

Automatic Performance Tuning and Machine Learning

Markus Püschel
Computer Science, ETH Zürich

with:

Frédéric de Mesmay
PhD, Electrical and Computer Engineering, Carnegie Mellon

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

SPIRAL 
www.spiral.net



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

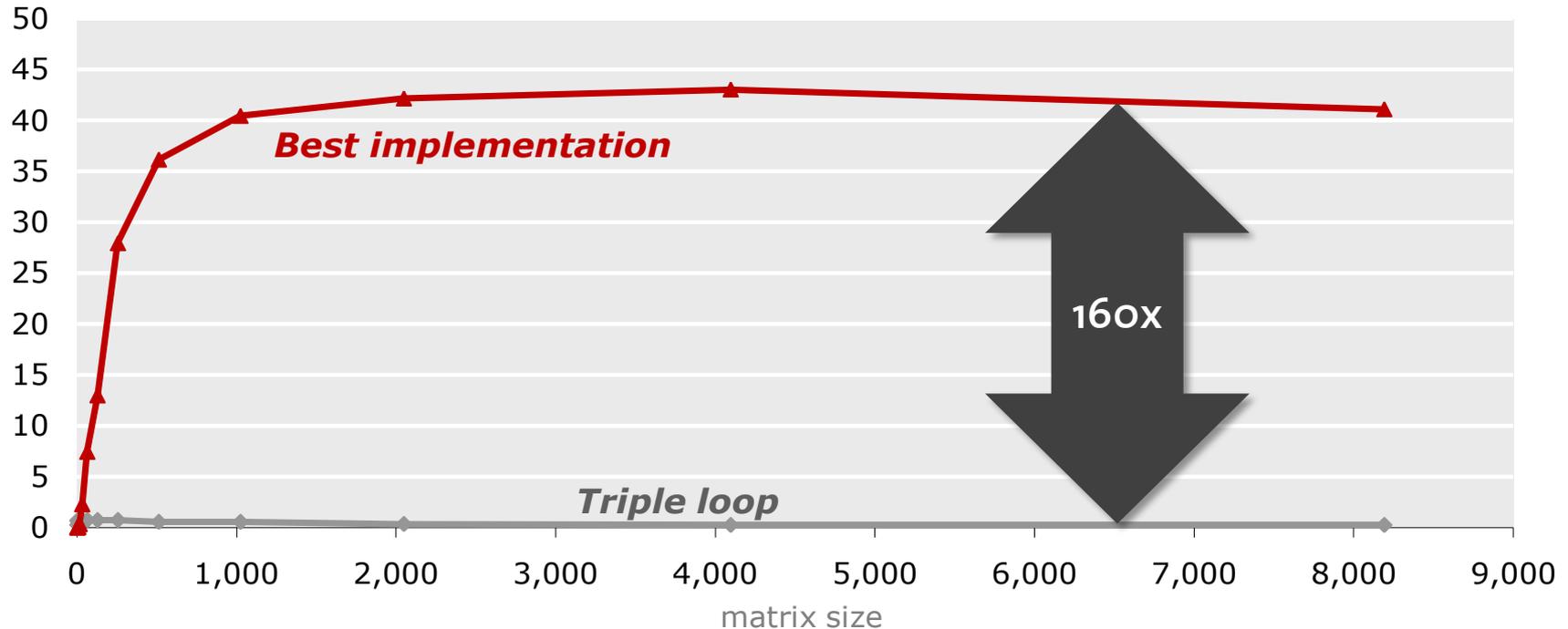
PhD and Postdoc openings:

- High performance computing
- Compilers
- Theory
- Programming languages/Generative programming

Why Autotuning?

Matrix-Matrix Multiplication (MMM) on quadcore Intel platform

Performance [Gflop/s]

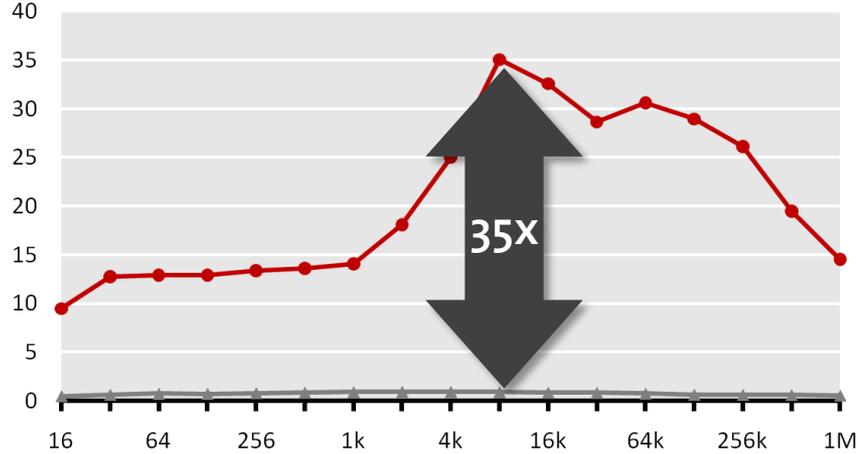


- Same (mathematical) operation count ($2n^3$)
- Compiler underperforms by 160x

Same for All Critical Compute Functions

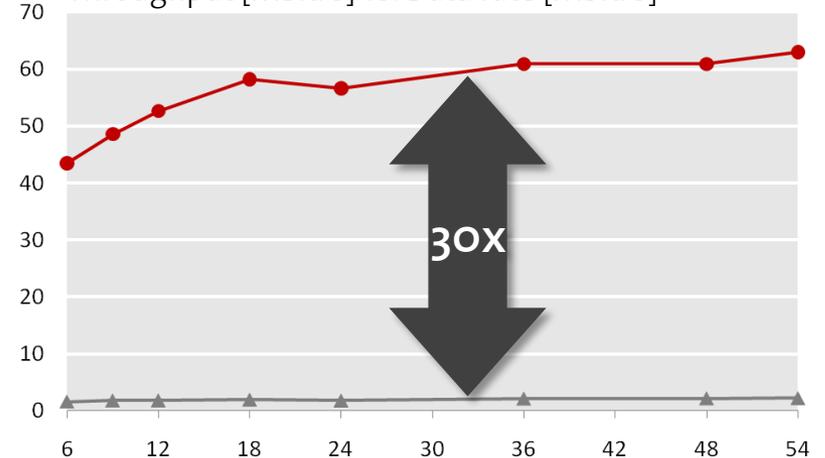
DFT (single precision) on Intel Core i7 (4 cores, 2.66 GHz)

Performance [Gflop/s]



WiFi Receiver (Physical layer) on one Intel Core

Throughput [Mbit/s] vs. Data rate [Mbit/s]



Solution: Autotuning

Definition: *Search over alternative implementations or parameters to find the fastest.*

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

However: *Search is an important tool. But expensive.*

Solution: Machine learning

Organization

- Autotuning examples
- An example use of machine learning

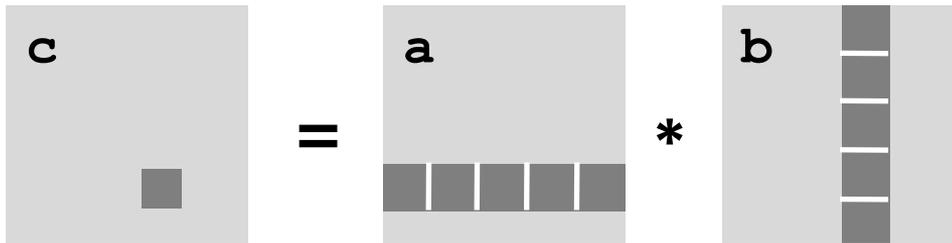
time of
implementation

time of
installation
platform known

time of
use
*problem parameters
known*

PhiPac/ATLAS: MMM Generator

Whaley, Bilmes, Demmel, Dongarra, ...

