$A \otimes I_n$

Program Synthesis For Performance

Markus Püschel

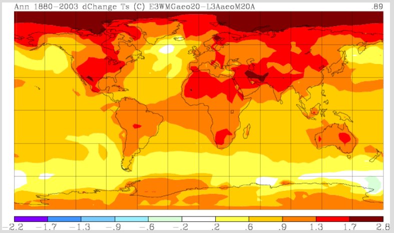
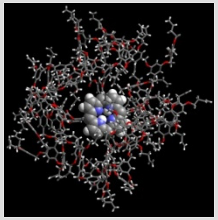
Computer Science

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

SPIRAL
www.spiral.net


```
__m128i t3  = _mm_unpacklo_epi16(X[0], X[1]);
__m128i t4  = _mm_unpackhi_epi16(X[0], X[1]);
__m128i t7  = _mm_unpacklo_epi16(X[2], X[3]);
__m128i t8  = _mm_unpackhi_epi16(X[2], X[3]);
```

Scientific Computing

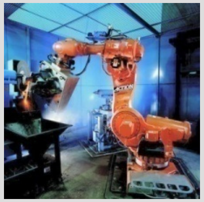
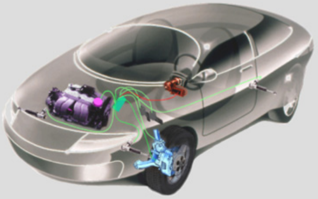



Simulations: Physics, Biology, ...

Mainstream Computing


Audio/Image/Video processing, ...

Embedded Computing

Signal processing, communication, ...

Unlimited need for performance

Many applications, but relatively few (~100 to 1000) components:

- Matrix multiplication
- Filters
- Fourier transform
- Coding/decoding
- Geometric transformations
- Graph algorithms
- ...

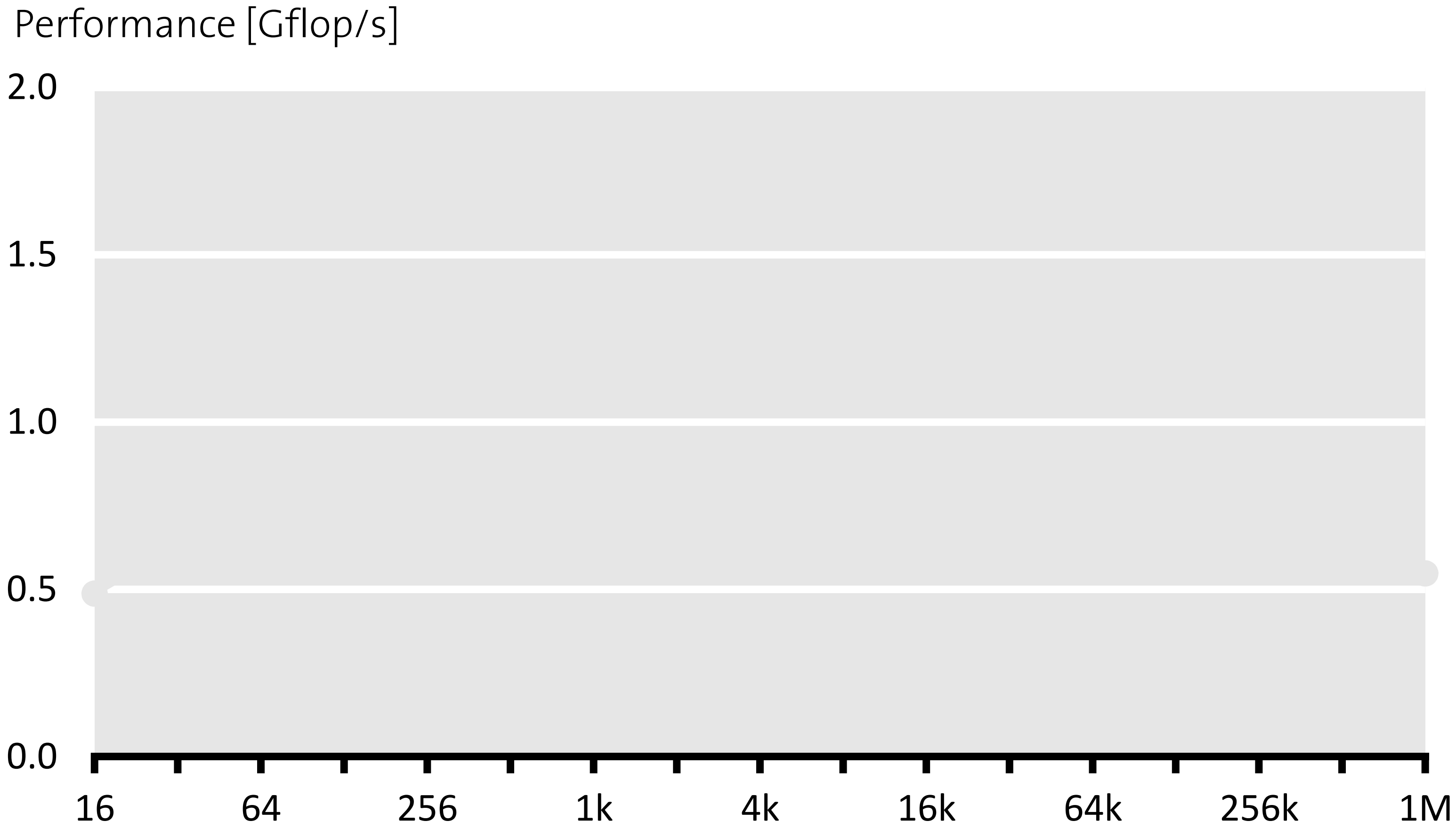
Fast components → fast applications

Software Performance: Traditional Approach

Algorithms	*optimal op count*
Software	
Compilers	*performance optimization*
Microarchitecture	

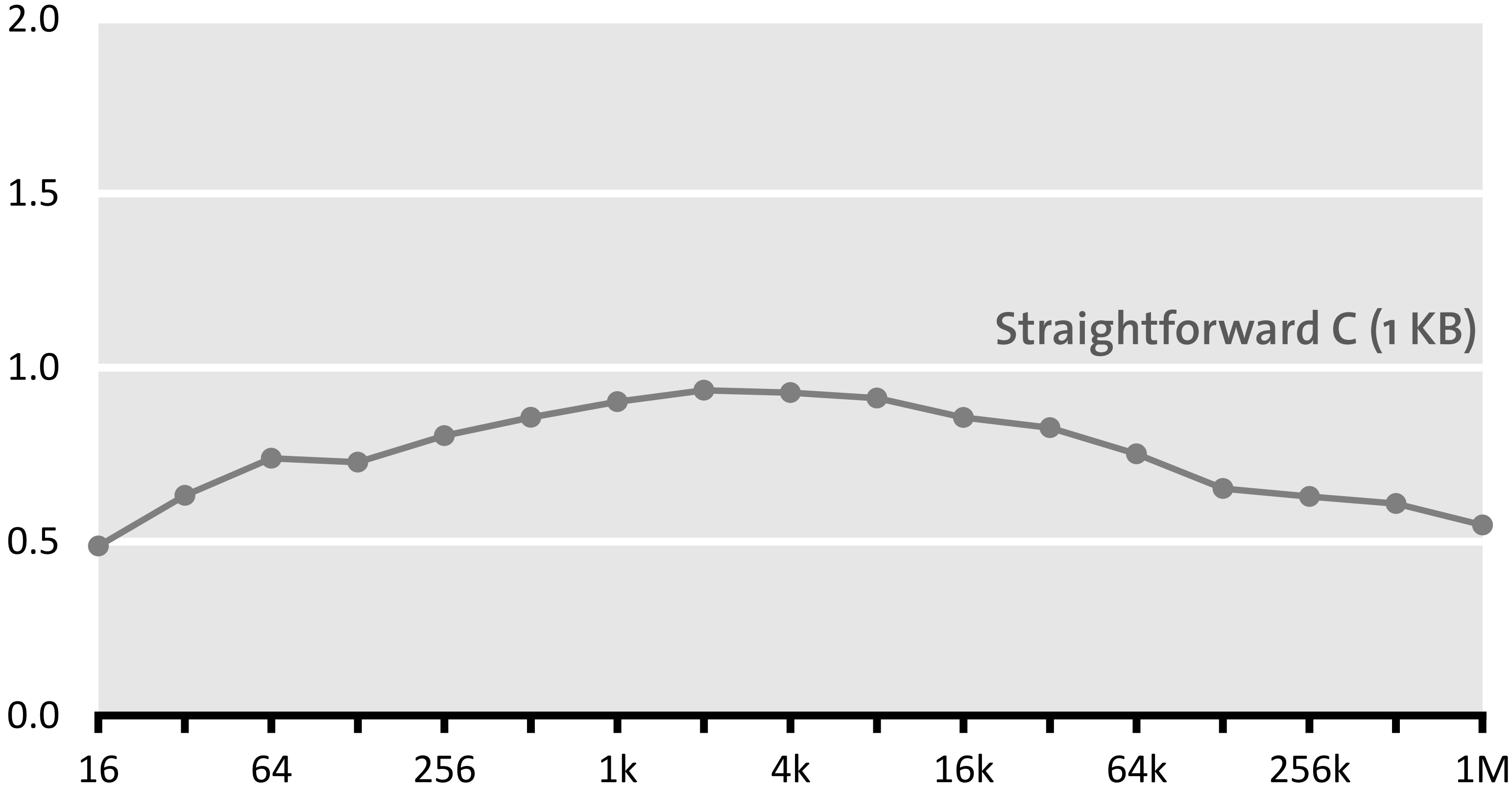
How well does that work?

