$A \otimes I_n$

Automatic Performance Programming?

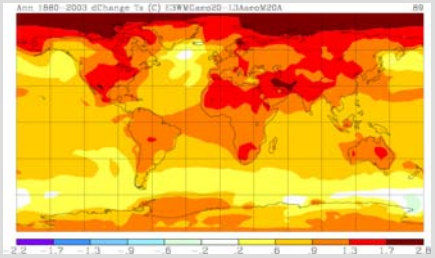
Markus Püschel
Computer Science

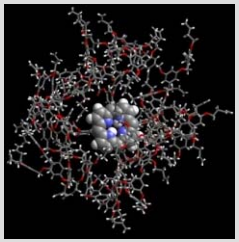
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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



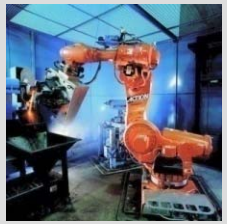


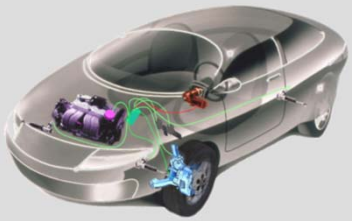
Simulations: Physics, Biology, …

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*

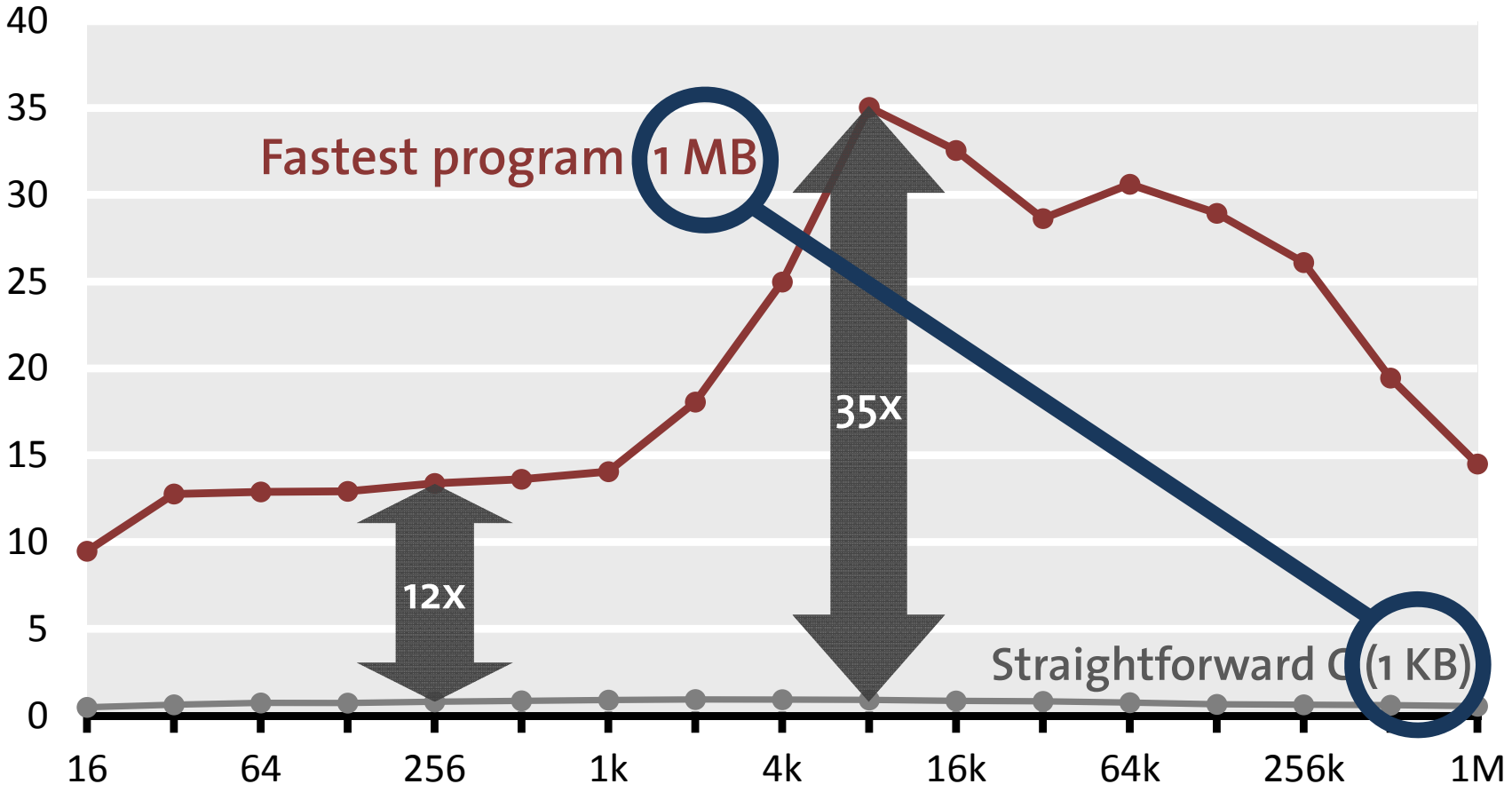
Microarchitecture

How well does that work?

The Problem: Example Fourier Transform

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

Performance [Gflop/s]

