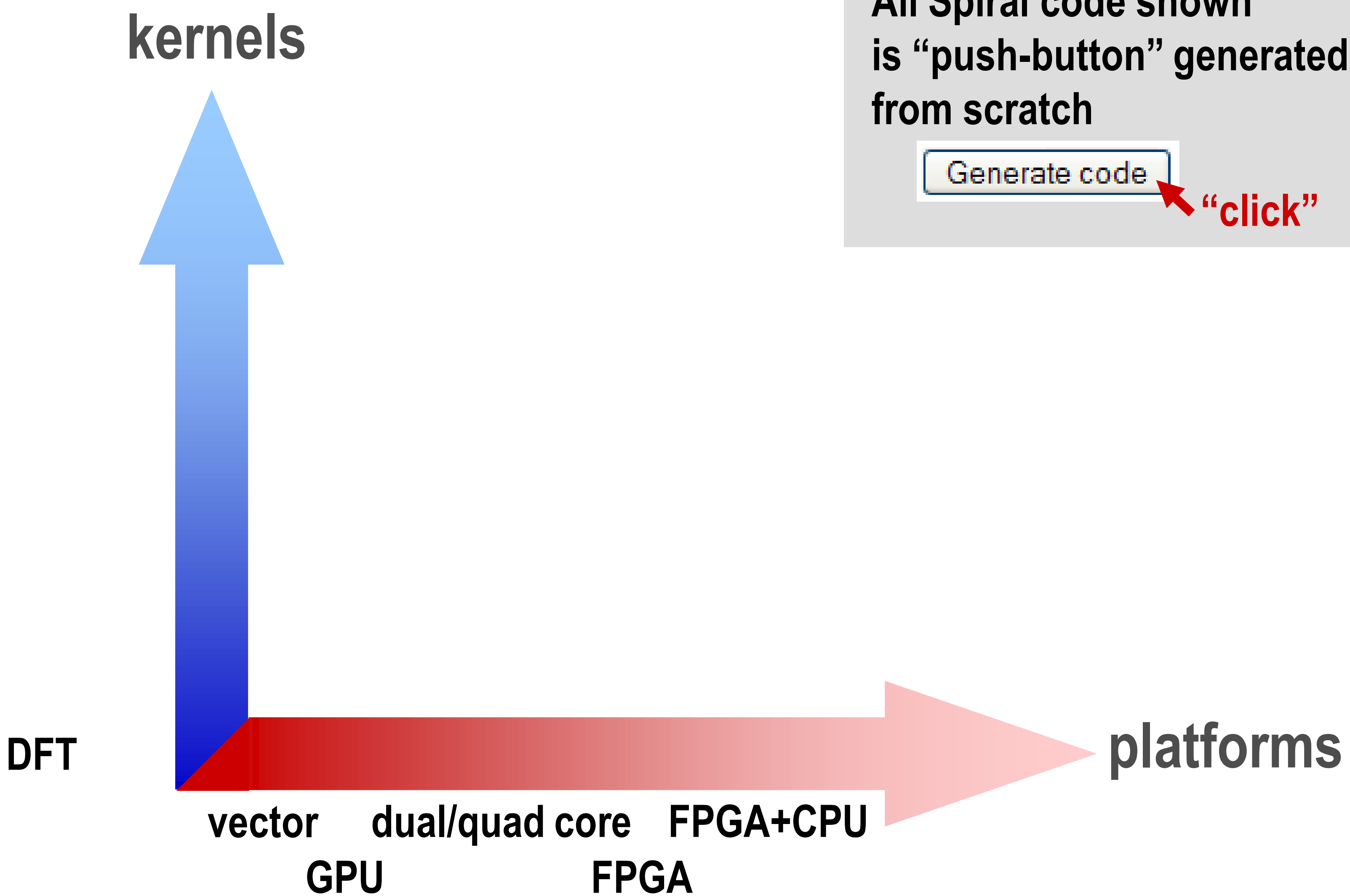
$$\mathrm{MMM} \to \left(\mathrm{I} \otimes \mathrm{L}\right) \circ \left(\mathrm{MMM} \otimes \mathrm{I}\right) \circ \left(\mathrm{I} \times (\mathrm{I} \otimes \mathrm{L})\right)$$