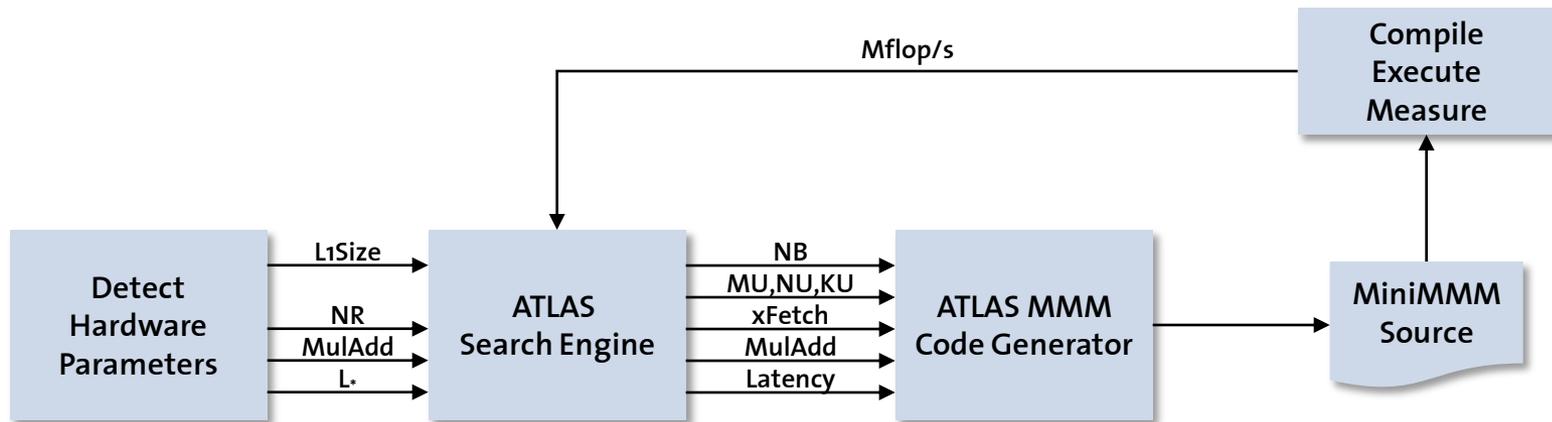


Blocking improves locality

```
c = (double *) calloc(sizeof(double), n*n);

/* Multiply n x n matrices a and b */
void mmm(double *a, double *b, double *c, int n) {
    int i, j, k;
    for (i = 0; i < n; i+=B)
        for (j = 0; j < n; j+=B)
            for (k = 0; k < n; k+=B)
                /* B x B mini matrix multiplications */
                for (i1 = i; i1 < i+B; i++)
                    for (j1 = j; j1 < j+B; j++)
                        for (k1 = k; k1 < k+B; k++)
                            c[i1*n+j1] += a[i1*n + k1]*b[k1*n + j1];
}
```

PhiPac/ATLAS: MMM Generator



time of
implementation

ATLAS
MMM generator

time of
installation
platform known

time of
use
*problem parameters
known*

FFTW: Discrete Fourier Transform (DFT)

Frigo, Johnson

Installation

configure/make

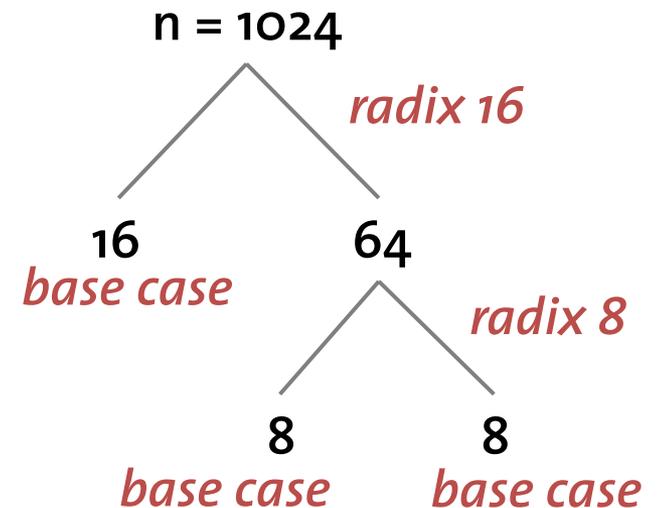
Usage

$d = \text{dft}(n)$

$d(x, y)$

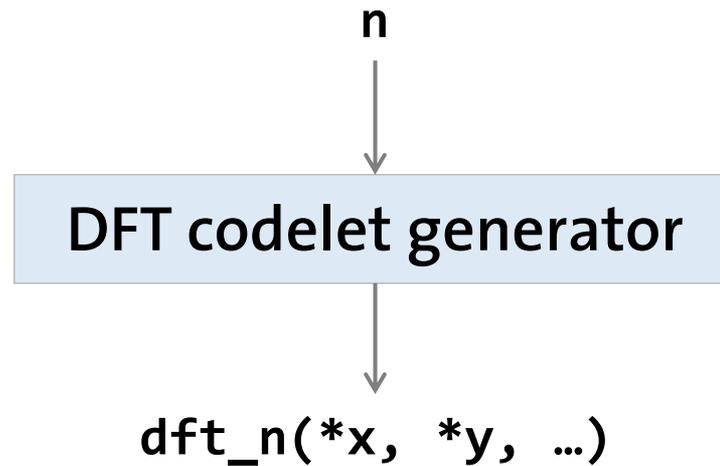
Twiddles

Search for fastest
computation strategy



FFTW: Codelet Generator

Frigo



*fixed size DFT function
straightline code*

FFTW codelet
generator



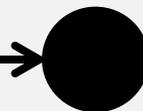
time of
implementation

ATLAS
MMM generator



time of
installation
platform known

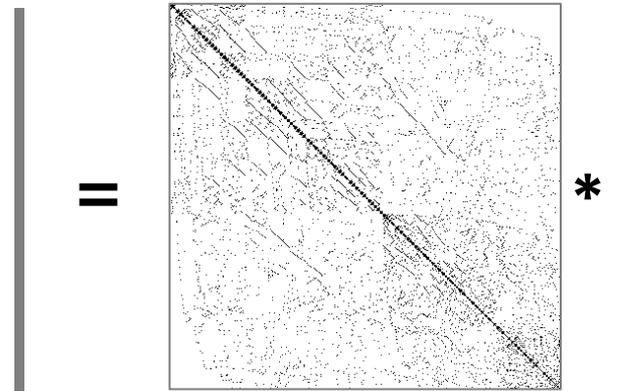
FFTW adaptive
library



time of
use
*problem parameters
known*

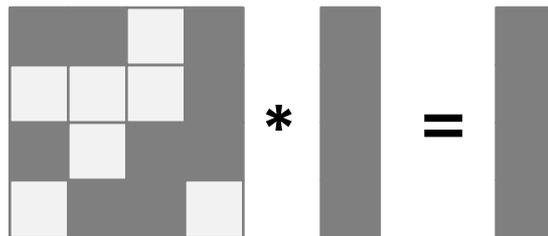
OSKI: Sparse Matrix-Vector Multiplication