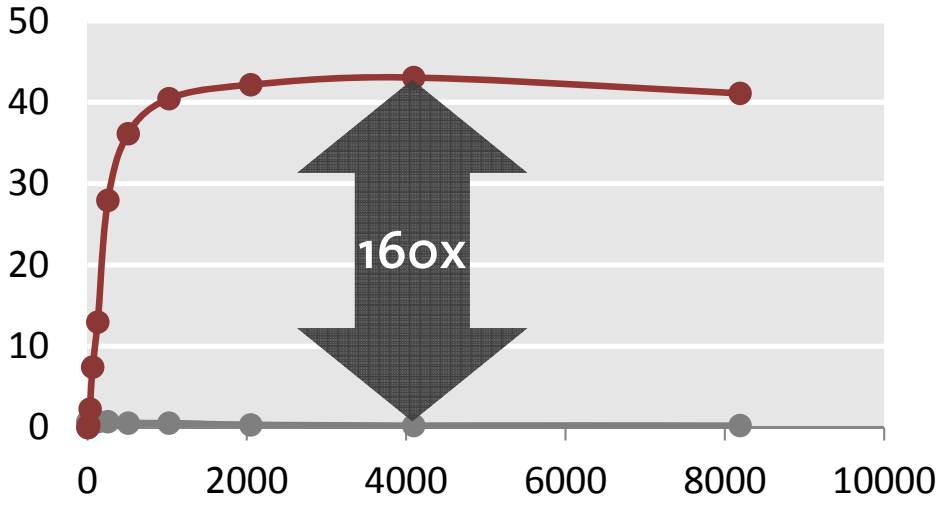


- Same opcount
- Best compiler

The Problem Is Everywhere

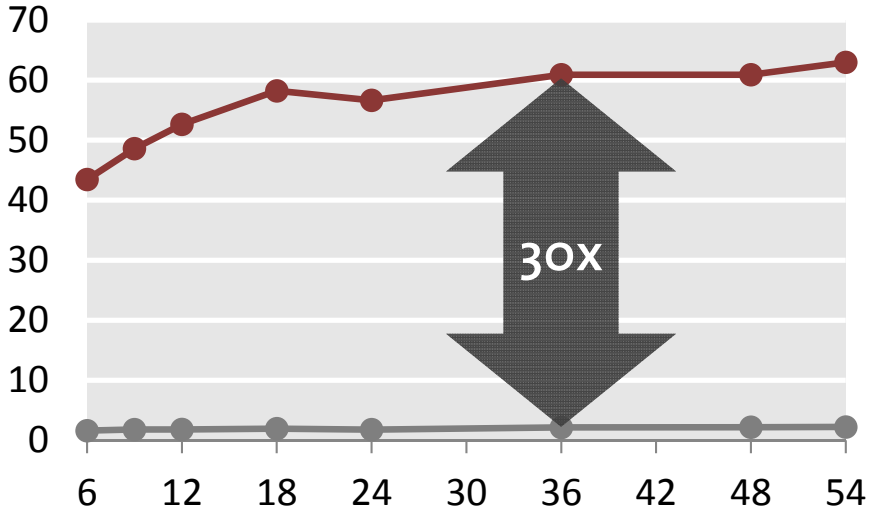
Matrix multiplication

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

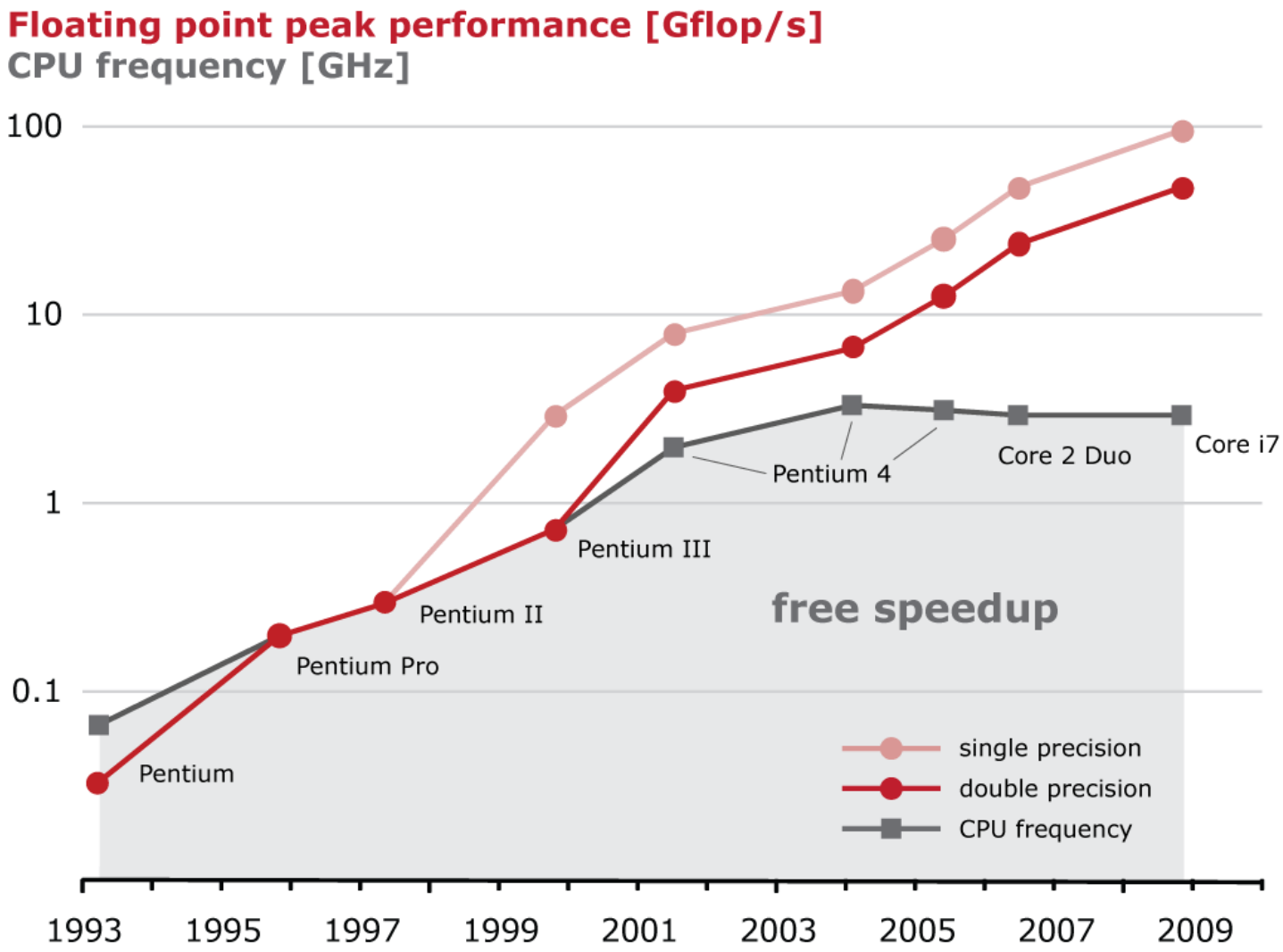
Biometrics identification

“Theorem:”

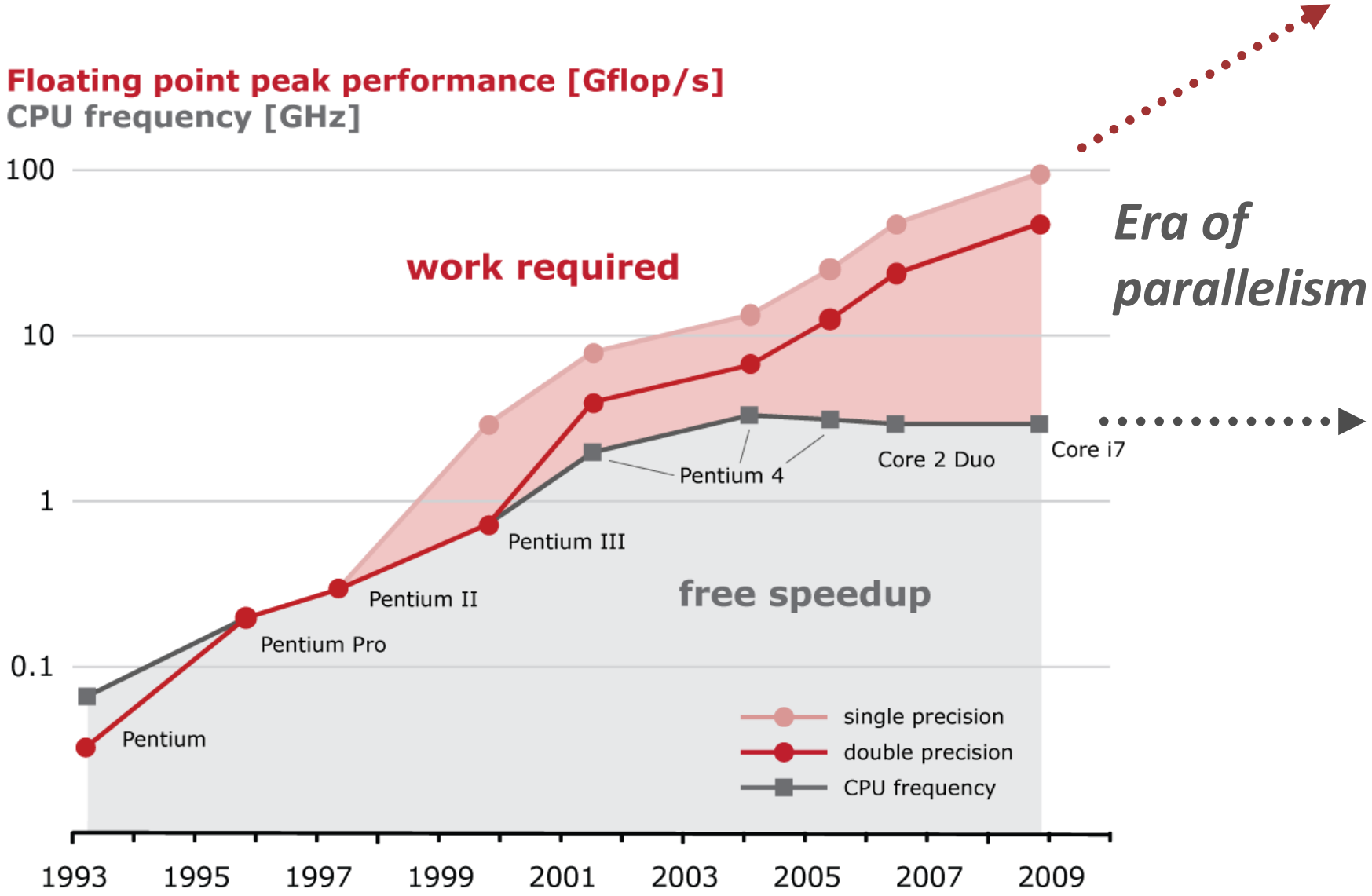
Let f be a mathematical function to be implemented on a state-of-the-art processor. Then