- **We have first results beyond transforms!**

Organization

- Spiral overview
- Parallelization in Spiral
- **Results**
- Concluding remarks

Benchmarks

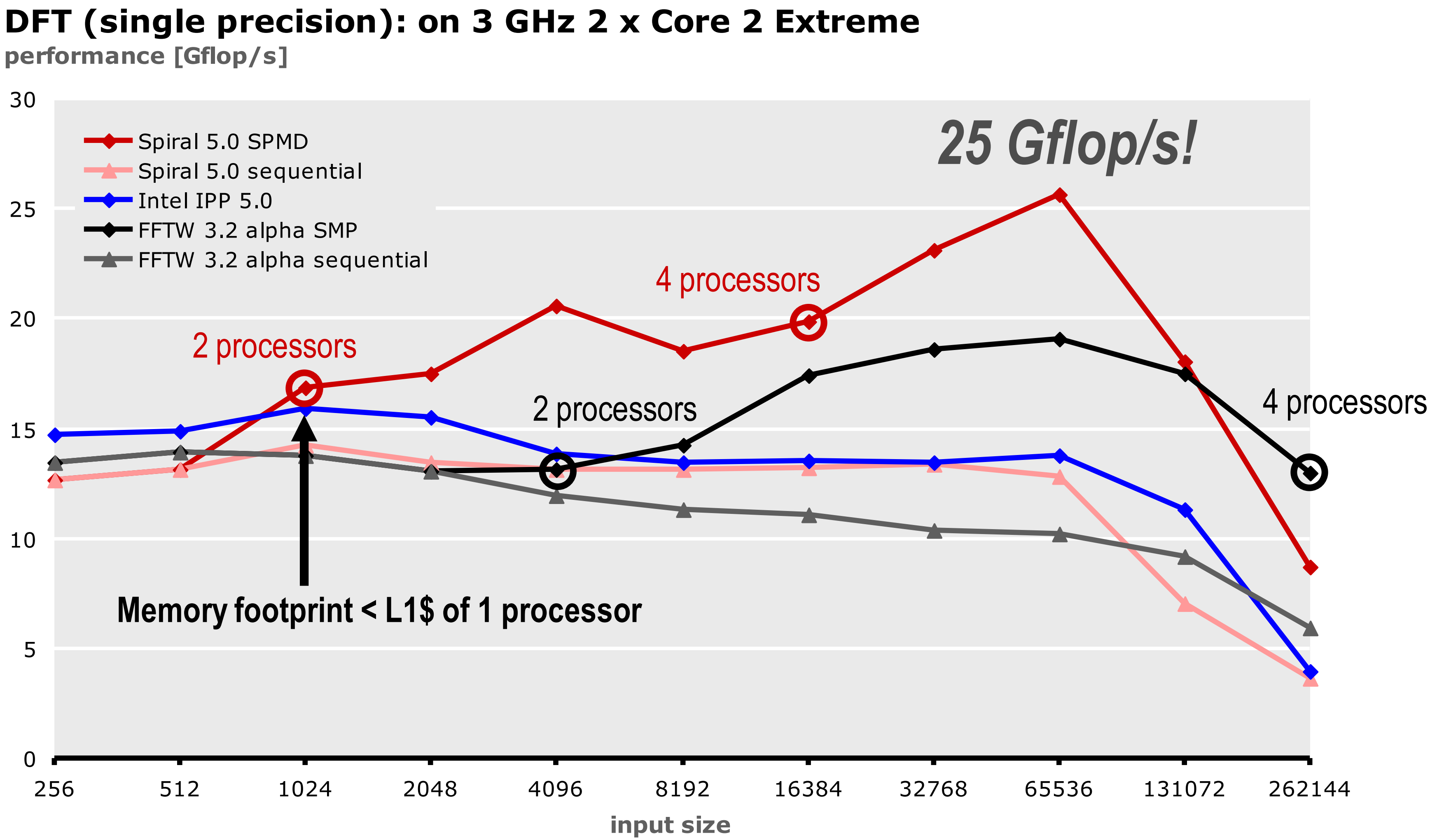


All Spiral code shown is "push-button" generated from scratch

Generate code

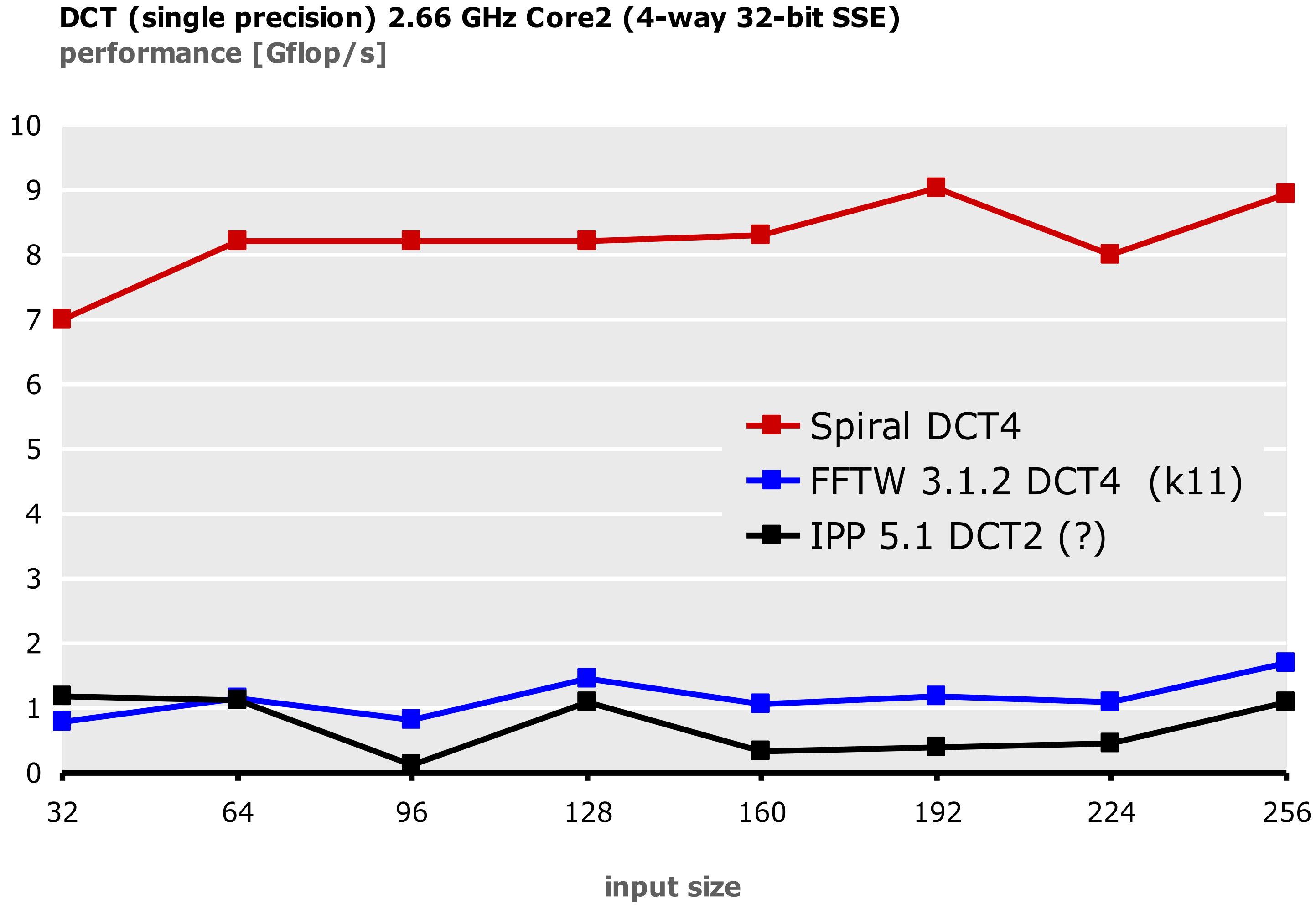
"click"