Discrete Fourier transform (single precision) on Intel Core i7 (4 cores)



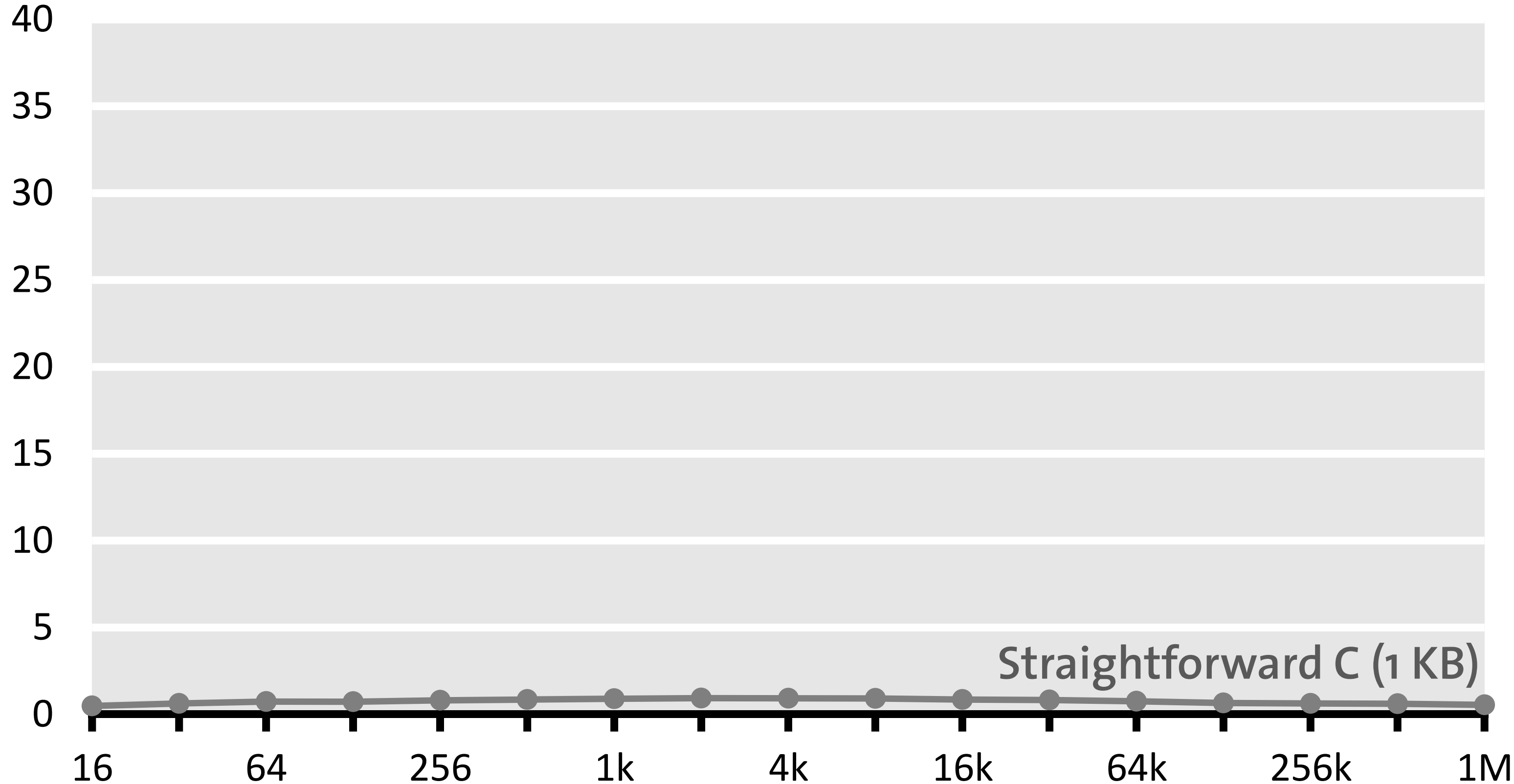
Discrete Fourier transform (single precision) on Intel Core i7 (4 cores)

Performance [Gflop/s]



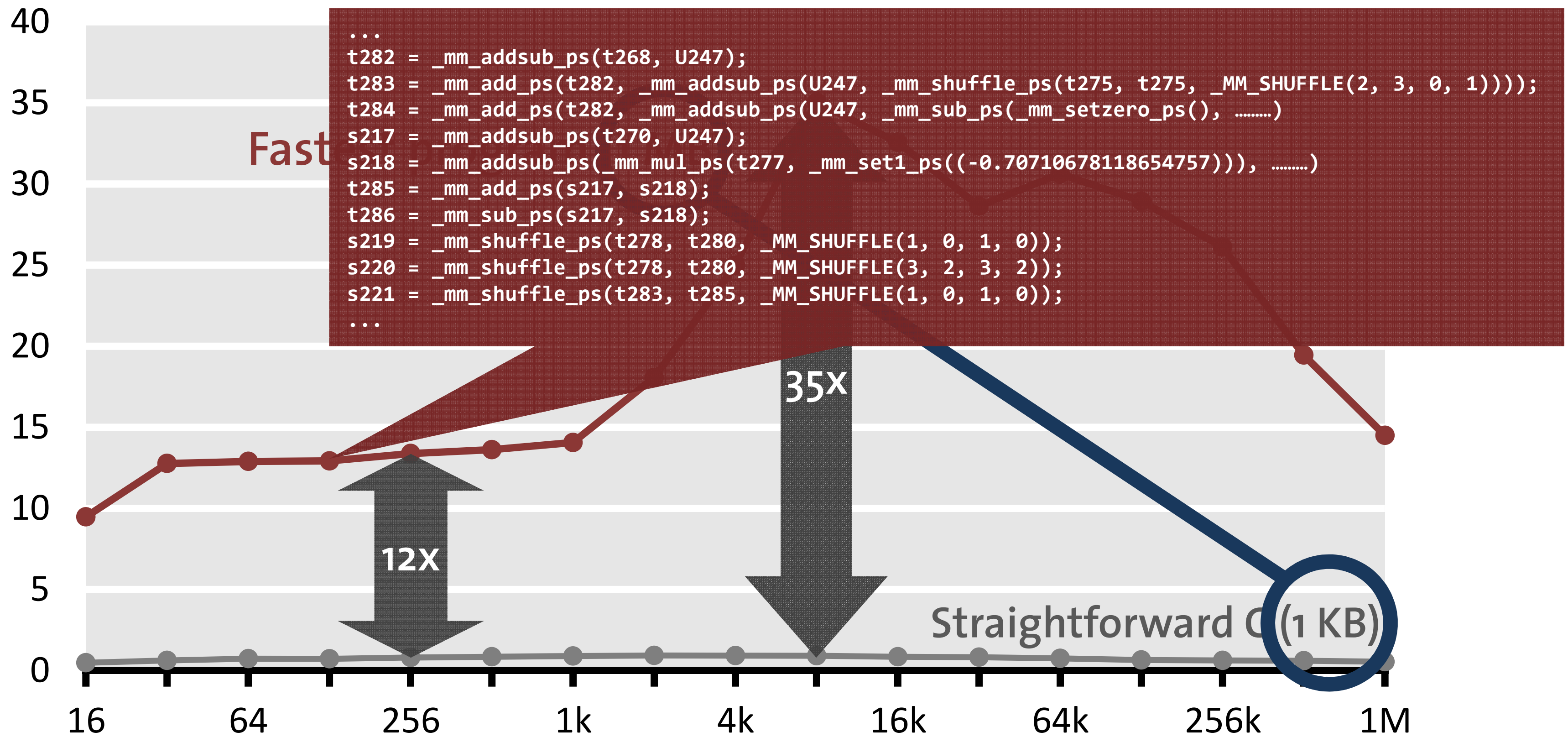
Discrete Fourier transform (single precision) on Intel Core i7 (4 cores)