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

Evolution of Processors (Intel)

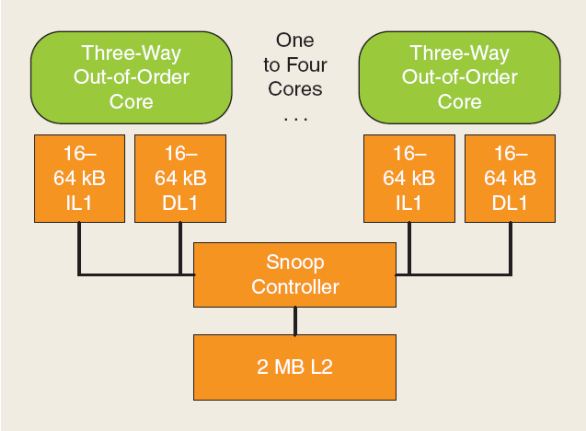


The End of Automatic Speedup

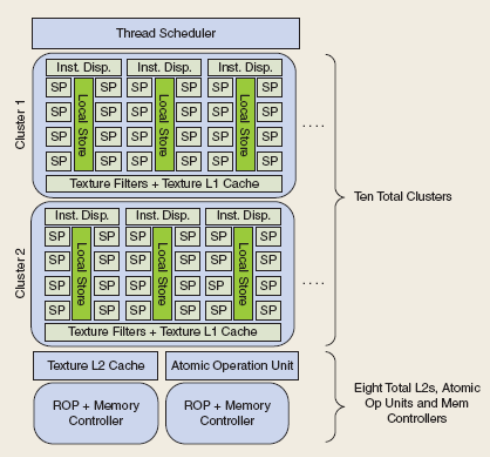


And There Will Be Variety …

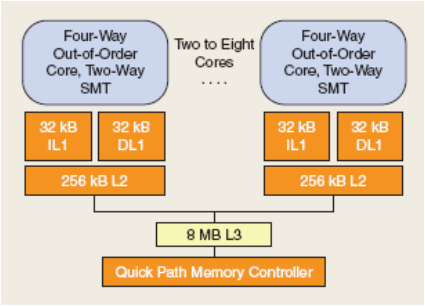
Arm Cortex A9



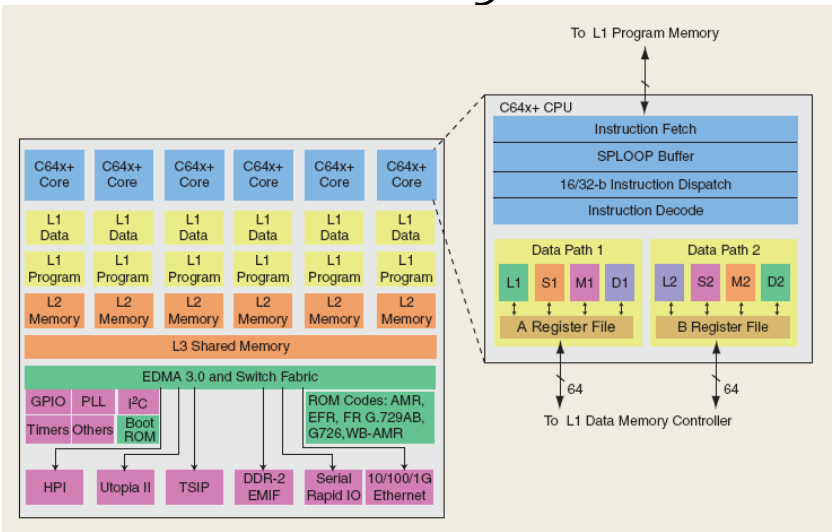
Nvidia G200



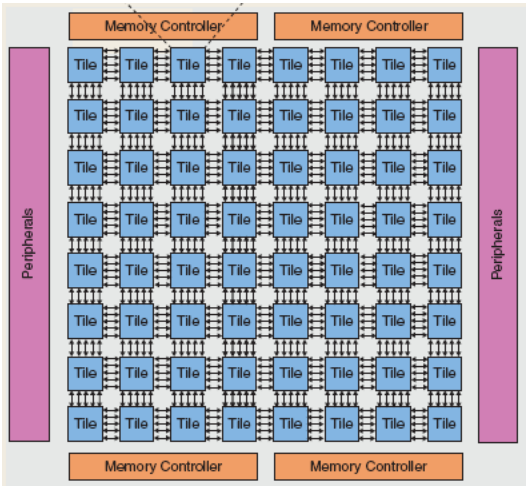
Core i7



TI TNETV3020



Tilera Tile64

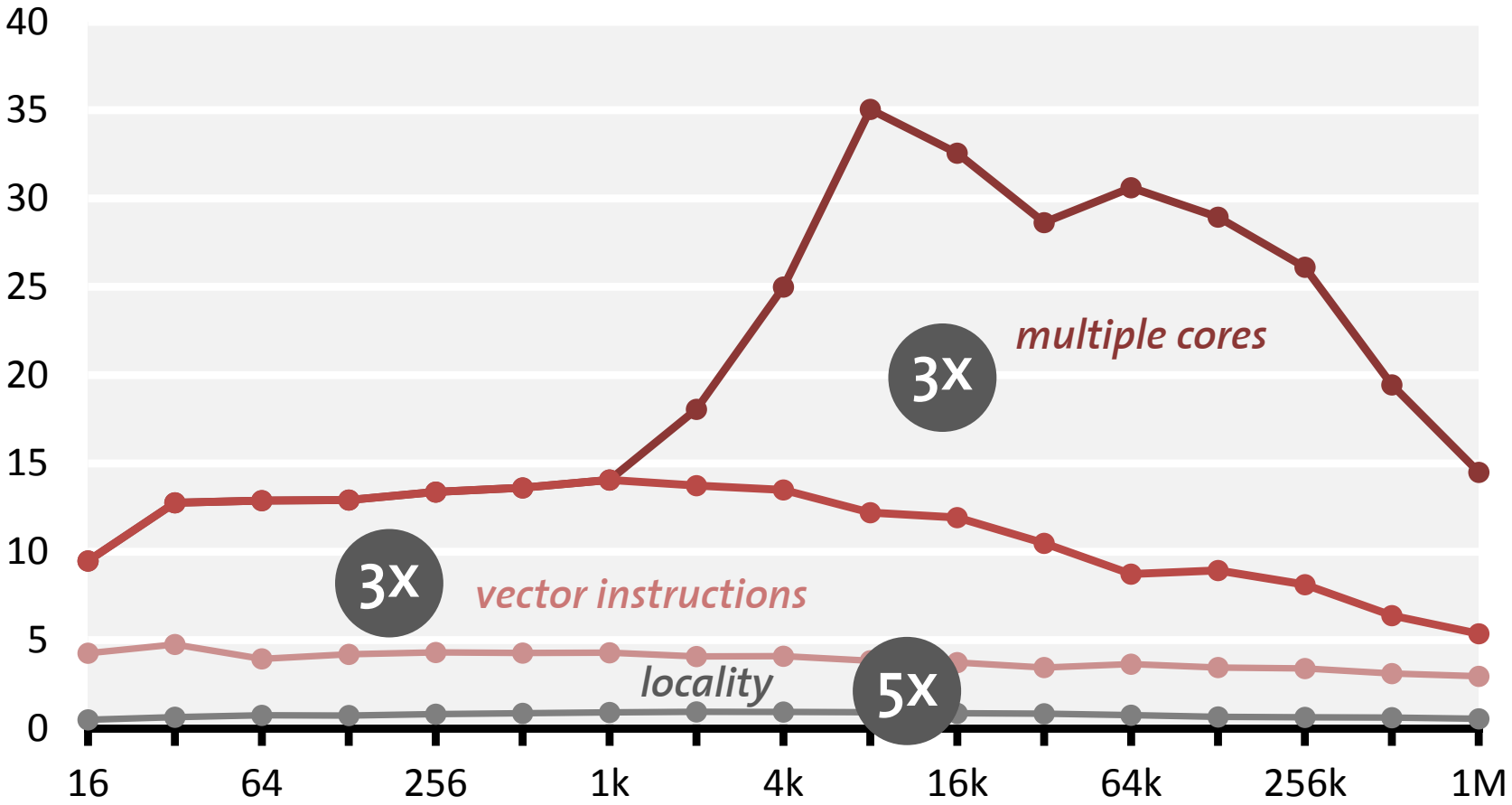


Source: IEEE SP Magazine, Vol. 26, November 2009

DFT: Analysis

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

Performance [Gflop/s]



- Compiler doesn't do the job (why?)
- Doing it by hand: requires guru knowledge

Optimization for Register Locality and ILP

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

Removes aliasing

Enables register allocation and instruction scheduling

Compiler typically does not do:

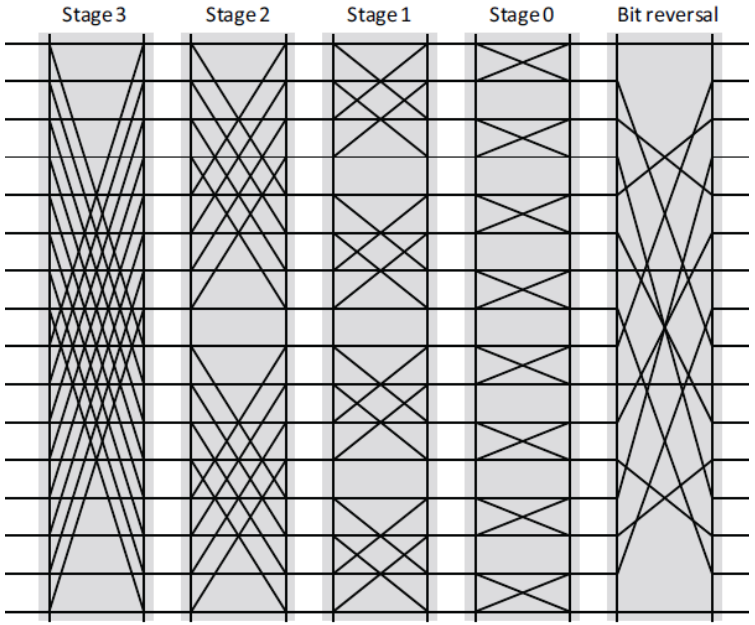
- often illegal
- many choices

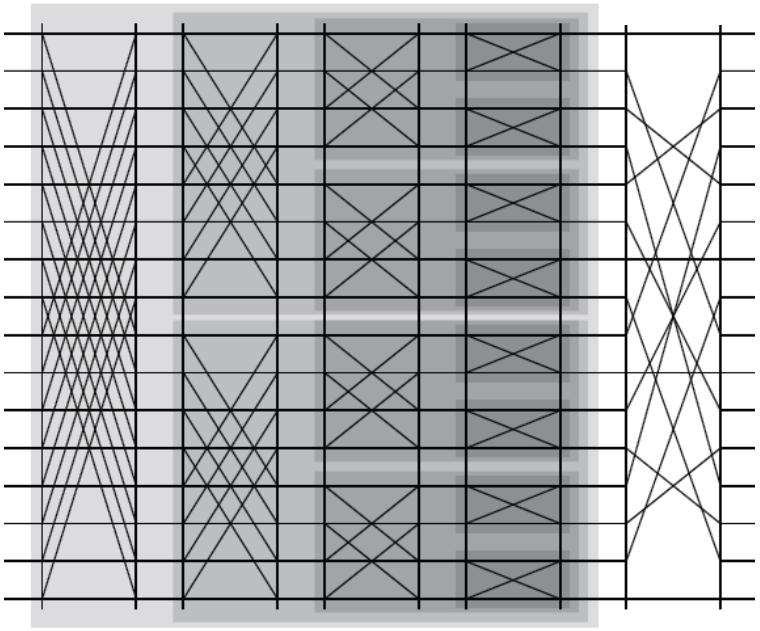
```
// unrolling + scalar replacement
for(i = 0; i < N; i += MU) {
    for(j = 0; j < N; j += NU) {
      for(k = 0; k < N; k += KU) {
        t1 = A[i*N + k];                  } load
        t2 = A[i*N + k + 1];
        t3 = A[i*N + k + 2];
        t4 = A[i*N + k + 3];
        t5 = A[(i + 1)*N + k];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

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

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

Optimization for Cache Locality





Many algorithms are formulated iteratively:

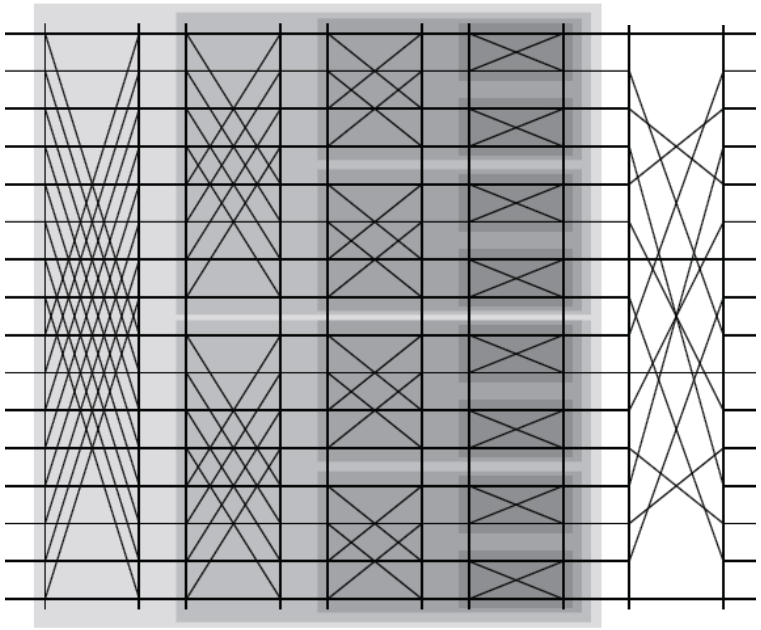
- many passes through data
- poor locality
- poor performance

Restructured for locality

Compiler usually does not do

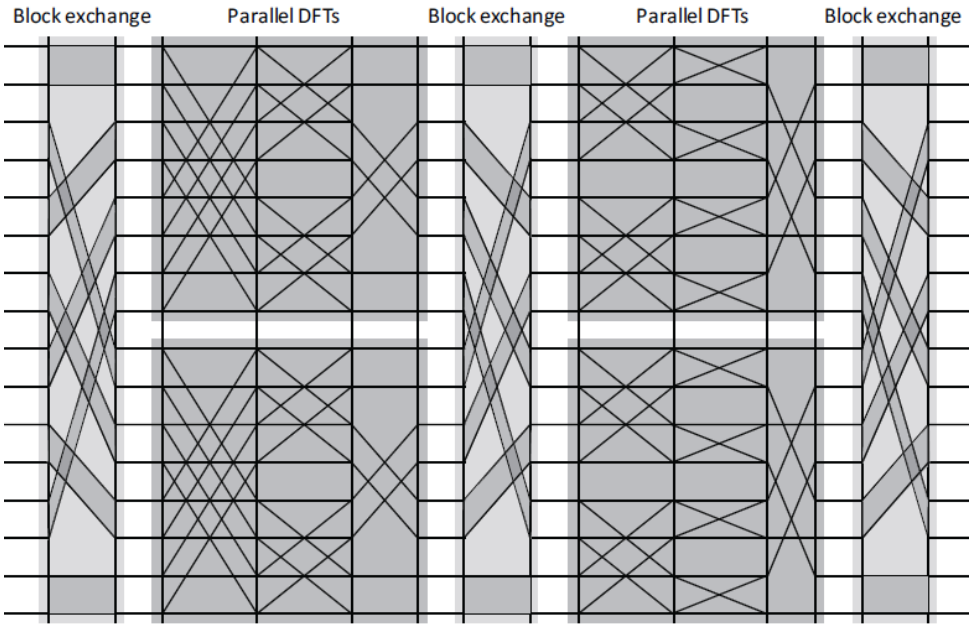
- analysis may be unfeasible
- possibly many choices
- may require algorithm changes
- may require domain knowledge
- may require cache parameters

Optimization for Parallelism (Threads)



Restructured for locality

Parallelism is present, but is not in the “right shape”



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

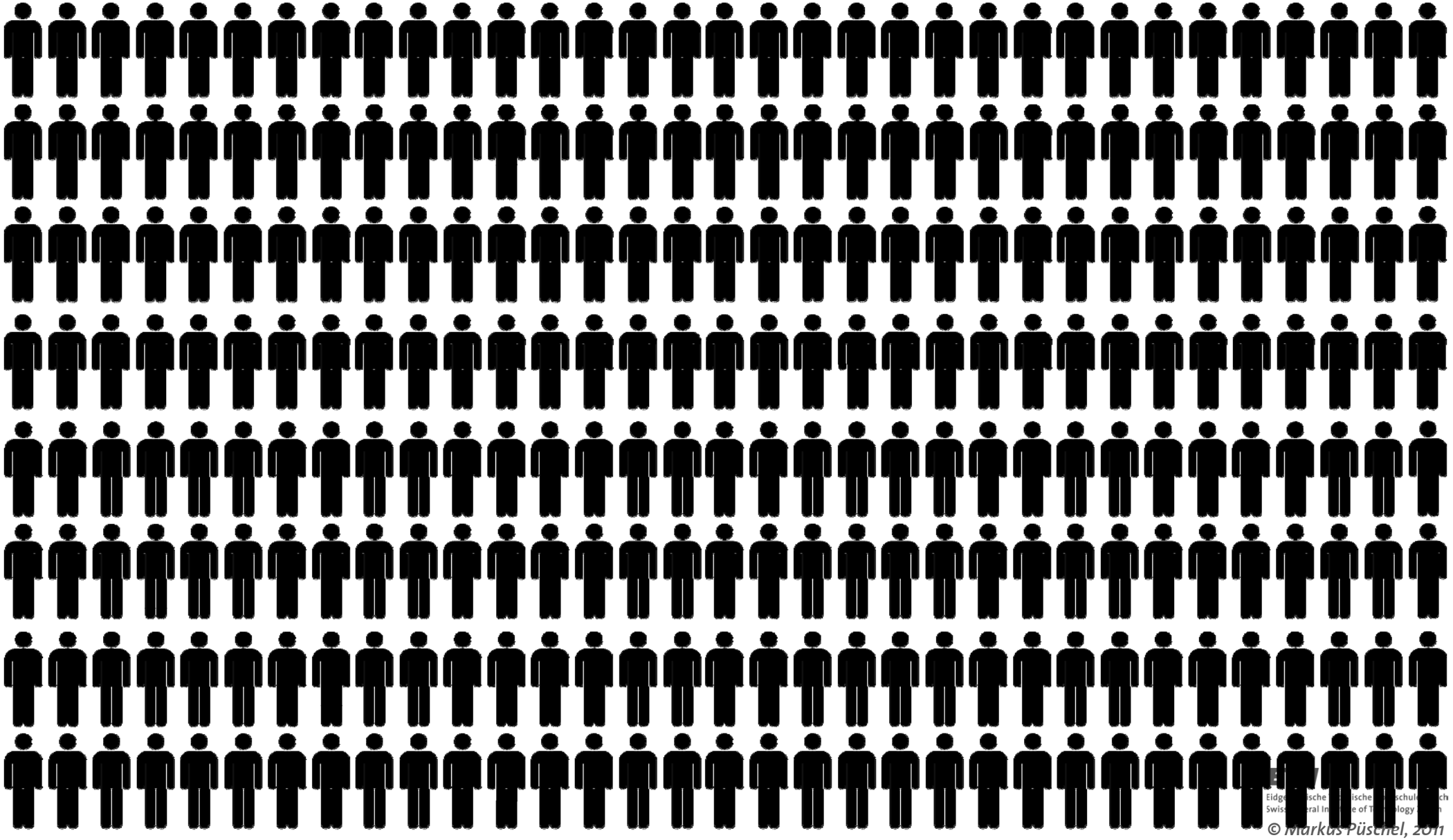
Compiler usually does not do