Benchmark: Vector and SMP



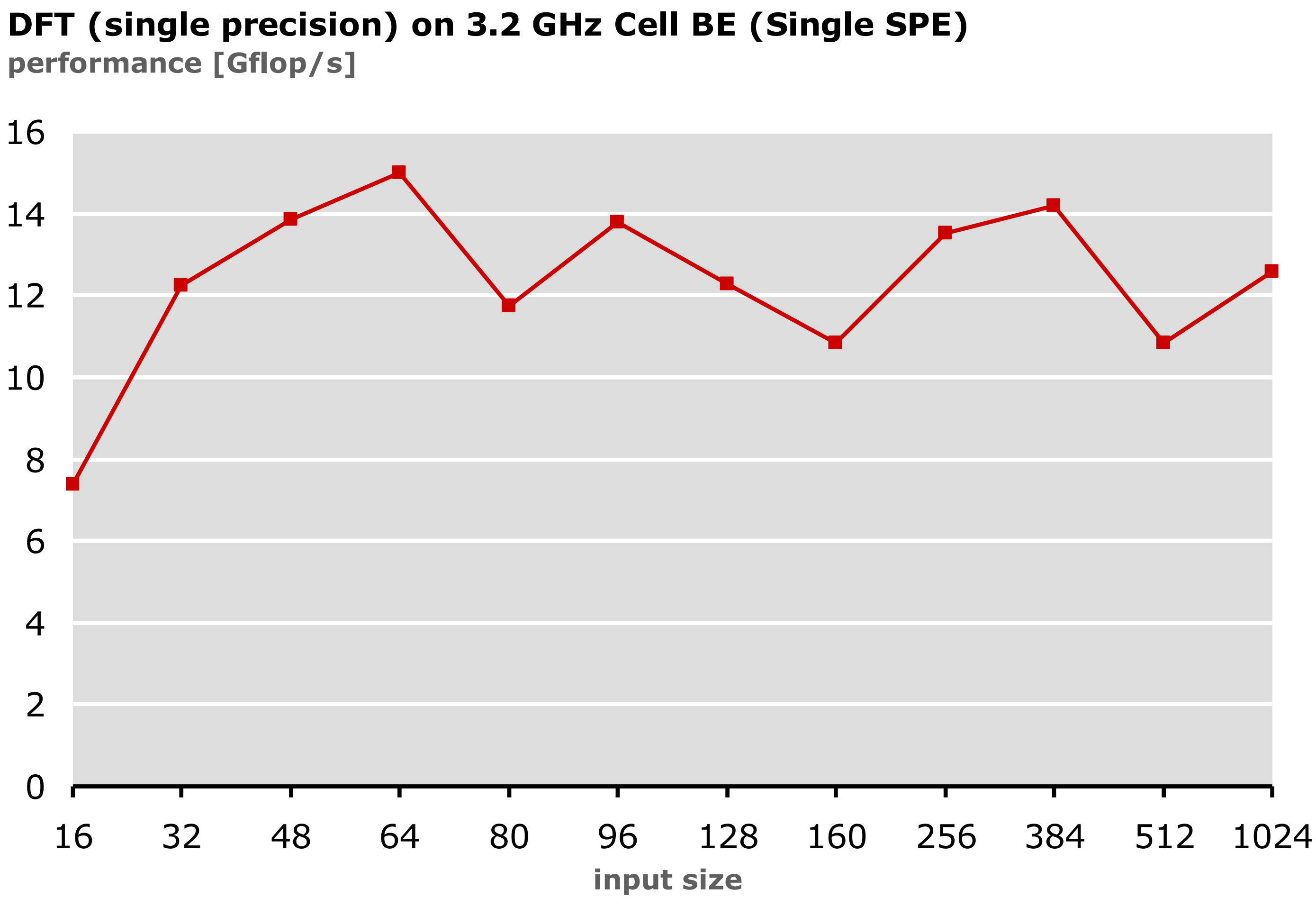
4-way vectorized + up to 4-threaded + adapted to the memory hierarchy

DCT4, Multiples of 32: 4-way Vectorized



Less popular transform usually means no good code

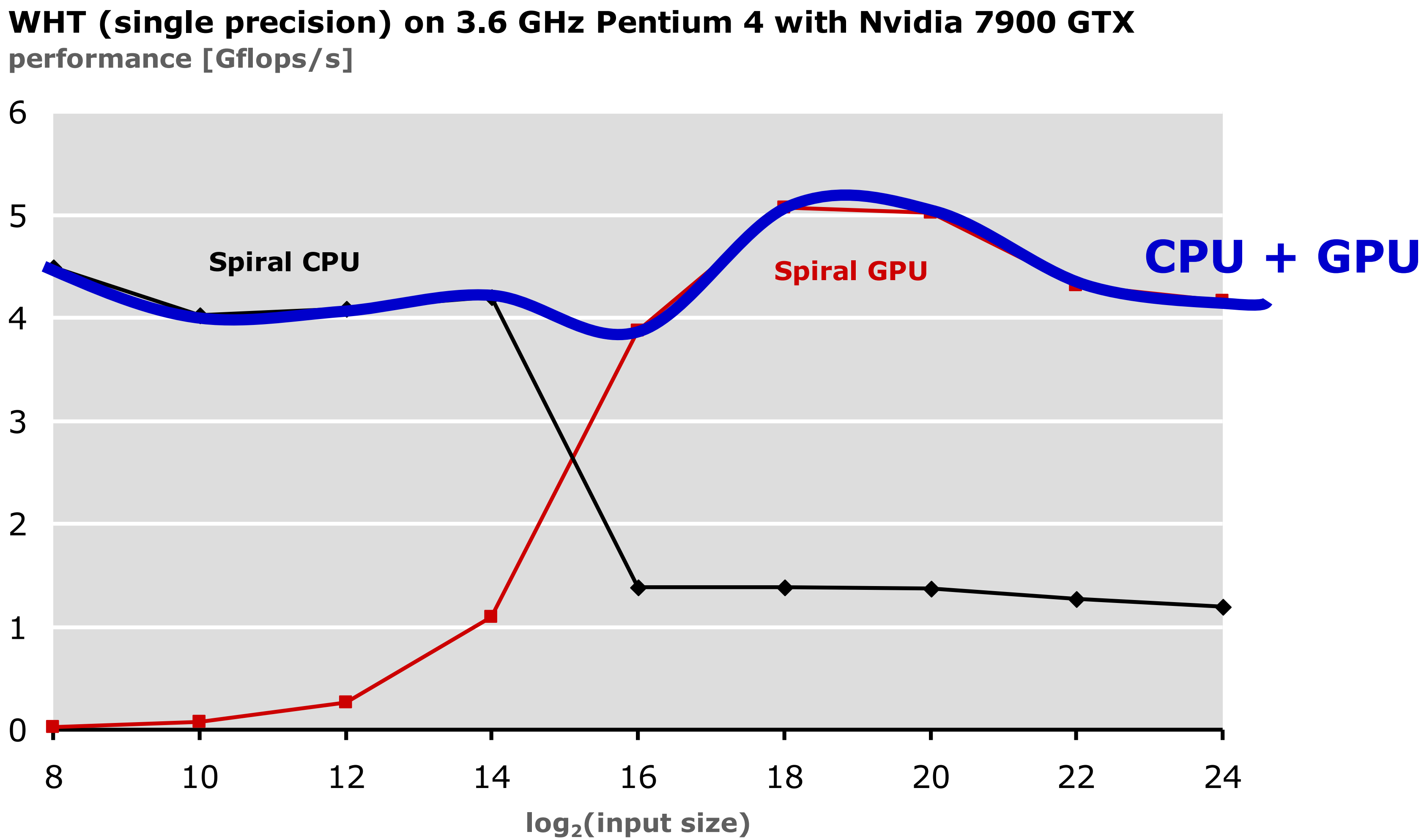
Benchmark: Cell (1 processor = SPE)