Performance [Gflop/s]



Discrete Fourier transform (single precision) on Intel Core i7 (4 cores)

Performance [Gflop/s]

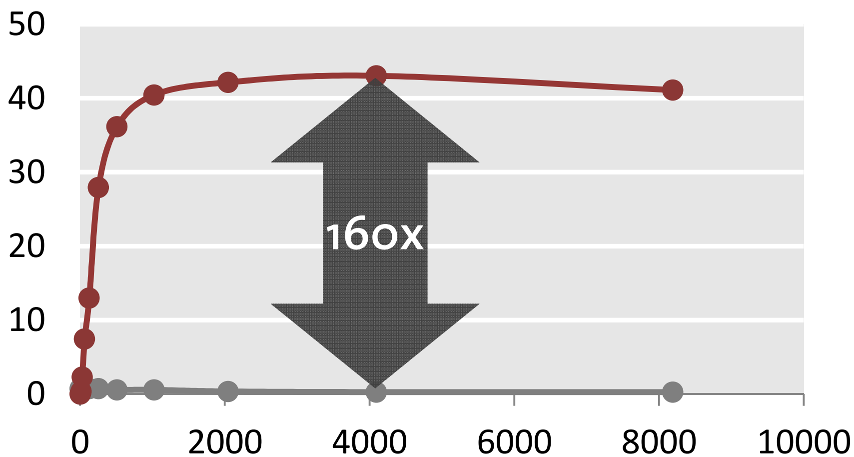


- Same opcount
- Best compiler

The Problem Is Everywhere

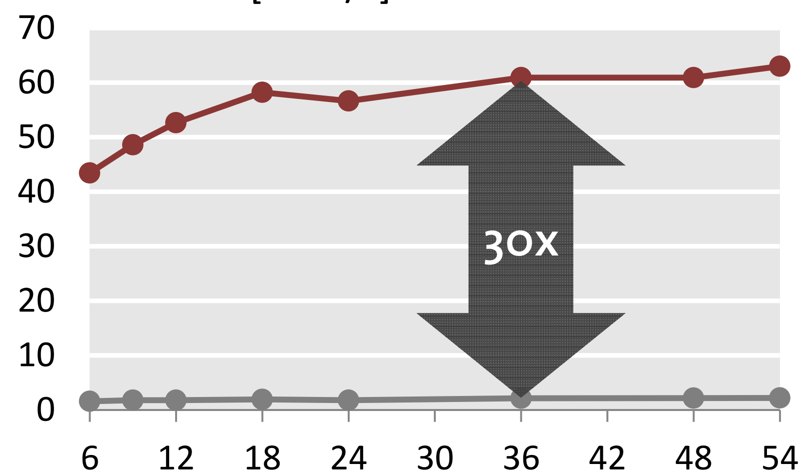
Matrix multiplication

Performance [Gflop/s]



WiFi Receiver

Performance [Mbit/s]



Model predictive control

Eigenvalues

LU factorization

Optimal binary search organization

Image color conversions

Image geometry transformations

Enclosing ball of points

Metropolis algorithm, Monte Carlo

Seam carving

SURF feature detection

Submodular function optimization

Graph cuts, Edmond-Karps Algorithm

Gaussian filter

Black Scholes option pricing

Disparity map refinement

Singular-value decomposition

Mean shift algorithm for segmentation

Stencil computations

Displacement based algorithms

Motion estimation

Multiresolution classifier

Kalman filter

Object detection

IIR filters

Arithmetic for large numbers

Optimal binary search organization

Software defined radio

Shortest path problem

Feature set for biomedical imaging

Biometrics identification

“Theorem:”

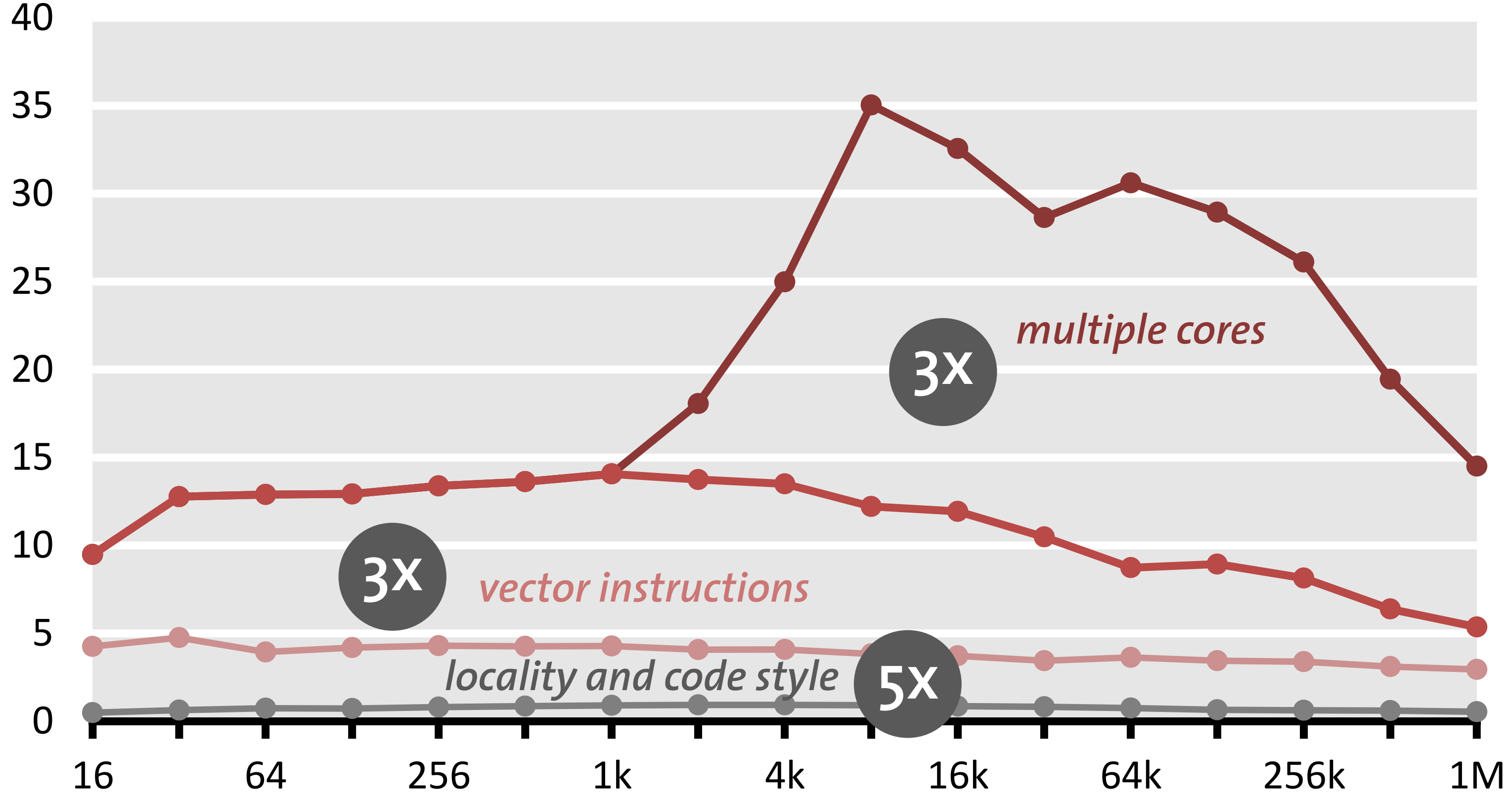
Let f be a mathematical function to be implemented on a state-of-the-art processor. Then