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

Software Performance: Facts

- Straightforward code is often slow
- Inherent compiler limitations
- End of free speedup for legacy code
- Performance optimization is very hard
- Fast code violates good software engineering practices
- Performance optimization is “vertical”
  - *algorithm changes*
  - *code style*
  - *considers microarchitectural parameters*
- Highest performance is generally non-portable

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



Algorithms

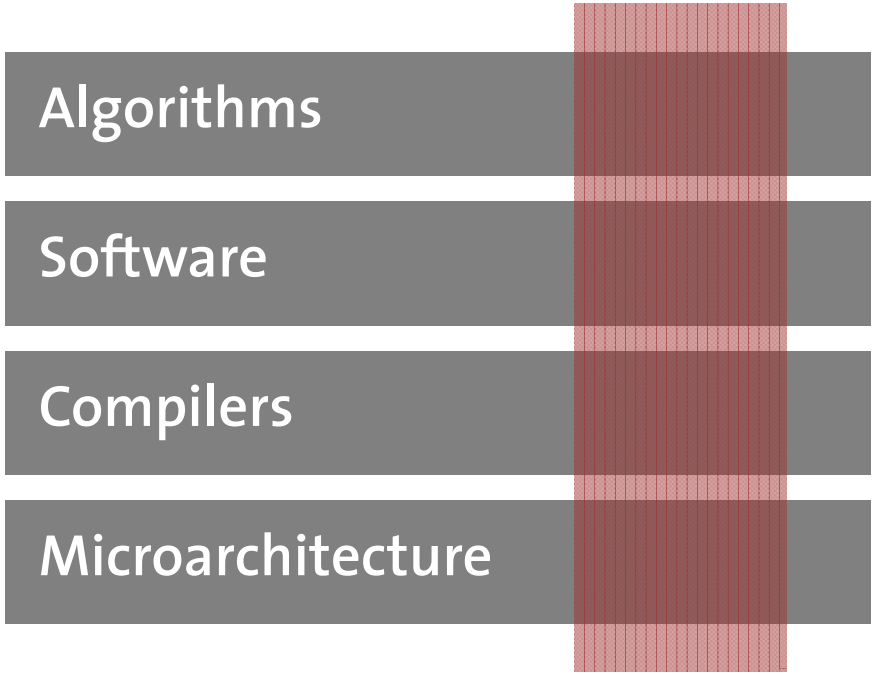
optimal op count

Software

Compilers

performance optimization

Microarchitecture



Performance is different than other software quality features

We need

- Courses on software performance
- Microarchitecture-cognizant algorithm analysis
  (e.g., Kung's work, cache oblivious algorithms, ...)
- Smarter compilers (e.g., iterative compilation, machine learning)
- *"Vertical" software engineering and programming languages techniques for performance optimization*

Organization

- Software performance issues
- *Automatic performance tuning (autotuning)*
- Automatic performance programming
- Conclusions

ATLAS/PhiPAC: MMM Generator

Bilmes et al. 97, Whaley et al. 98