Vuduc, Im, Yelick, Demmel



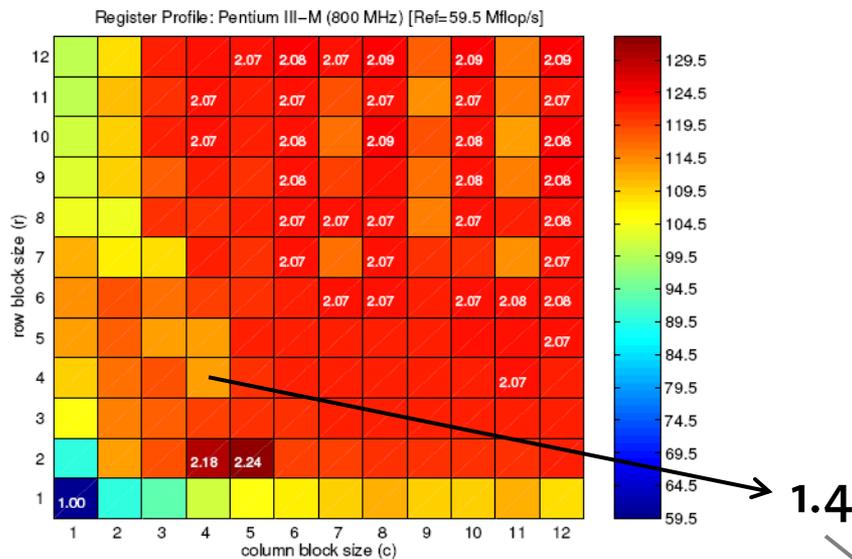
■ Blocking for registers:

- Improves locality (reuse of input vector)
- But creates overhead (zeros in block)

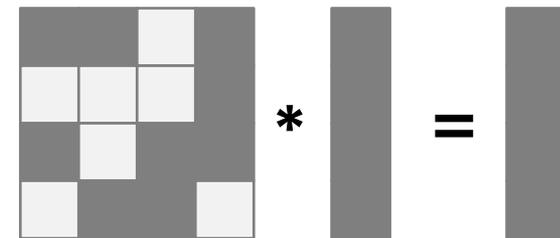


OSKI: Sparse Matrix-Vector Multiplication

Gain by blocking (dense MVM)