Generated using the simulator; run at Mercury (thanks to Robert Cooper)

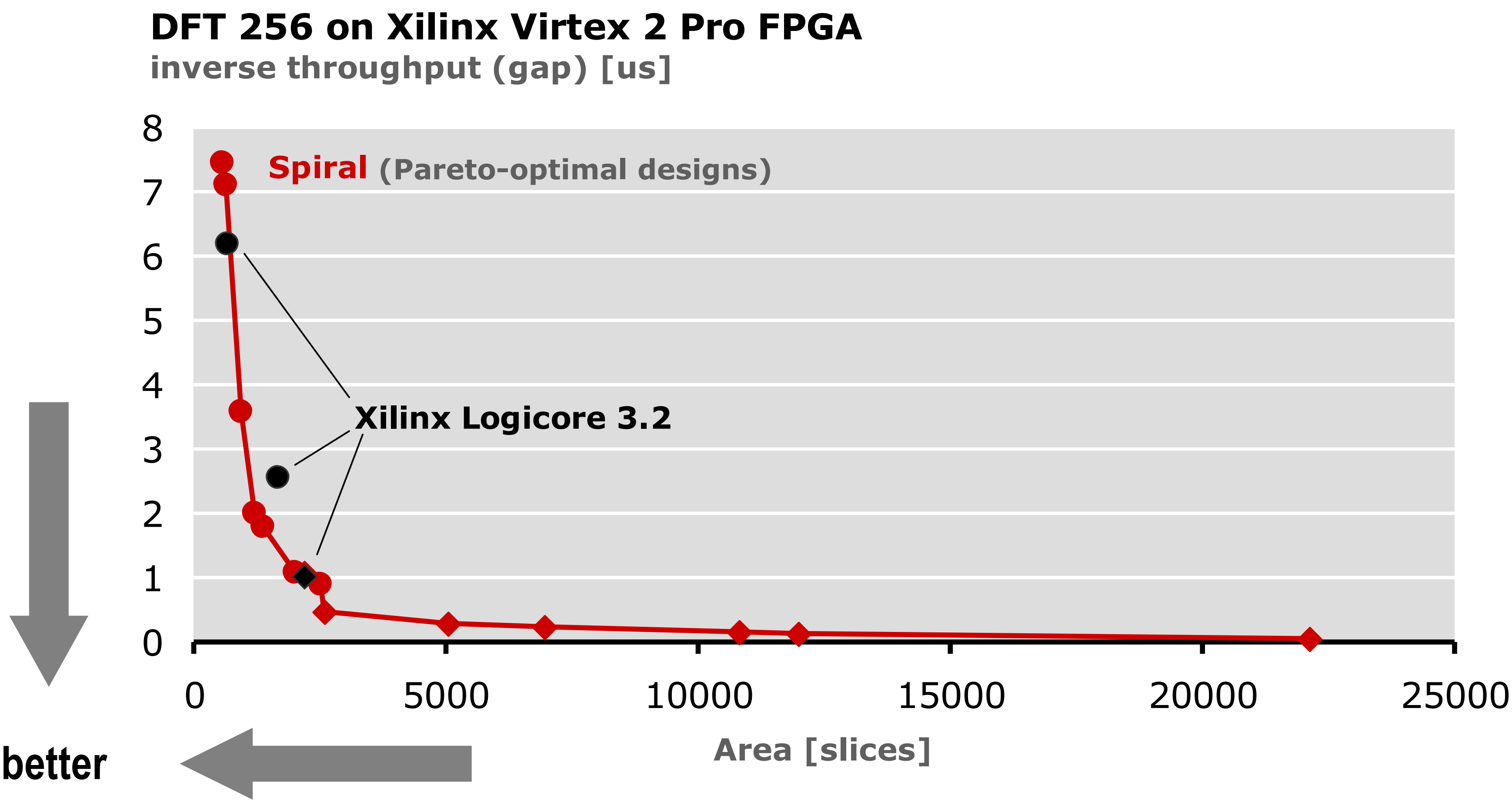
Joint work with Th. Peter (ETH Zurich), S. Chellappa, M. Telgarsky, J. Moura (CMU)