```
// MMM loop-nest
for i = 0:N_B:N-1
   for j = 0:N_B:M-1
      for k = 0:N_B:K-1
```

- *ijk or jik depending on N and M*
- *Blocking for cache*

```
// mini-MMM loop nest
for i' = i:M_U:i+N_B-1
   for j' = j:N_U:j+N_B-1
      for k' = k:K_U:k+N_B-1
```

- *Blocking for registers*

```
// micro-MMM loop nest
for k" = k':1:k'+K_U-1
   for i" = i':1:i'+M_U-1
      for j" = j':1:j'+N_U-1
```

- *Unrolling*
- *Scalar replacement*
- *Add/mult interleaving*
- *Skewing*

Search parameters: N_B, M_U, N_U, K_U, L_S, …

Offline tuning: tuning at installation

FFTW: Adaptive DFT Library

Frigo, Johnson 98, 05

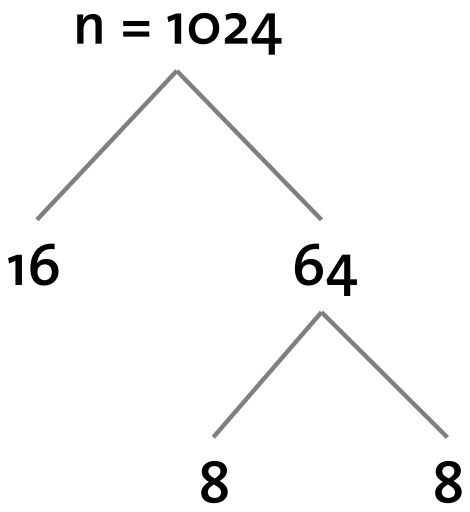
Installation

```
configure/make
```

Usage

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

Search for fastest computation strategy



Online tuning: tuning at time of use

Autotuning

- **Use of models**
  - OSKI (Vuduc, Demmel, Yelick et al.)
  - Adaptive sorting (Li, Padua et al.)
- **Tools**
  - Iterative compilation (Knijnenburg, O'Boyle, Cooper, ...)
  - Machine learning in compilation (O'Boyle, Cavazos, ...)
  - Source code tools (POET, Rose)

Most common form of autotuning: **Search over alternative implementations to find the fastest for a given platform**

Problem: usually parameter based

- **not portable to new types of platforms (e.g., parallel)**
- **not portable to different functions**

Organization

- Software performance issues
- Automatic performance tuning (autotuning)
- *Automatic performance programming*
- Conclusions

Goal:

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

Possible Approach:

Capturing algorithm knowledge:
Domain-specific languages (DSLs)

Structural optimization:
Rewriting systems

High performance code style:
Compiler

Decision making for choices:
Machine learning

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

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

funding: DARPA, NSF, ONR, Intel

Linear Transforms

Output *Input*

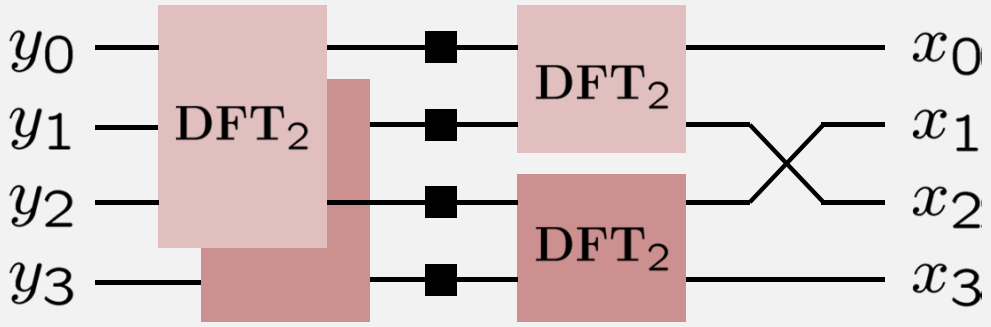
Example: $T = \mathbf{DFT}_n = [e^{-2k\ell\pi i/n}]_{0 \le k,\ell < n}$

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)

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

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

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

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

$$\mathbf{DCT\text{-}3}_n \to P^{\top}\left(\mathbf{DCT\text{-}3}_{\cdots} \oplus \left(I_{\cdots} \otimes N_{\cdots}\,\mathbf{RDFT\text{-}3}^{\top}_{\cdots}\right)\right) R_{\cdots}\left(I^{n/2} \otimes I_2\right)\left(I_{\cdots} \otimes \mathbf{RDFT}'_2\right) Q_{\cdots}$$

Rules = algorithm knowledge

(≈100 journal papers)

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

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^{km}_k	`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)*

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

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>
```

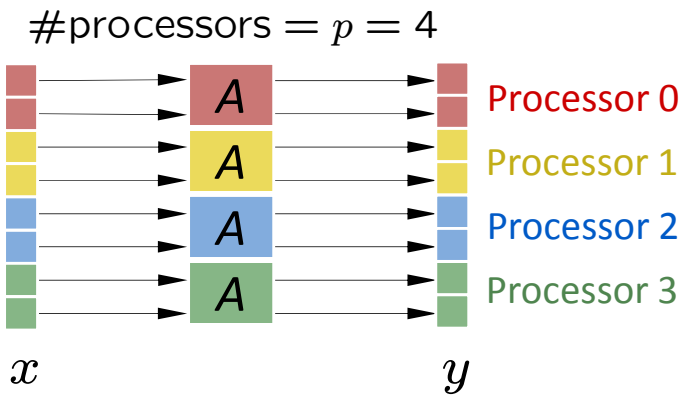
code style
code level
optimization

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

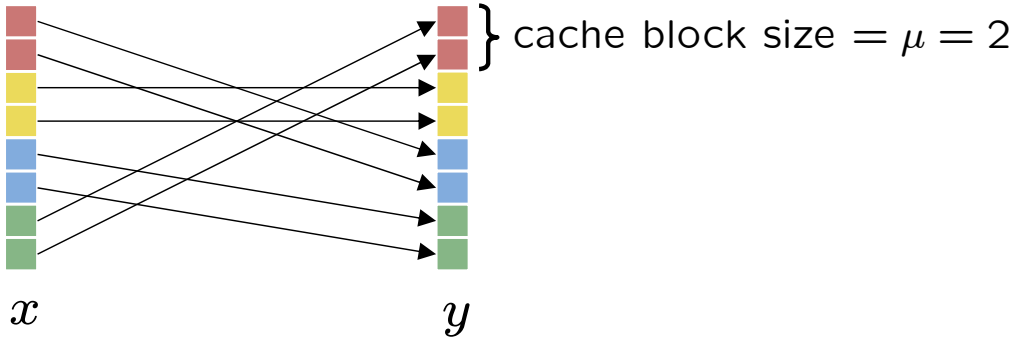
SPL to Shared Memory Code: Basic Idea

"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

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

$$\ldots$$

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

$$\ldots$$

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

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

load-balanced, no false sharing

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

Challenge: General Size Libraries

So far:

Code specialized to fixed input size

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

- Algorithm fixed (offline tuning)
- Nonrecursive code

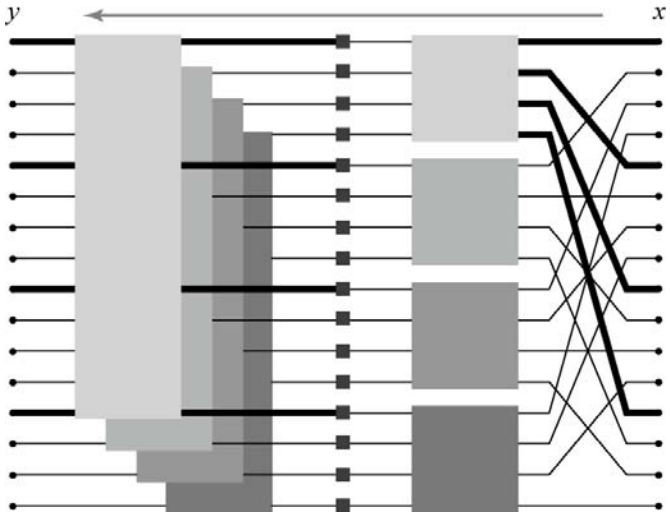
Challenge:

Library for general input size

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

- Algorithm cannot be fixed (online tuning)
- Recursive code
- Creates many challenges

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) {
```

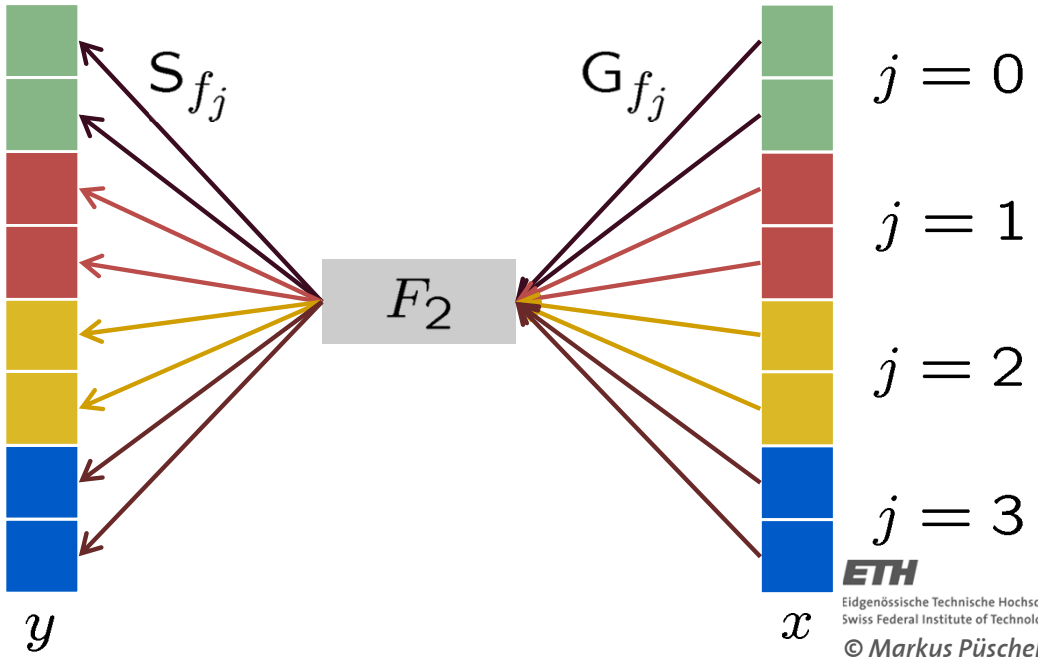
Σ–SPL : Basic Idea

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

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



Find Recursion Step Closure

Voronenko, 2008

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

$$\downarrow$$

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

$$\downarrow$$

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

$$\downarrow$$

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

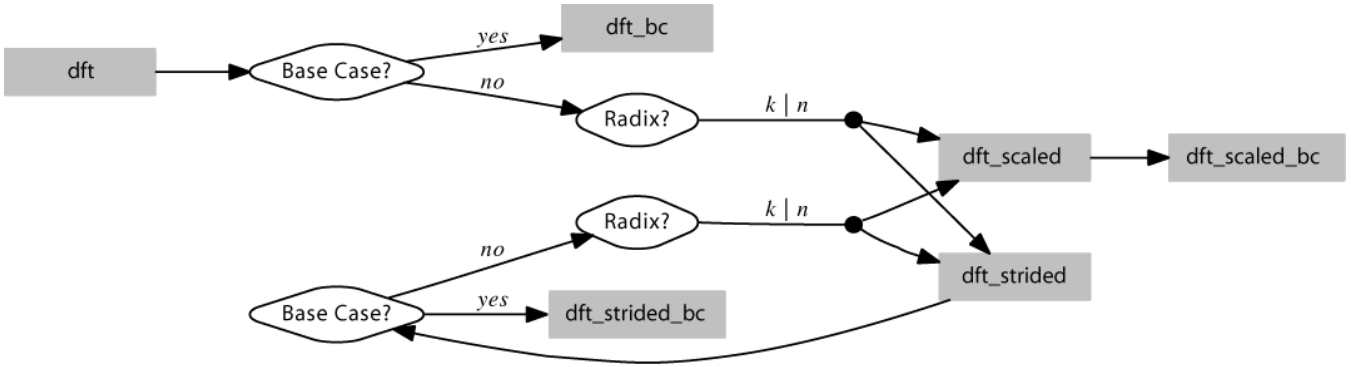
$$\downarrow$$

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

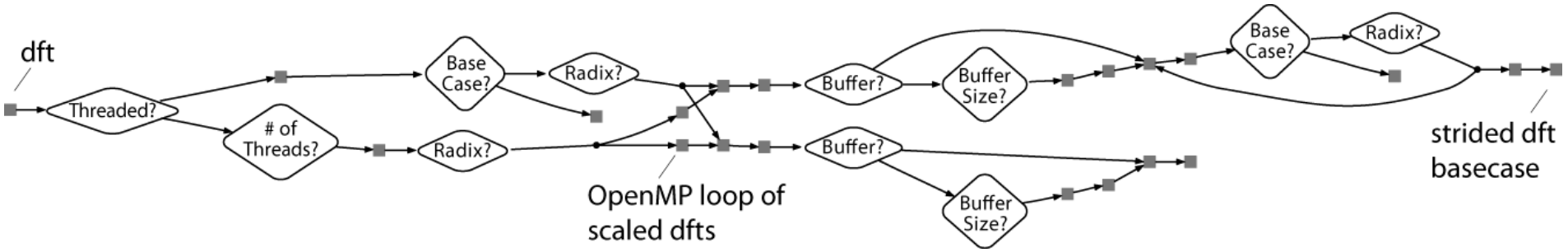
Repeat until closure

Recursion Step Closure: Examples

DFT: scalar code



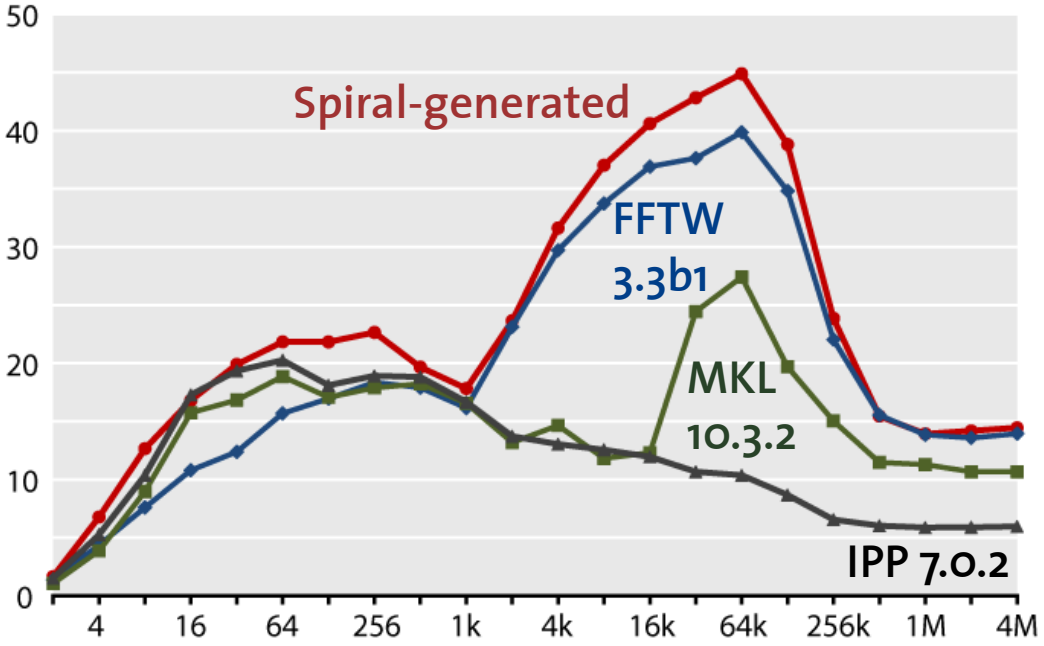
DFT: full-fledged (vectorized and parallel code)



It Really Works

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

Performance [Gflop/s]

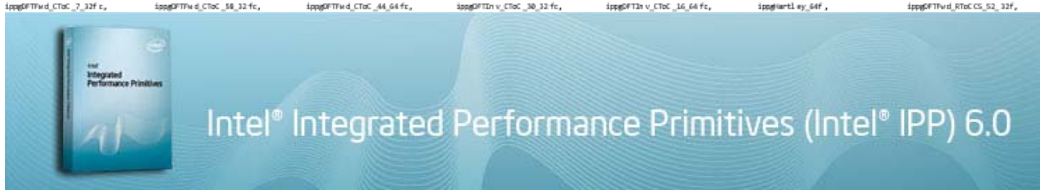


$$\mathbf{DFT}_n \to (\mathbf{DFT}_k \otimes I_m) T^n_m (I_k \otimes \mathbf{DFT}_m) L^n_k$$
$$\mathbf{DFT}_n \to P^\top_{k/2,2m} \left(\mathbf{DFT}_{2m} \oplus \left(I_{k/2-1} \otimes_i C_{2m}\, \mathbf{rDFT}_{2m}(i/k)\right)\right) (\mathbf{RDFT}_k \otimes I_m)$$
$$\mathbf{RDFT}_n \to (P^\top_{k/2,m}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

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.

Online tuning

Installation