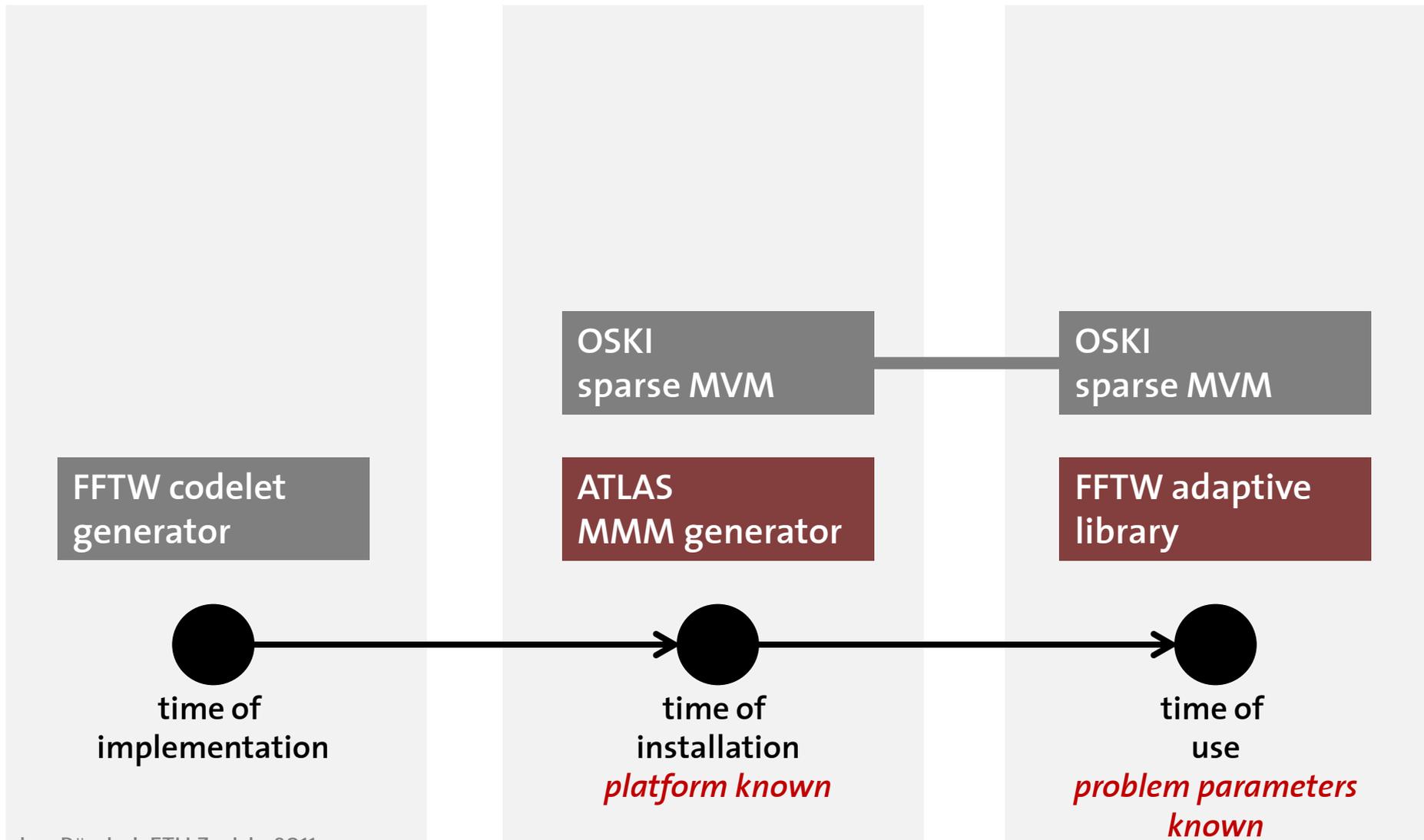


Overhead by blocking



$$16/9 = 1.77$$

$$1.4/1.77 = 0.79 \text{ (no gain)}$$



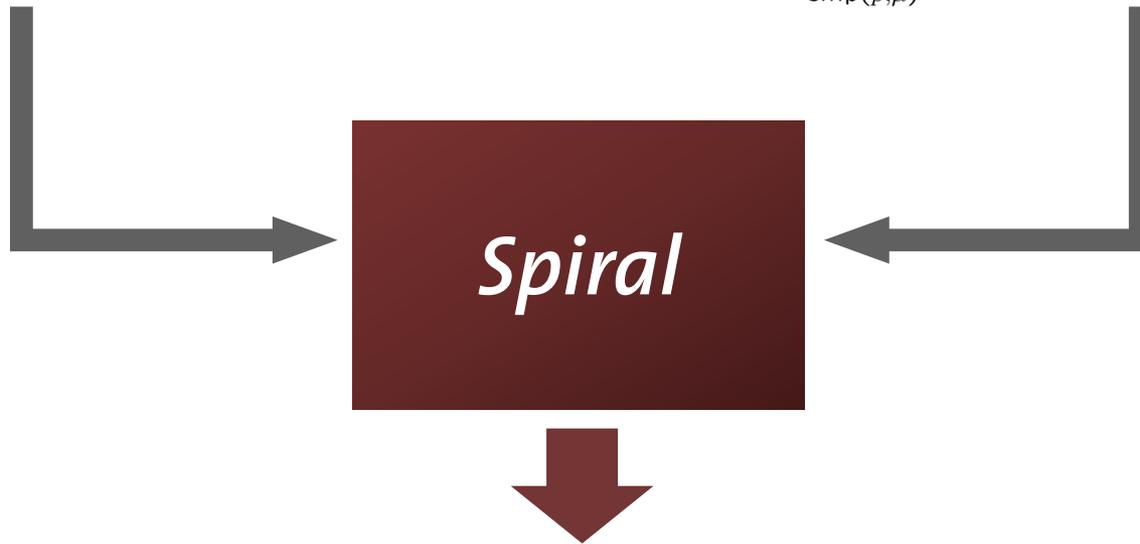
Spiral: Linear Transforms & More

Algorithm knowledge

$$\begin{aligned} \text{DFT}_n &\rightarrow P_{k/2,2m}^\top \left(\text{DFT}_{2m} \oplus \left(I_{k/2-1} \otimes_i C_{2m} \text{rDFT}_{2m}(i/k) \right) \right) \left(\text{RDFT}'_k \otimes I_m \right) \\ \begin{vmatrix} \text{rDFT}_{2n}(u) \\ \text{rDHT}_{2n}(u) \end{vmatrix} &\rightarrow L_m^{2n} \left(I_k \otimes_i \begin{vmatrix} \text{rDFT}_{2m}((i+u)/k) \\ \text{rDHT}_{2m}((i+u)/k) \end{vmatrix} \right) \left(\begin{vmatrix} \text{rDFT}_{2k}(u) \\ \text{rDHT}_{2k}(u) \end{vmatrix} \otimes I_m \right) \\ \text{RDFT-3}_n &\rightarrow (Q_{k/2,m}^\top \otimes I_2) (I_k \otimes_i \text{rDFT}_{2m}(i+1/2/k)) (\text{RDFT-3}_k \otimes I_m) \end{aligned}$$

Platform description

$$\begin{aligned} \underbrace{A_m \otimes I_n}_{\text{smp}(p,\mu)} &\rightarrow \underbrace{\left(L_m^{mp} \otimes I_{n/p} \right) \left(I_p \otimes (A_m \otimes I_{n/p}) \right) \left(L_p^{mp} \otimes I_{n/p} \right)}_{\text{smp}(p,\mu)} \\ \underbrace{I_m \otimes A_n}_{\text{smp}(p,\mu)} &\rightarrow I_p \otimes_{||} \left(I_{m/p} \otimes A_n \right) \\ \underbrace{(P \otimes I_n)}_{\text{smp}(p,\mu)} &\rightarrow (P \otimes I_{n/\mu}) \bar{\otimes} I_\mu \end{aligned}$$



Optimized implementation
regenerated for every new platform

Program Generation in Spiral (Sketched)

Transform
user specified

DFT₈

 *Algorithm rules*

$$(DFT_2 \otimes I_4) T_4^8 (I_2 \otimes ((DFT_2 \otimes I_2) \cdot T_2^4 (I_2 \otimes DFT_2) L_2^4)) L_2^8$$

Fast algorithm
in SPL
many choices



$$\sum (S_j DFT_2 G_j) \sum \left(\sum (S_{k,l} \text{diag}(t_{k,l}) DFT_2 G_l) \sum (S_m \text{diag}(t_m) DFT_2 G_{k,m}) \right)$$

Σ-SPL



Optimized implementation

*Optimization at all
abstraction levels*



parallelization
vectorization



loop
optimizations



constant folding
scheduling
.....

+ search

Machine learning

Machine learning

Spiral: transforms
general input size

Spiral: transforms
general input size

Spiral: transforms
fixed input size

OSKI
sparse MVM

OSKI
sparse MVM

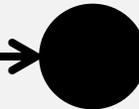
FFTW codelet
generator

ATLAS
MMM generator

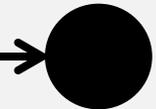
FFTW adaptive
library



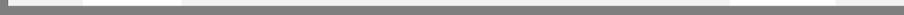
time of
implementation



time of
installation
platform known



time of
use
*problem parameters
known*



Organization

- Autotuning examples
- An example use of machine learning

Online tuning (time of use)

Installation

configure/make

Use

$d = \text{dft}(n)$
 $d(x, y)$

Twiddles

Search for fastest
computation strategy

Offline tuning (time of installation)

Installation

configure/make

for a few n : search
learn decision trees

Use

$d = \text{dft}(n)$
 $d(x, y)$

Twiddles



Goal

Integration with Spiral-Generated Libraries

Voronenko 2008

$(\text{DFT}_k \otimes \text{I}_m) \Upsilon_m^n (\text{I}_k \otimes \text{DFT}_m) \text{L}_k^n$
+ some platform information