$$\frac{\textit{Performance of optimal implementation of } f}{\textit{Performance of straightforward implementation of } f} \approx 10\text{–}100$$

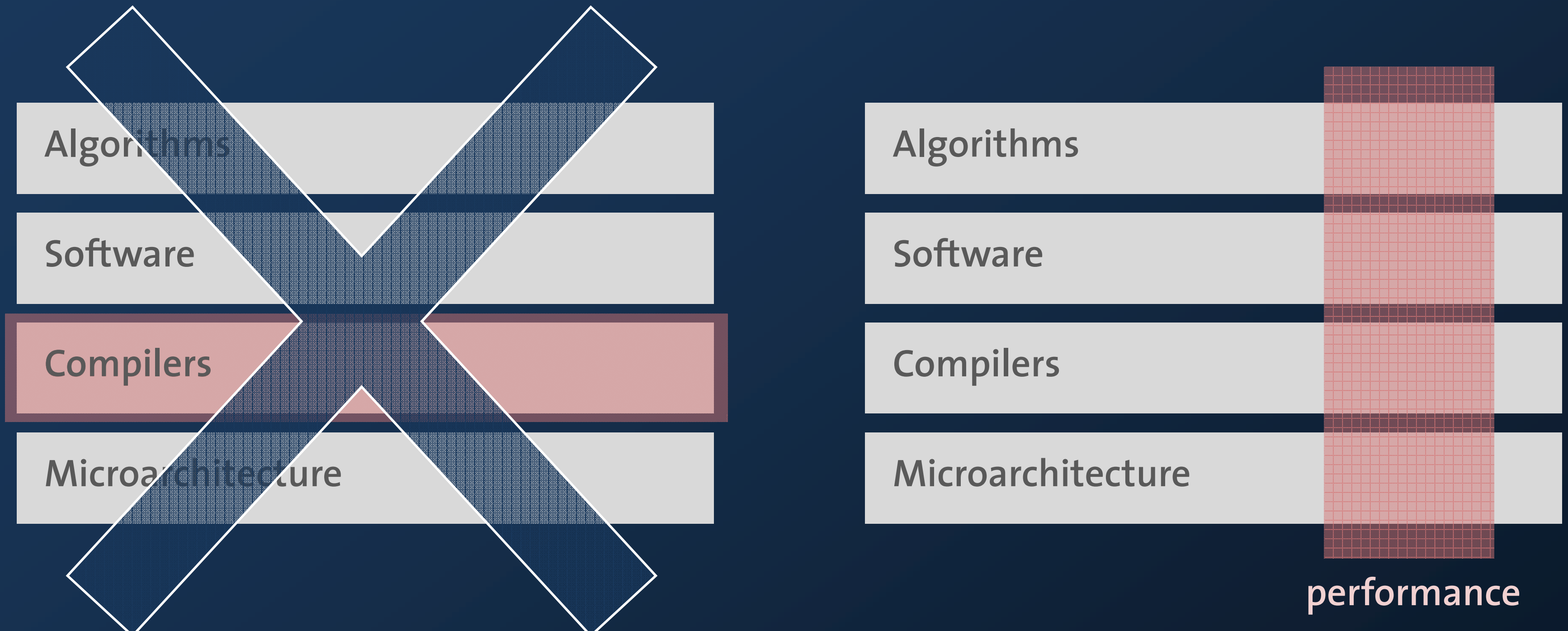
DFT: Analysis

Discrete Fourier transform (single precision) on Intel Core i7 (4 cores)

Performance [Gflop/s]

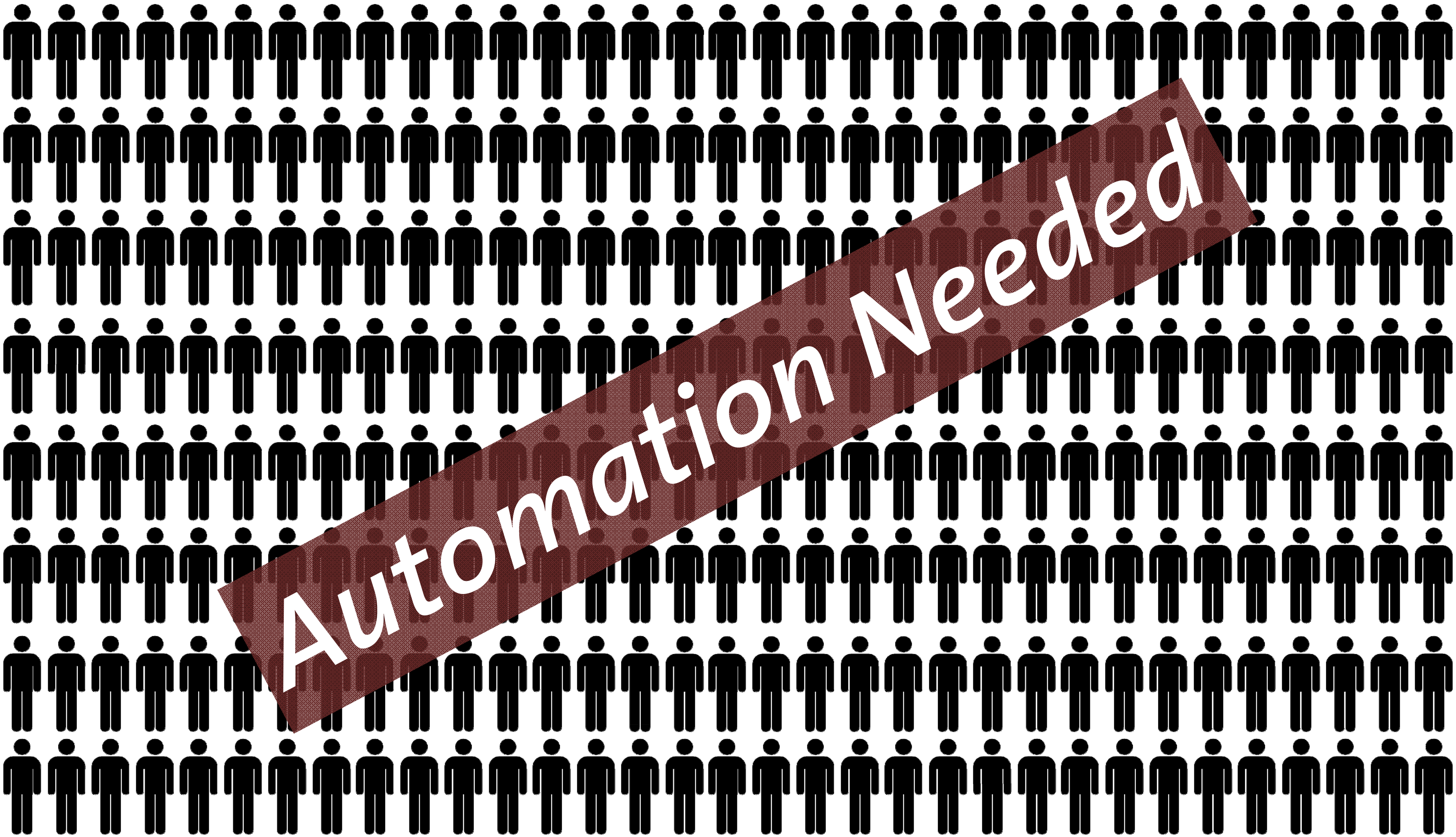


- **Compiler doesn't do the job**
- **Doing it by hand: requires guru knowledge**



Performance is different than other software quality features

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



Organization

- Software performance issues
- *Synthesis of fast mathematical libraries*
- Some benchmarks
- Conclusions

Goal:

Program synthesis of high performance components

“click”

Select convolutional code

Select a preset code or customize parameters

- *custom*
- Voyager
- NASA-DSN
- CCSDS/NASA-GSFC
- WiMax
- CDMA IS-95A
- LTE (3GPP - Long Term Evolution)
- UWB (802.15)
- CDMA 2000
- Cassini
- Mars Pathfinder & Stereo

rate	1 / 2	code rate (?)
K	7	constraint length (?)
polynomials	109 79	polynomials for the code in decimal notation (?)