Benchmark: GPU



Joint work with H. Shen, TU Denmark

Benchmark: FPGA



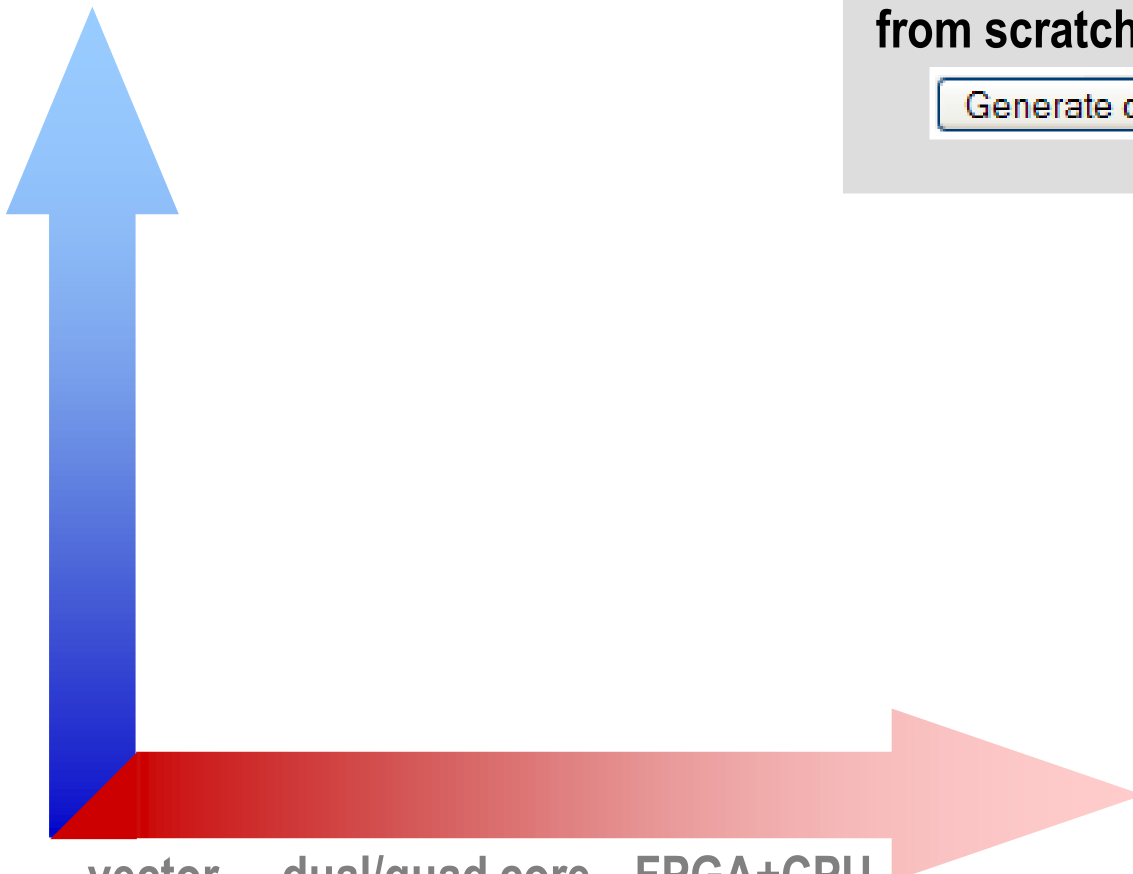
- competitive with professional designs
- much larger set of performance/area trade-offs