Online tunable library

$$\begin{aligned}
 \text{DFT}_n &\rightarrow P_{k/2,2m}^\top (\text{DFT}_{2m} \oplus (I_{k/2-1} \otimes_i C_{2m} \text{rDFT}_{2m}(i/k))) (\text{RDFT}'_k \otimes \text{I}_m), \quad k \text{ even,} \\
 \begin{pmatrix} \text{RDFT}'_n \\ \text{RDFT}'_n \\ \text{DHT}'_n \\ \text{DHT}'_n \end{pmatrix} &\rightarrow (P_{k/2,m}^\top \otimes \text{I}_2) \left(\begin{pmatrix} \text{RDFT}'_{2m} \\ \text{RDFT}'_{2m} \\ \text{DHT}'_{2m} \\ \text{DHT}'_{2m} \end{pmatrix} \oplus \left(I_{k/2-1} \otimes_i D_{2m} \begin{pmatrix} \text{rDFT}'_{2m}(i/k) \\ \text{rDFT}'_{2m}(i/k) \\ \text{rDHT}'_{2m}(i/k) \\ \text{rDHT}'_{2m}(i/k) \end{pmatrix} \right) \right) \left(\begin{pmatrix} \text{RDFT}'_k \\ \text{RDFT}'_k \\ \text{DHT}'_k \\ \text{DHT}'_k \end{pmatrix} \otimes \text{I}_m \right), \quad k \text{ even,} \\
 \begin{pmatrix} \text{rDFT}'_{2n}(u) \\ \text{rDHT}'_{2n}(u) \end{pmatrix} &\rightarrow L_m^{2n} \left(I_k \otimes_i \begin{pmatrix} \text{rDFT}'_{2m}((i+u)/k) \\ \text{rDHT}'_{2m}((i+u)/k) \end{pmatrix} \right) \left(\begin{pmatrix} \text{rDFT}'_{2k}(u) \\ \text{rDHT}'_{2k}(u) \end{pmatrix} \otimes \text{I}_m \right), \\
 \text{RDFT-3}_n &\rightarrow (Q_{k/2,m}^\top \otimes \text{I}_2) (I_k \otimes_i \text{rDFT}_{2m}(i+1/2/k)) (\text{RDFT-3}_k \otimes \text{I}_m), \quad k \text{ even,} \\
 \text{DCT-2}_n &\rightarrow P_{k/2,2m}^\top (\text{DCT-2}_{2m} K_2^{2m} \oplus (I_{k/2-1} \otimes N_{2m} \text{RDFT-3}_{2m}^\top)) B_n(L_{k/2}^{n/2} \otimes \text{I}_2) (\text{I}_m \otimes \text{RDFT}'_k) Q_{m/2,k}, \\
 \text{DCT-3}_n &\rightarrow \text{DCT-2}_n^\top, \\
 \text{DCT-4}_n &\rightarrow Q_{k/2,2m}^\top (I_{k/2} \otimes N_{2m} \text{RDFT-3}_{2m}^\top) B_n(L_{k/2}^{n/2} \otimes \text{I}_2) (\text{I}_m \otimes \text{RDFT-3}_k) Q_{m/2,k}. \\
 \text{DFT}_n &\rightarrow (\text{DFT}_k \otimes \text{I}_m) \Upsilon_m^n (\text{I}_k \otimes \text{DFT}_m) \text{L}_k^n, \quad n = km \\
 \text{DFT}_n &\rightarrow P_n (\text{DFT}_k \otimes \text{DFT}_m) Q_n, \quad n = km, \text{ gcd}(k, m) = 1 \\
 \text{DFT}_p &\rightarrow R_p^\top (\text{I}_1 \oplus \text{DFT}_{p-1}) D_p (\text{I}_1 \oplus \text{DFT}_{p-1}) R_p, \quad p \text{ prime} \\
 \text{DCT-3}_n &\rightarrow (\text{I}_m \oplus \text{J}_m) \text{L}_m^n (\text{DCT-3}_m(1/4) \oplus \text{DCT-3}_m(3/4)) \\
 &\quad \cdot (\text{F}_2 \otimes \text{I}_m) \begin{bmatrix} \text{I}_m & 0 \oplus -\text{J}_{m-1} \\ \frac{1}{\sqrt{2}} (\text{I}_1 \oplus 2\text{I}_m) \end{bmatrix}, \quad n = 2m \\
 \text{DCT-4}_n &\rightarrow S_n \text{DCT-2}_n \text{diag}_{0 \leq k < n} (1/(2 \cos((2k+1)\pi/4n))) \\
 \text{IMDCT}_{2m} &\rightarrow (\text{J}_m \oplus \text{I}_m \oplus \text{I}_m \oplus \text{J}_m) \left(\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} \otimes \text{I}_m \right) \oplus \left(\begin{bmatrix} -1 \\ -1 \end{bmatrix} \otimes \text{I}_m \right) \right) \text{J}_{2m} \text{DCT-4}_{2m} \\
 \text{WHT}_{2k} &\rightarrow \prod_{i=1}^t (\text{I}_{2^{k_1+\dots+k_{i-1}}} \otimes \text{WHT}_{2^{k_i}} \otimes \text{I}_{2^{k_{i+1}+\dots+k_t}}), \quad k = k_1 + \dots + k_t \\
 \text{DFT}_2 &\rightarrow \text{F}_2 \\
 \text{DCT-2}_2 &\rightarrow \text{diag}(1, 1/\sqrt{2}) \text{F}_2 \\
 \text{DCT-4}_2 &\rightarrow \text{J}_2 \text{R}_{13\pi/8}
 \end{aligned}$$

Organization

- Autotuning examples
- An example use of machine learning
 - Anatomy of an adaptive discrete Fourier transform library
 - Decision tree generation using C4.5
 - Results

Discrete/Fast Fourier Transform

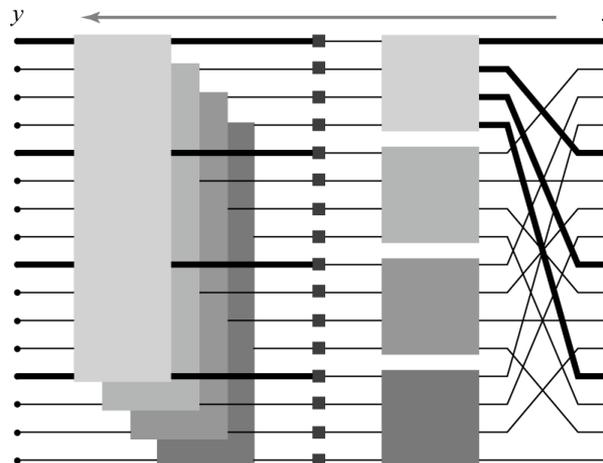
- Discrete Fourier transform (DFT):

$$y = \text{DFT}_n x, \quad \text{DFT}_n = [e^{-2k\ell\pi i/n}]_{0 \leq k, \ell < n}$$

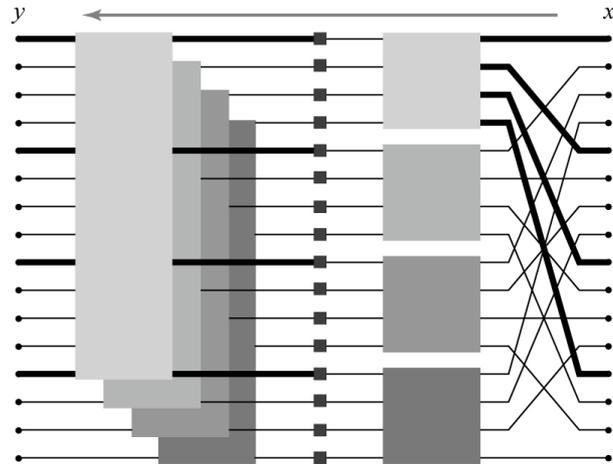
- Cooley/Tukey fast Fourier transform (FFT):

$$\text{DFT}_n = (\text{DFT}_k \otimes \mathbf{I}_m) \mathbb{T}_m^n (\mathbf{I}_k \otimes \text{DFT}_m) \mathbb{L}_k^n, \quad n = km$$

- Dataflow (right to left): $16 = 4 \times 4$



Adaptive Scalar Implementation (FFTW 2.x)