Select implementation options

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

Generate Code   Reset

Viterbi Decoder

DFT IP Cores

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

Generate Verilog   Reset

@ www.spiral.net

Possible Approach:

Capturing algorithm knowledge:
Domain-specific languages (DSLs)

Structural optimization:
Rewriting systems

High performance code style:
Compiler

Decision making for choices:
Machine learning

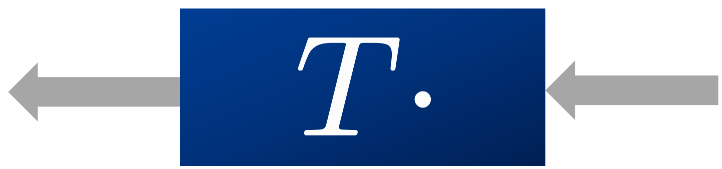
Spiral: Program Generation for Performance (www.spiral.net)

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

$$\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} = y = Tx$$



$$x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix}$$

Output

Input

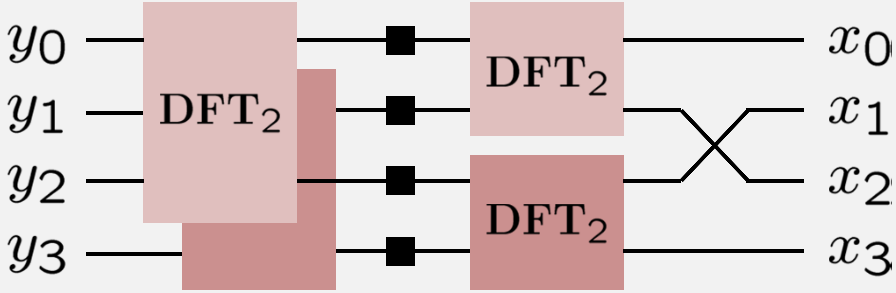
Example: $T = \mathbf{DFT}_n = [e^{-2k\ell\pi i/n}]_{0\le k,\ell<n}$

Algorithms: Example FFT, n = 4

Fast Fourier transform (FFT):

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

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)

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

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

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

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

$$\ldots \mathbf{FT}'_k)Q_{m/2,k},$$

Rules = algorithm knowledge

(≈100 journal papers)

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

SPL to Code

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

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

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

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

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

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

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

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)*

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

↻ locality optimization

↓

C Program

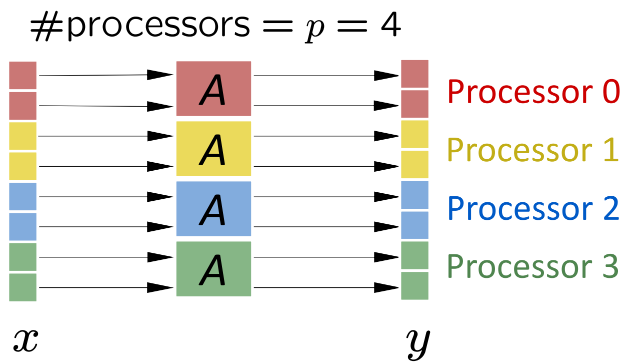
```
void sub(double *y, double *x) {
double f0, f1, f2, f3, f4, f7, f8, f10, f11;
    f0 = x[0] - x[3];
    f1 = x[0] + x[3];
    f2 = x[1] - x[2];
    f3 = x[1] + x[2];
    f4 = f1 - f3;
    y[0] = f1 + f3;
    y[2] = 0.7071067811865476 * f4;
    f7 = 0.9238795325112867 * f0;
< more lines>
```

↻ code style, code level optimization

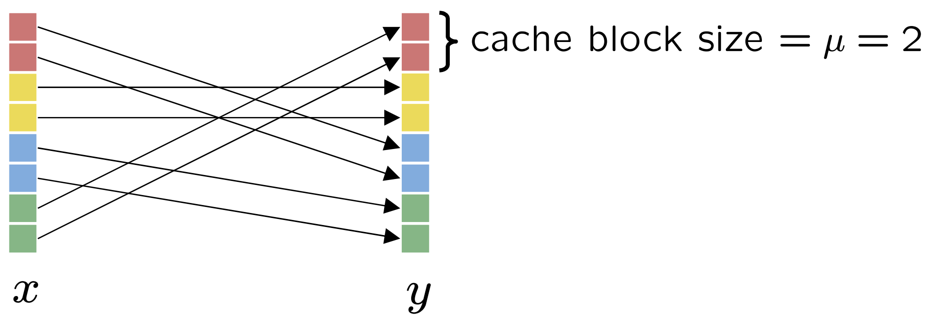
SPL to Shared Memory Code: Basic Idea

"Good" SPL structures

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



$$y = \left(P \otimes \mathrm{I}_\mu\right) x$$



Rewriting: Bad structures ➡ good structures

Example: SMP Parallelization

Franchetti et al., 2006

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

$$\dots$$

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

$$\dots$$

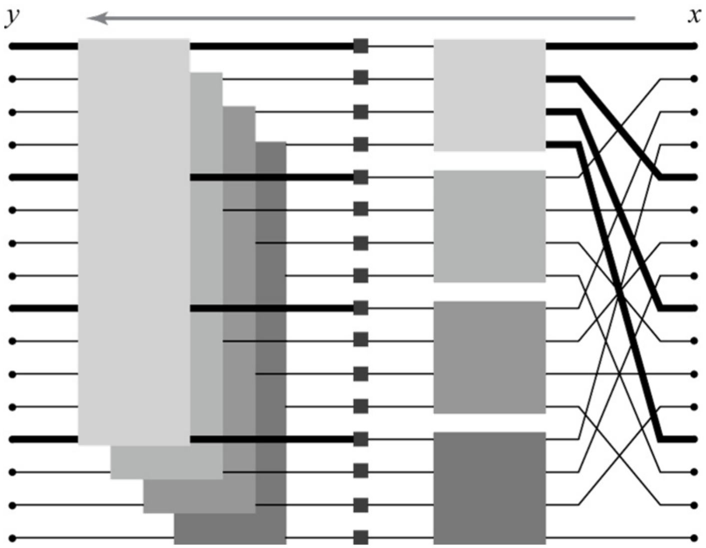
$$\to \underline{\left((\mathrm{L}_m^{mp} \otimes \mathrm{I}_{n/p\mu}) \otimes \mathrm{I}_\mu\right)} \underline{\left(\mathrm{I}_p \otimes (\mathbf{DFT}_m \otimes \mathrm{I}_{n/p})\right)} \underline{\left((\mathrm{L}_p^{mp} \otimes \mathrm{I}_{n/p\mu}) \otimes \mathrm{I}_\mu\right)}$$