Joint work with P. Milder, J. Hoe (CMU)

Benchmarks

All Spiral code shown
is "push-button" generated
from scratch

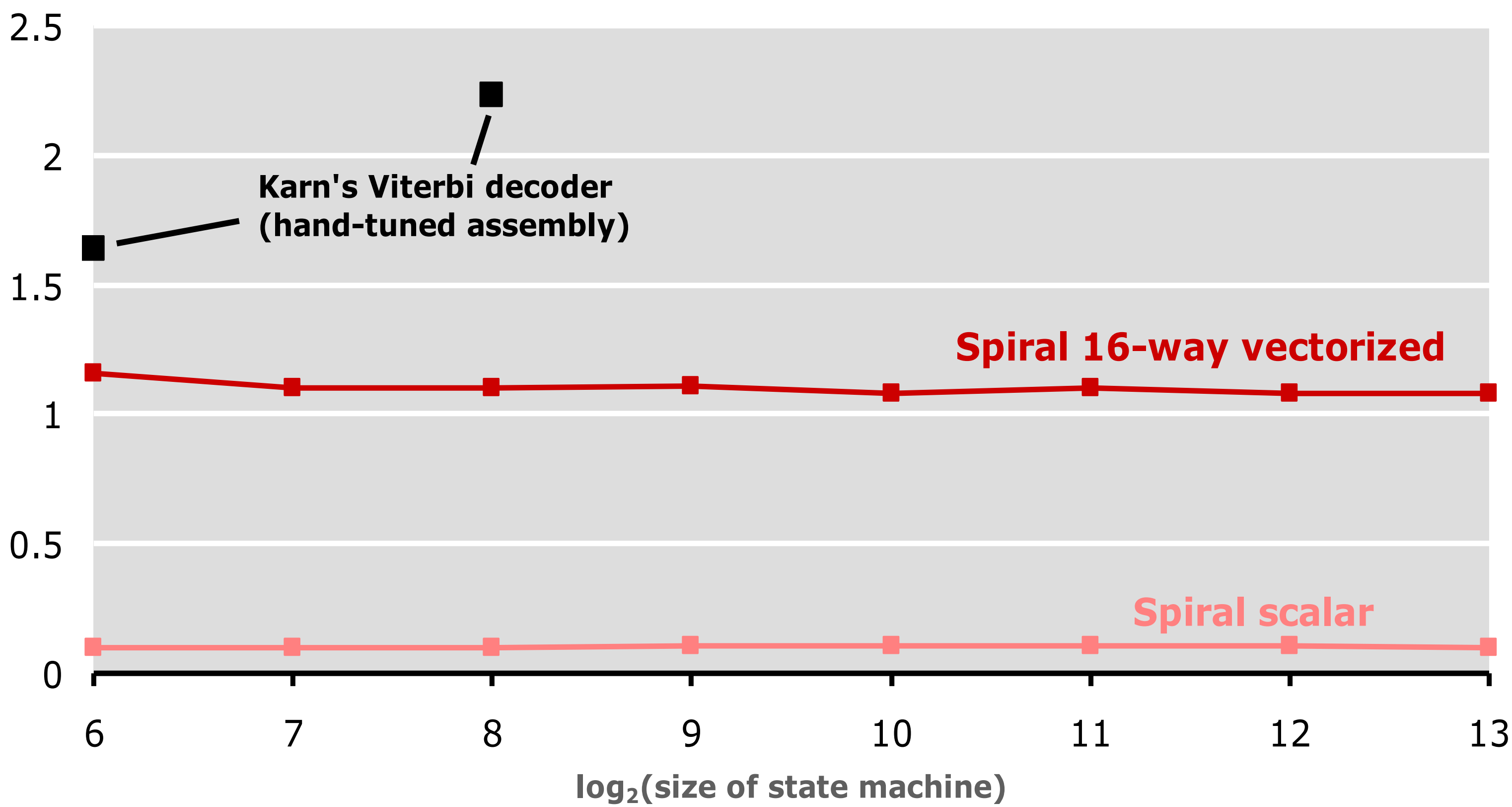
Generate code "click"



Beyond Transforms : Viterbi Decoding

Viterbi decoding (8-bit) on 2.66 GHz Core 2 Duo
performance [Gbutterflies/s]