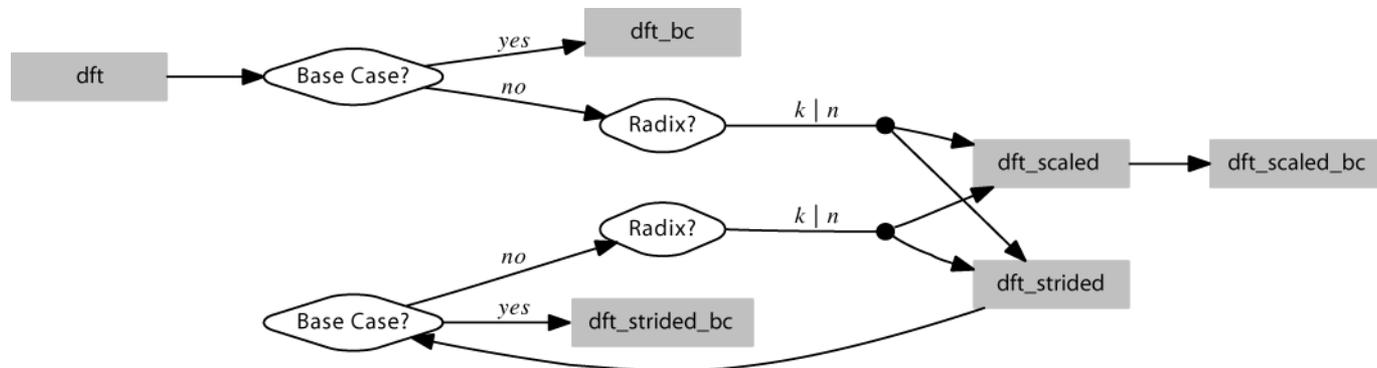
```
void dft(int n, cpx *y, cpx *x) {  
    if (use_dft_base_case(n))  
        dft_bc(n, y, x);  
    else {  
        int k = choose_dft_radix(n);  
        for (int i=0; i < k; ++i)  
            dft_strided(m, k, t + m*i, x + m*i);  
        for (int i=0; i < m; ++i)  
            dft_scaled(k, m,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

Choices used for adaptation

Decision Graph of Library

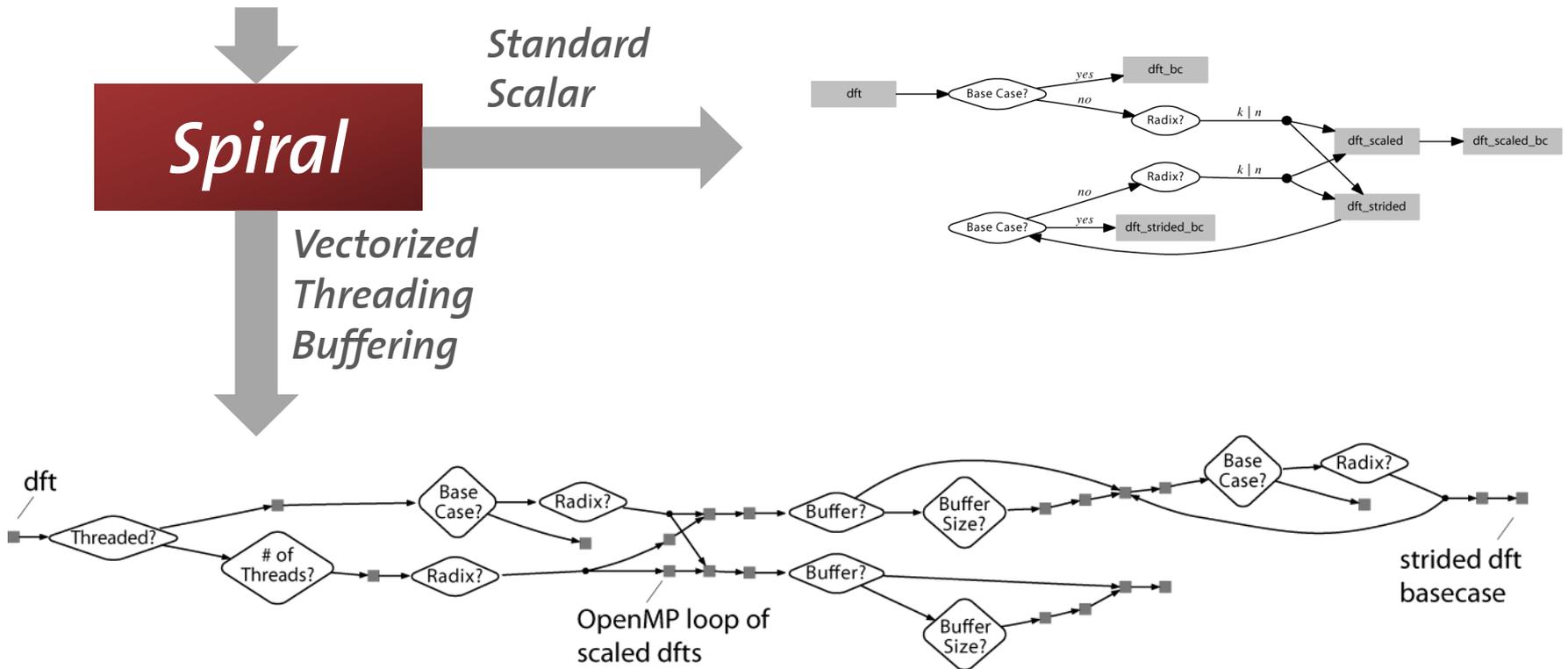
```
void dft(int n, cpx *y, cpx *x) {  
    if (use_dft_base_case(n))  
        dft_bc(n, y, x);  
    else {  
        int k = choose_dft_radix(n);  
        for (int i=0; i < k; ++i)  
            dft_strided(m, k, t + m*i, x + m*i);  
        for (int i=0; i < m; ++i)  
            dft_scaled(k, m,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

Choices used for adaptation



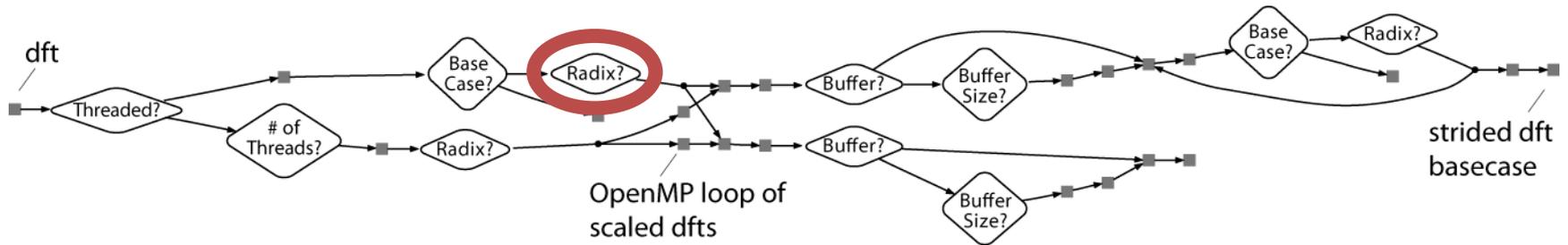
Spiral-Generated Libraries

$$(\text{DFT}_k \otimes \text{I}_m) \text{T}_m^n (\text{I}_k \otimes \text{DFT}_m) \text{L}_k^n$$



- 20 mutually recursive functions
- 10 different choices (occurring recursively)
- Choices are heterogeneous (radix, threading, buffering, ...)