```
configure/make
```

Use

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

Search for fastest computation strategy

Offline tuning

Installation

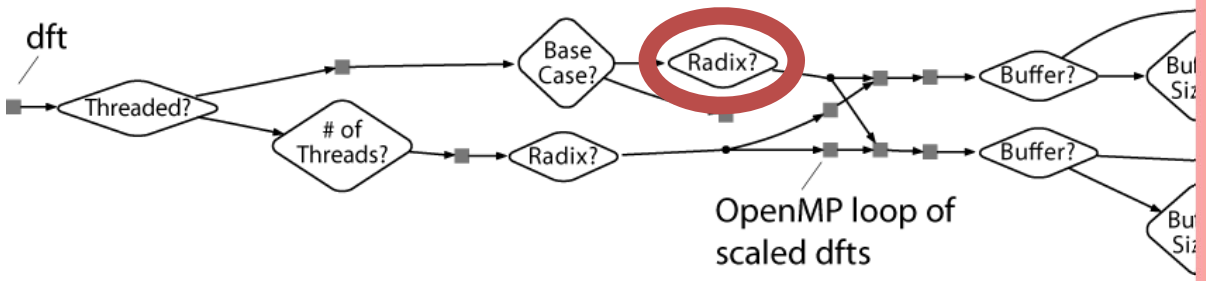
```
configure/make
```

for a few n: search
learn decision trees

Use

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

Machine learning



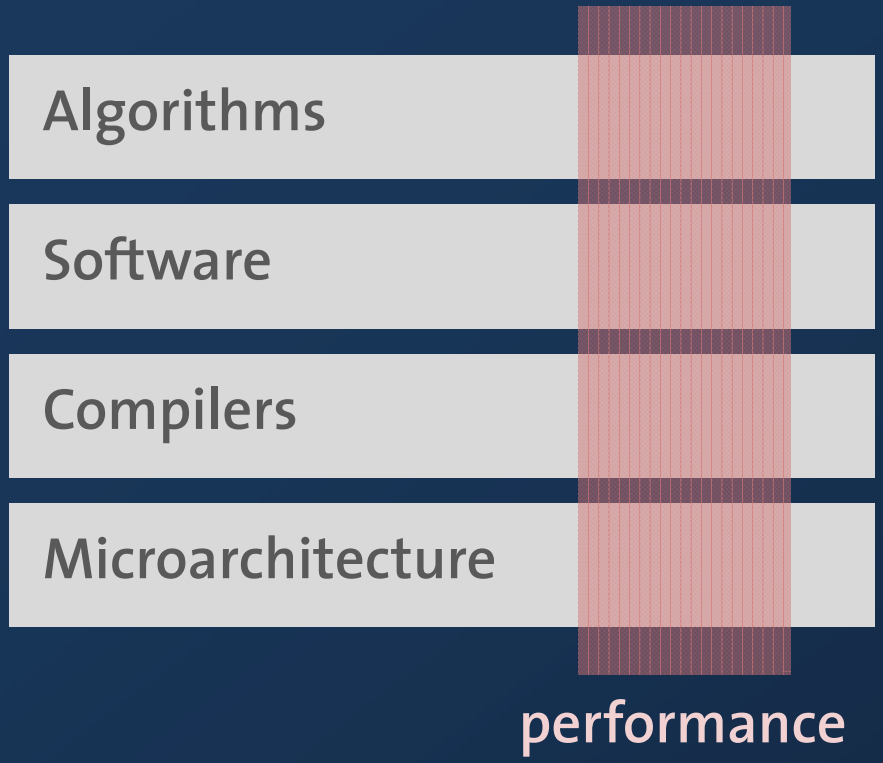

```
if ( n <= 65536 ) {
   if ( n <= 32 ) {
      if ( n <= 4 ) {return 2;}
      else {return 4;}
   }
   else {
      if ( n <= 1024 ) {
         if ( n <= 256 ) {return 8;}
         else {return 32;}
      }
      else {
```

Program Generators: Related Work

- FFTW codelet generator (Frigo)
- Flame (van de Geijn, Quintana-Orti, Bientinesi, …)
- Tensor contraction engine (Baumgartner, Sadayappan, Ramanujan, …)
- cvxgen (Mattingley, Boyd)
- PetaBricks (Ansel, Amarasinghe, …)
- Spiral

Organization

- Software performance issues
- Automatic performance tuning (autotuning)
- Automatic performance programming
- *Conclusions*



- End of free speedup
- Straightforward code often 10-100x suboptimal
- Performance ≠ efficient parallelization
- Likely inherent compiler limitations
- Performance optimization violates good programming practices

So What Do We Do?

- Teaching
- Better autotuning

Search over parameterized alternatives

Automating performance optimization with tools that complement/aid the compiler or programmer.

- Example: Program generation for performance
  Maybe Spiral can be generalized
  - *DSLs*
  - *Rewriting for platform paradigms*
  - *Search/learning*