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

load-balanced, no false sharing

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

Challenge: Recursive Composition

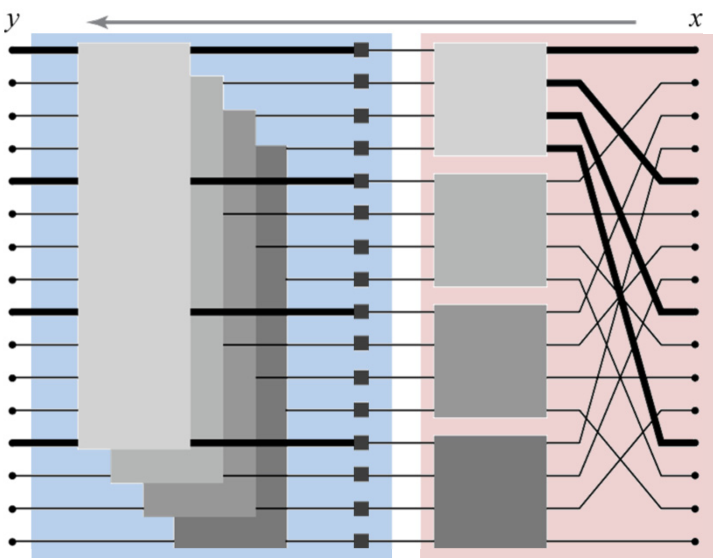


$$(\mathbf{DFT}_k \otimes \mathrm{I}_m)\, \mathsf{T}^{km}_m (\mathrm{I}_k \otimes \mathbf{DFT}_m)\, \mathsf{L}^{km}_k$$

16 = 4 x 4

```
void dft(int n, cpx *y, cpx *x) {
```

Challenge: Recursive Composition



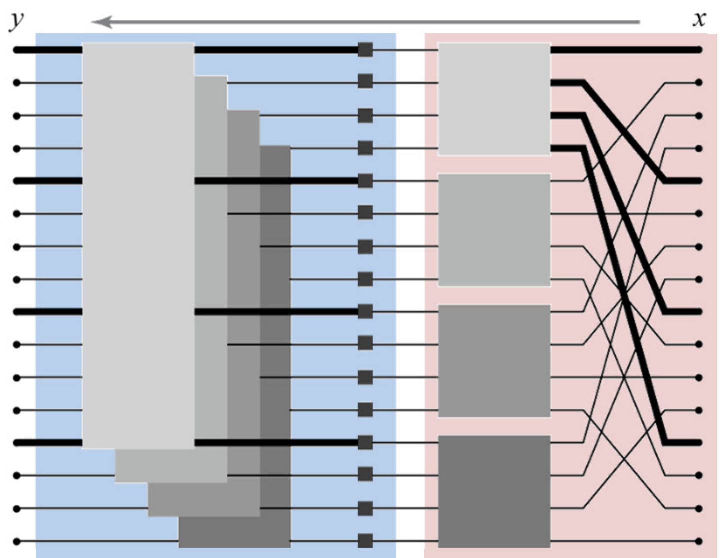
$$(\mathbf{DFT}_k \otimes \mathrm{I}_m)\, \mathsf{T}^{km}_m (\mathrm{I}_k \otimes \mathbf{DFT}_m)\, \mathsf{L}^{km}_k$$

16 = 4 x 4

```
void dft(int n, cpx *y, cpx *x) {
```

```
for (int i=0; i < k; ++i)
    dft_strided(m, k, t + m*i, x + m*i);
for (int i=0; i < m; ++i)
    dft_scaled(k, m, precomp_d[i], y + i, t + i);
```

Challenge: Recursive Composition



$$(\mathbf{DFT}_k \otimes \mathrm{I}_m)\, \mathsf{T}^{km}_m (\mathrm{I}_k \otimes \mathbf{DFT}_m)\, \mathsf{L}^{km}_k$$

16 = 4 x 4

```
void dft(int n, cpx *y, cpx *x) {
    if (use_dft_base_case(n))
        dft_bc(n, y, x);
    else {
        int k = choose_dft_radix(n);
        for (int i=0; i < k; ++i)
            dft_strided(m, k, t + m*i, x + m*i);
        for (int i=0; i < m; ++i)
            dft_scaled(k, m, precomp_d[i], y + i, t + i);
    }
}
```

How to discover these DFT variants?

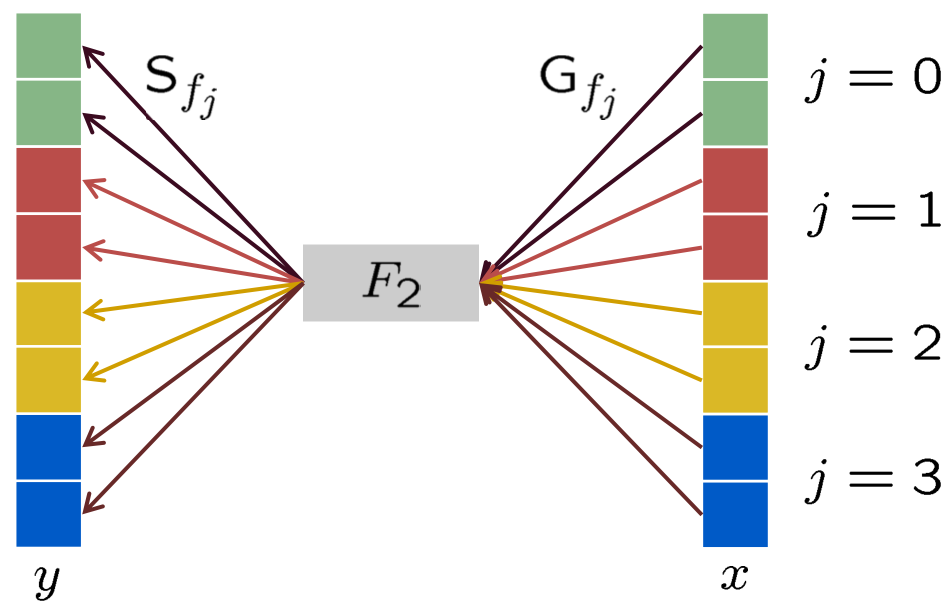
Σ–SPL : Basic Idea

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

- Σ (sum) — explicit loop
- G_f (gather) — load data with index mapping f
- S_f (scatter) — store data with index mapping f
- Perm_f — permute data with the index mapping f

Example: $y = (I_4 \otimes F_2)x \;\rightarrow\; y = \sum_{j=0}^{3} \mathsf{S}_{f_j}\, F_2\, \mathsf{G}_{f_j}\, x$

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



Find Recursion Step Closure

Voronenko, 2008

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

$$(\{\mathbf{DFT}_{n/k}\} \otimes I_k) T_k^n (I_{n/k} \otimes \{\mathbf{DFT}_k\}) L_{n/k}^n$$

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

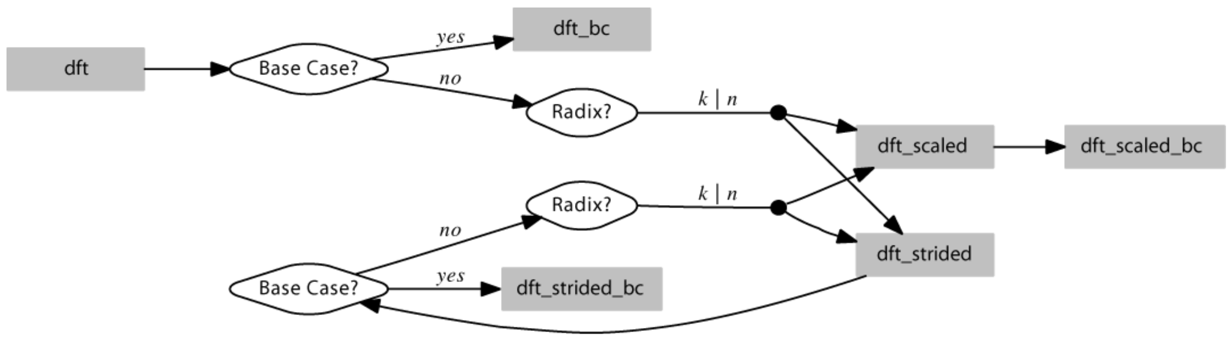
$$\sum_{i=0}^{k-1} \mathsf{S}_{h_{i,k}} \{\mathbf{DFT}_{n/k}\}\, \mathsf{diag}(f \circ h_{i,k})\, \mathsf{G}_{h_{i,k}} \sum_{j=0}^{n/k-1} \mathsf{S}_{h_{jk,1}} \{\mathbf{DFT}_k\}\, \mathsf{G}_{h_{j,n/k}}$$