Our Work



Upon installation, generate decision trees for each choice

Example:

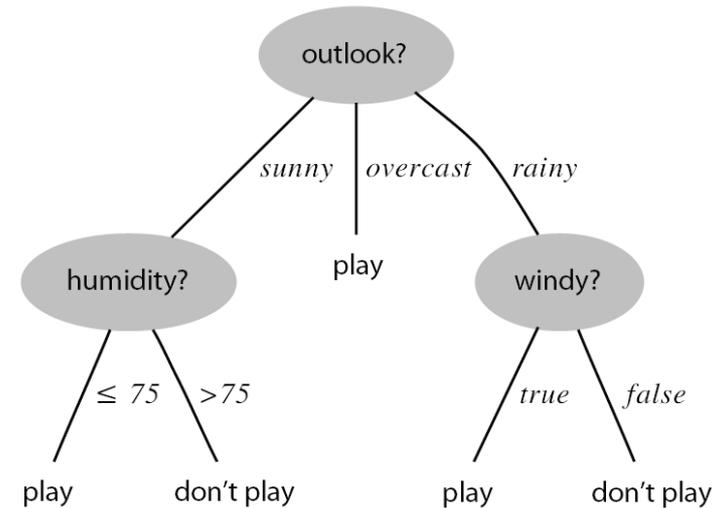
```
if ( n <= 65536 ) {  
    if ( n <= 32 ) {  
        if ( n <= 4 ) {return 2;}  
        else {return 4;}  
    }  
    else {  
        if ( n <= 1024 ) {  
            if ( n <= 256 ) {return 8;}  
            else {return 32;}  
        }  
        else {
```



Statistical Classification: C4.5

Features (events)

Outlook	Temperature	Humidity	Windy	Decision
sunny	85	85	false	don't play
sunny	80	90	true	don't play
overcast	83	78	false	play
rain	70	96	false	play
rain	68	80	false	play
rain	65	70	true	don't play
overcast	64	65	true	play
sunny	72	95	false	don't play
sunny	69	70	false	play
rain	75	80	false	play
sunny	75	70	true	play
overcast	72	90	true	play
overcast	81	75	false	play
rain	71	80	true	don't play



$$P(\text{play}|\text{windy}=\text{false}) = 6/8$$

$$P(\text{don't play}|\text{windy}=\text{false}) = 2/8$$

$$P(\text{play}|\text{windy}=\text{true}) = 1/2$$

$$P(\text{don't play}|\text{windy}=\text{true}) = 1/2$$



$$H(\text{windy}=\text{false}) = 0.81$$

$$H(\text{windy}=\text{true}) = 1.0$$



Entropy of Features

$$H(\text{windy}) = 0.89$$

$$H(\text{outlook}) = 0.69$$

$$H(\text{humidity}) = \dots$$

Application to Libraries

- Features = arguments of functions (except variable pointers)

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

```
dft_strided(int n, int istr, cpx *y, cpx *x)
```

```
dft_scaled(int n, int str, cpx *d, cpx *y, cpx *x)
```

- **At installation time:**

- Run search for a few input sizes n
- Yields training set: features and associated decisions (several for each size)
- Generate decision trees using C4.5 and insert into library

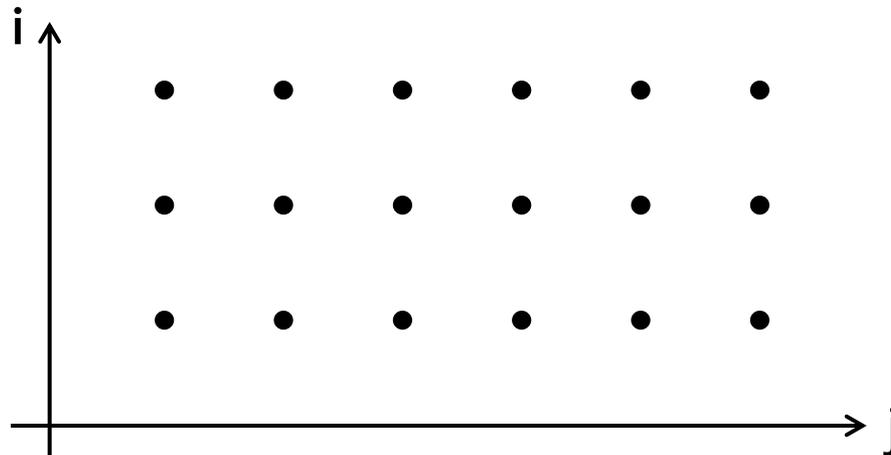
Issues

- **Correctness of generated decision trees**

- Issue: learning sizes n in $\{12, 18, 24, 48\}$, may find radix 6
- Solution: correction pass through decision tree

- **Prime factor structure**

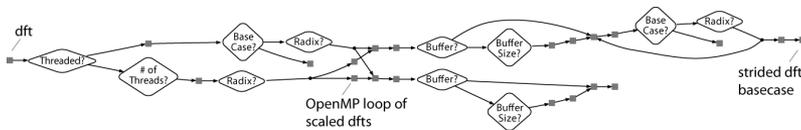
→
 $n = 2^i 3^j = 2, 3, 4, 6, 9, 12, 16, 18, 24, 27, 32, \dots$



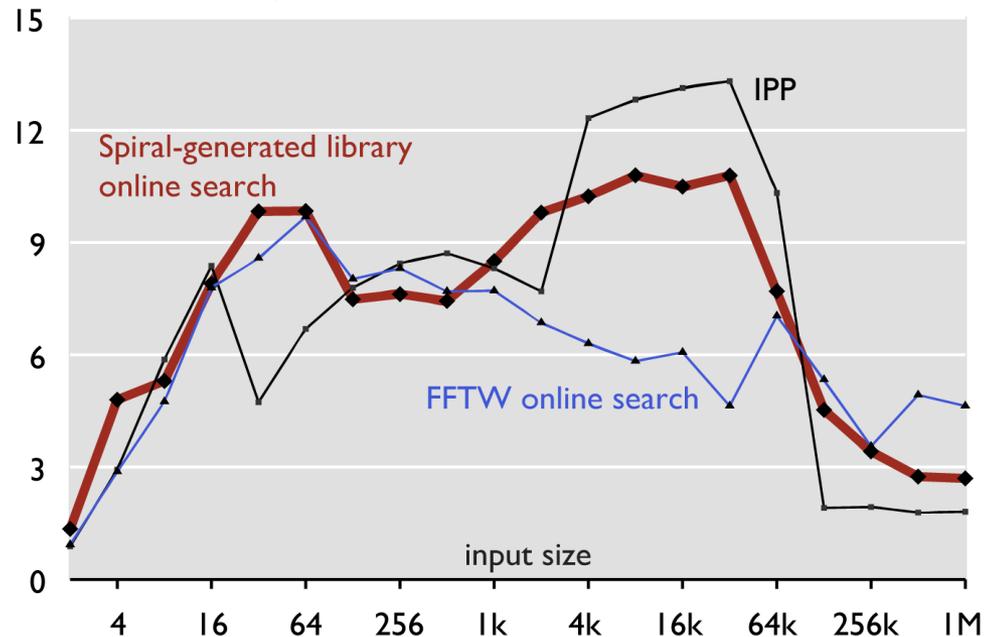
*Compute i, j
and add to features*

Experimental Setup

- 3GHz Intel Xeon 5160 (2 Core 2 Duos = 4 cores)
- Linux 64-bit, icc 10.1
- Libraries:
 - IPP 5.3
 - FFTW 3.2 alpha 2
 - Spiral-generated library