1 butterfly
= ~22 ops

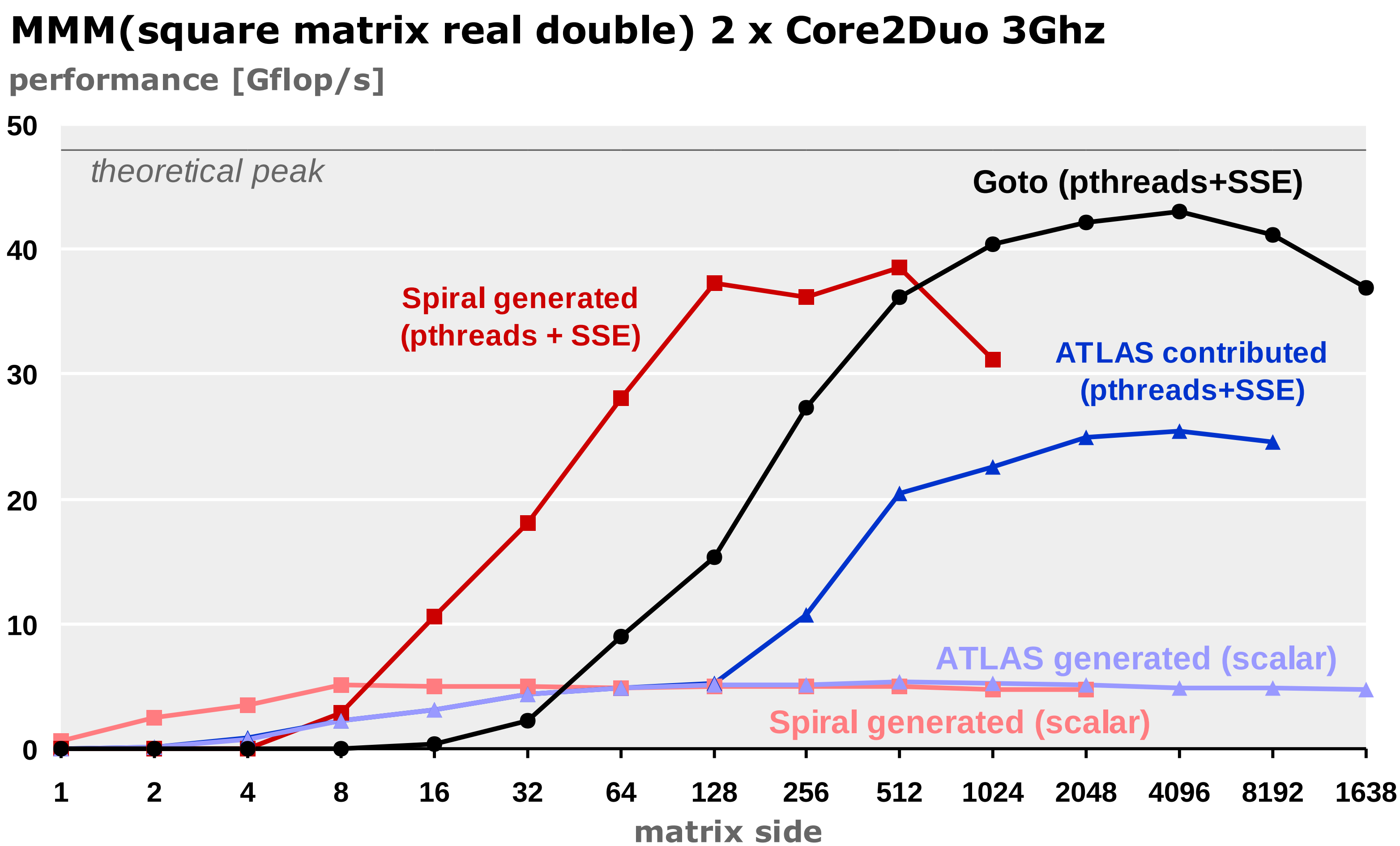


Vectorized using practically the same rules as for DFT

Joint work with S. Chellappa, CMU

Karn: http://www.ka9q.net/code/fec/

Beyond Transforms: Matrix-Matrix-Multiply



All measurements with warm cache
ATLAS code is optimized for cold cache

work with F. de Mesmay, CMU

Organization

- Spiral overview
- Parallelization in Spiral
- Results
- Concluding remarks

Spiral Summary

- **Spiral: The computer implements and optimizes transforms**
  - Rigorous formal framework
  - Support for different platform paradigms (vector, SMP, GPU, FPGA, …)
  - Optimization at high abstraction level through rewriting
    + empirical search over alternatives
  - The generated code is often faster than human written code
  - We have extended the framework beyond transforms

- **What we have learned**
  - Declarative representation of algorithms (mathematical DSL)
  - 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 …..

Attacking the Parallel Library Challenge

Automation in High Performance Library Development

Programming languages
Program generation

Symbolic Computation
Rewriting

High-Performance Parallel Library Development

Software
Scientific Computing

Algorithms
Mathematics

Compilers

We Need to Work Together