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

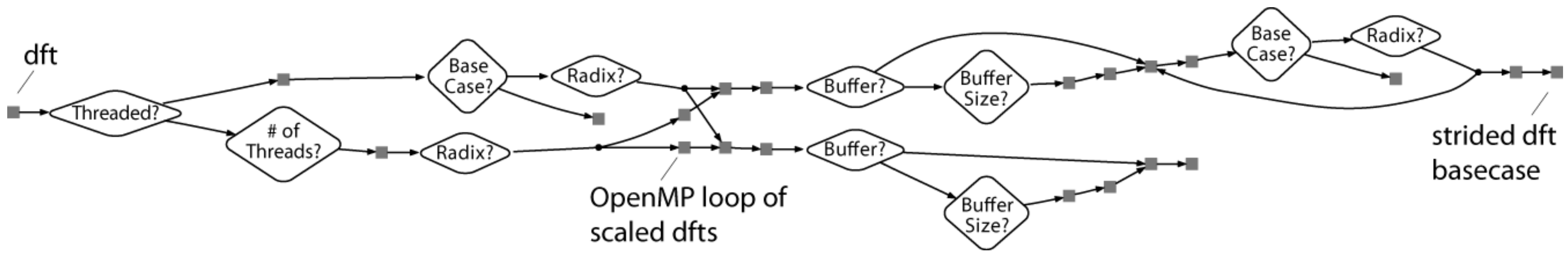
Repeat until closure

Recursion Step Closure: Examples

DFT: scalar code



DFT: full-fledged (vectorized and parallel code)



Online tuning

Installation

`configure/make`

Use

`d = dft(n)`
`d(x,y)`

Search for fastest recursion

Offline tuning

Installation

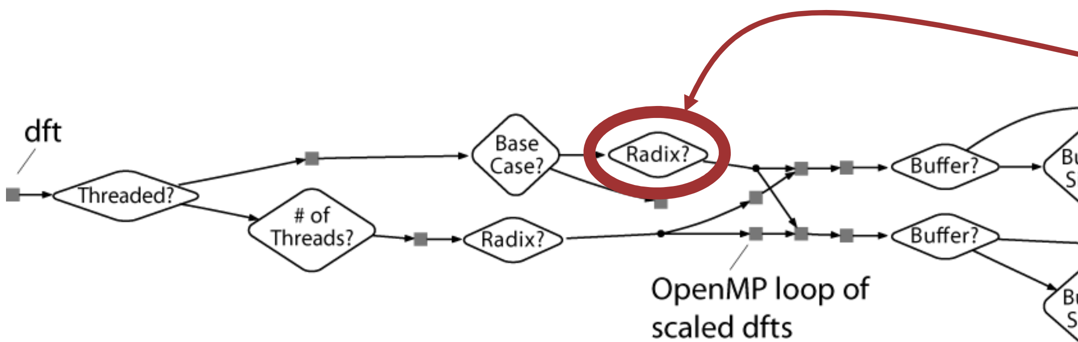
`configure/make`

for a few n: search
learn decision trees

Use

`d = dft(n)`
`d(x,y)`

Machine learning (e.g., C4.5)




```
if ( n <= 65536 ) {
  if ( n <= 32 ) {
    if ( n <= 4 ) {return 2;}
    else {return 4;}
  }
  else {
    if ( n <= 1024 ) {
      if ( n <= 256 ) {return 8;}
      else {return 32;}
    }
    else {
```

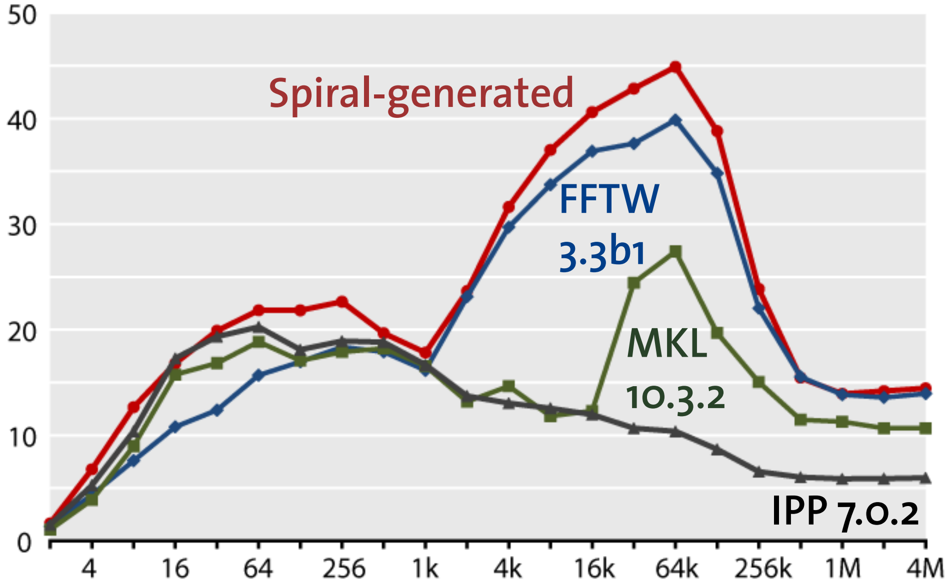
Organization

- Software performance issues
- Synthesis of fast mathematical libraries
- *Some benchmarks*
- Conclusions

It Really Works

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

Performance [Gflop/s]



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

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

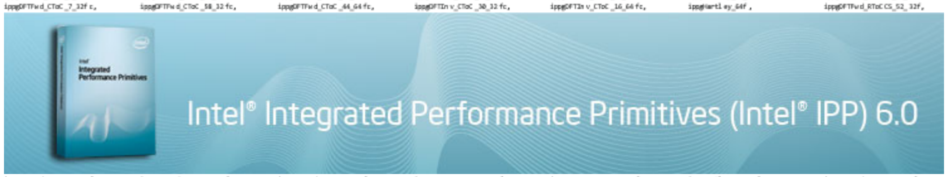
$$\mathbf{RDFT}_n \to (P^\top_{k/2,m} \otimes I_2) \left(\mathbf{RDFT}_{2m} \oplus \left(I_{k/2-1} \otimes_i D_{2m}\, \mathbf{rDFT}_{2m}(i/k)\right)\right) (\mathbf{RDFT}_k \otimes I_m)$$

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

5MB vectorized, threaded, general-size, adaptive library

- Many transforms, often the generated code is best
- Vector, shared/distributed memory parallel, FPGAs

Computer generated Functions for Intel IPP 6.0



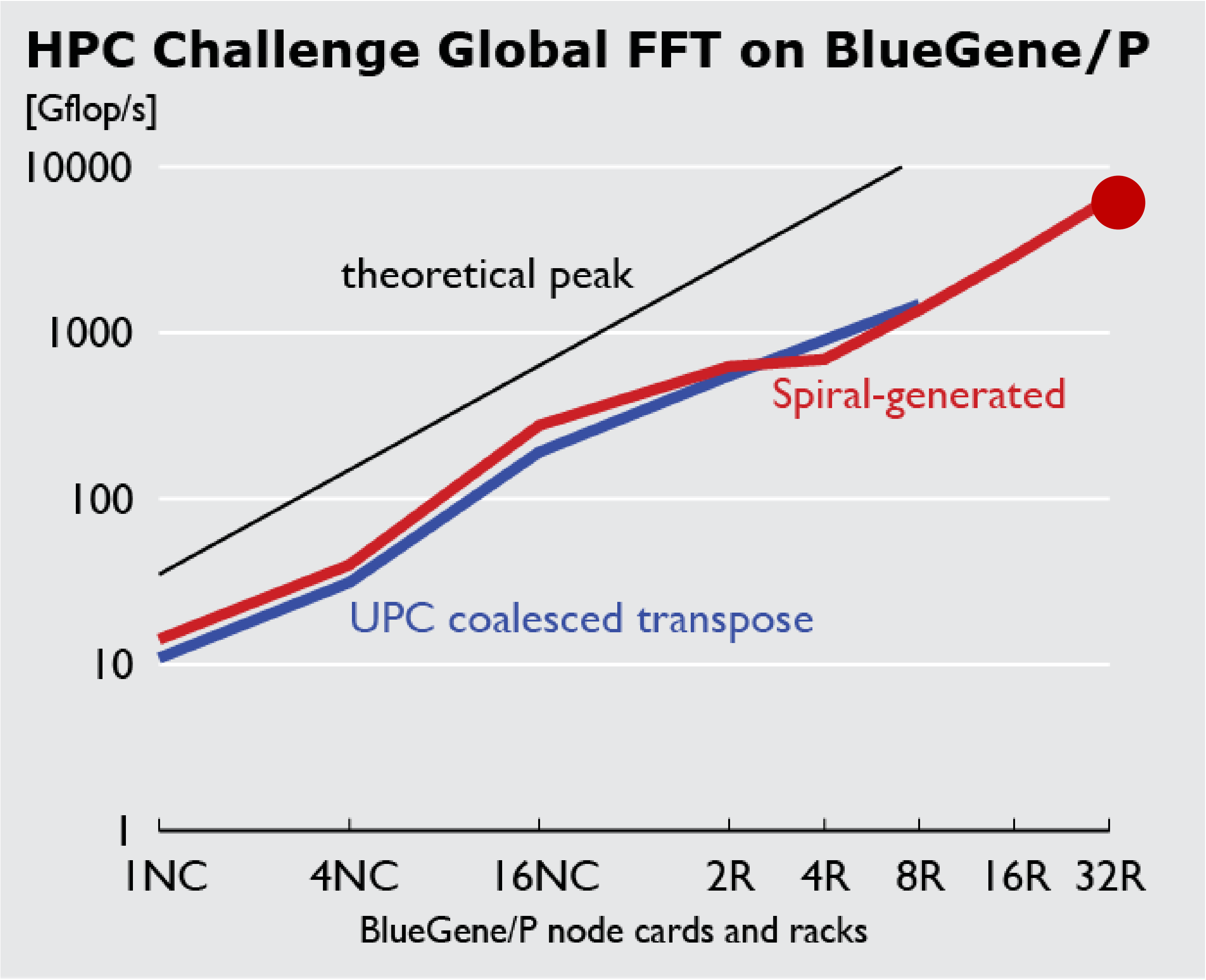
3984 C functions
1M lines of code

Transforms: DFT (fwd+inv), RDFT (fwd+inv), DCT2, DCT3, DCT4, DHT, WHT
Sizes: 2–64 (DFT, RDFT, DHT); 2-powers (DCTs, WHT)
Precision: single, double
Data type: scalar, SSE, AVX (DFT, DCT), LRB (DFT)

Computer generated

Results: SpiralGen Inc.

Very Large Scale: BG/P



6.4 Tflop/s

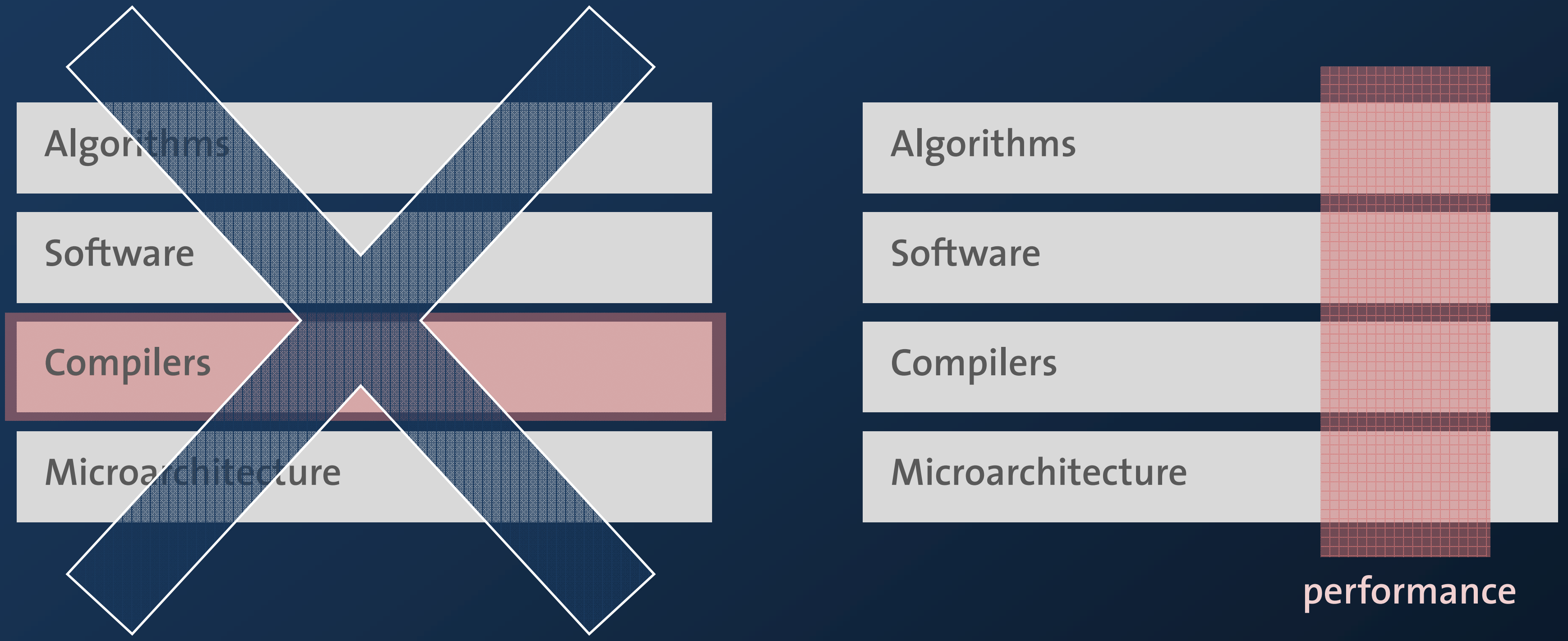
32 racks
= 32K node cards
= 128K cores

Related Work

- **Program generators for performance**
  - FFTW codelet generator (Frigo)
  - Flame (van de Geijn, Quintana-Orti, Bientinesi, …)
  - Tensor contraction engine (Baumgartner, Sadayappan, Ramanujan, …)
  - cvxgen (Mattingley, Boyd)
  - PetaBricks (Ansel, Amarasinghe, …)
  - Spiral
- **Metaprogramming**
  - Eigen
  - Pochoir
- **Autotuning**
  - ATLAS/PhiPAC (Whaley, Bilmes, Demmel, Dongarra, …)
  - FFTW adaptive library (Frigo, Johnson)
  - OSKI (Vuduc et al.)
  - Adaptive sorting (Li et al.)

Organization

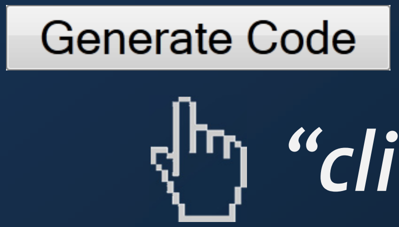
- Software performance issues
- Synthesis of fast mathematical libraries
- Some benchmarks
- *Conclusions*



We need tools aiding the programmer or compiler in achieving performance

Spiral

Program synthesis for performance



"click"

Principles

Capturing algorithm knowledge:
DSLs

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

Structural optimization:
Rewriting

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

Decision making:
Search and learning

More information: www.spiral.net