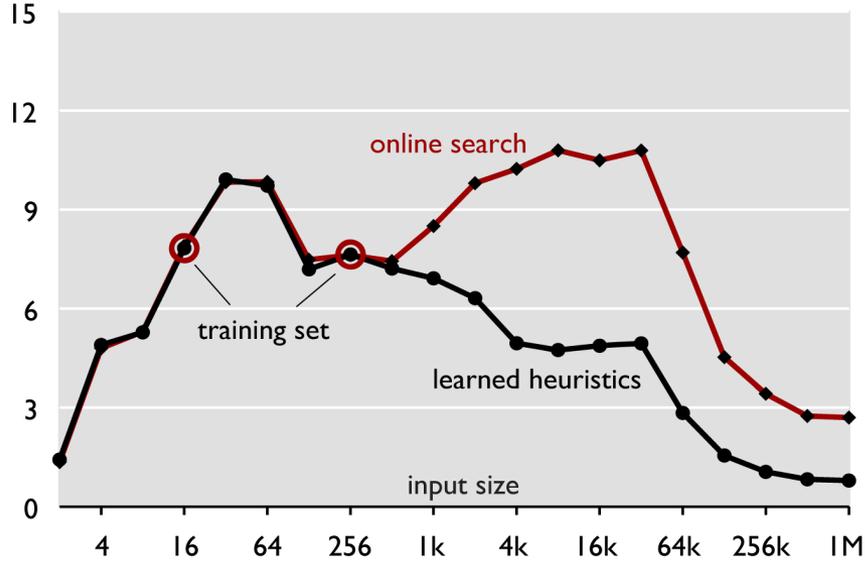


Complex DFT, double precision, up to 4 threads
Performance [GFlop/s]



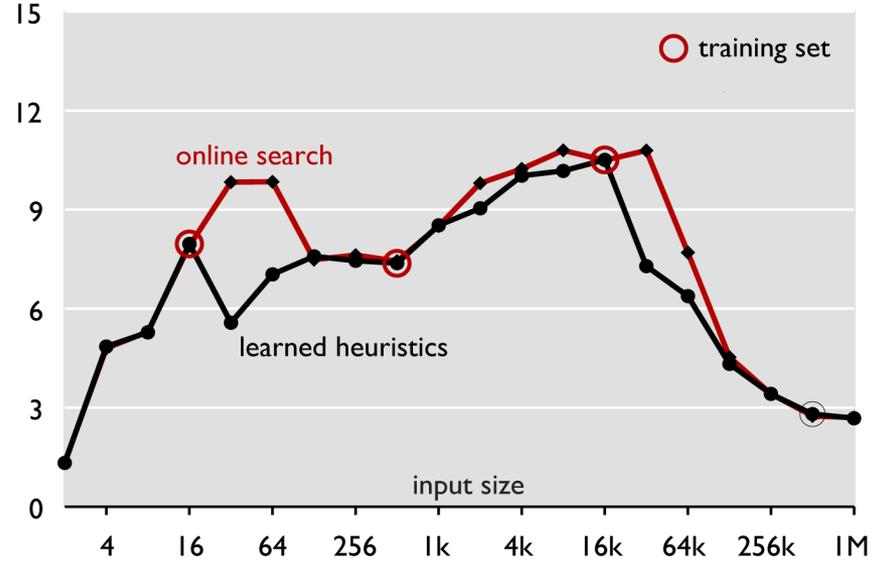
Complex DFT, double precision, up to 4 threads

Performance [GFlop/s]



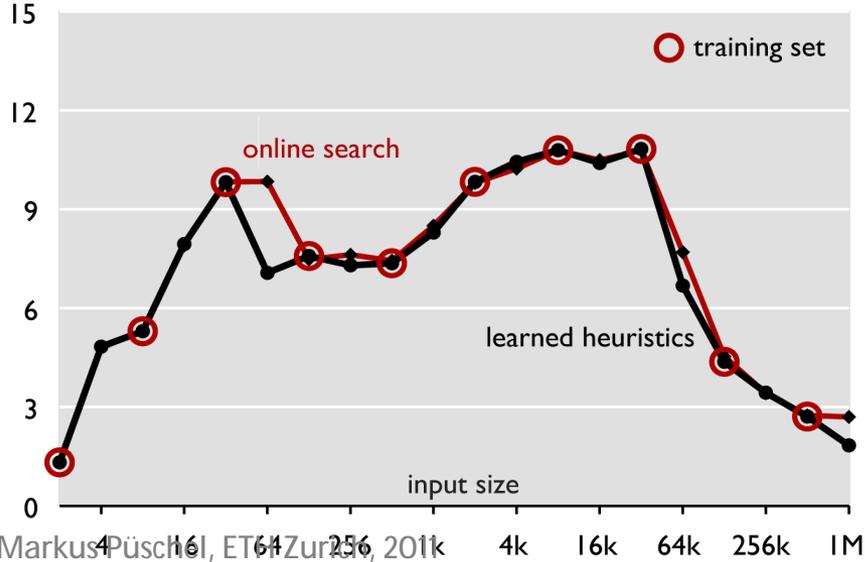
Complex DFT, double precision, up to 4 threads

Performance [GFlop/s]



Complex DFT, double precision, up to 4 threads

Performance [GFlop/s]

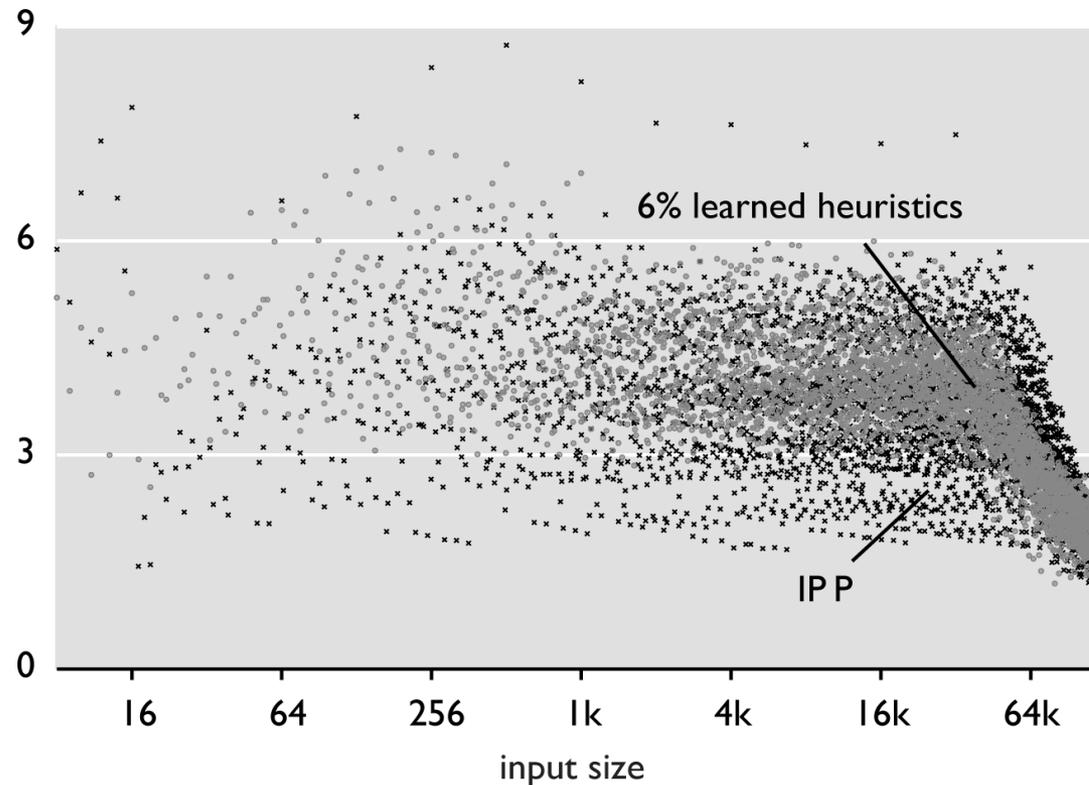


Learning works as expected

“All” Sizes

Complex DFT, double precision, mixed sizes

Performance [GFlop/s]

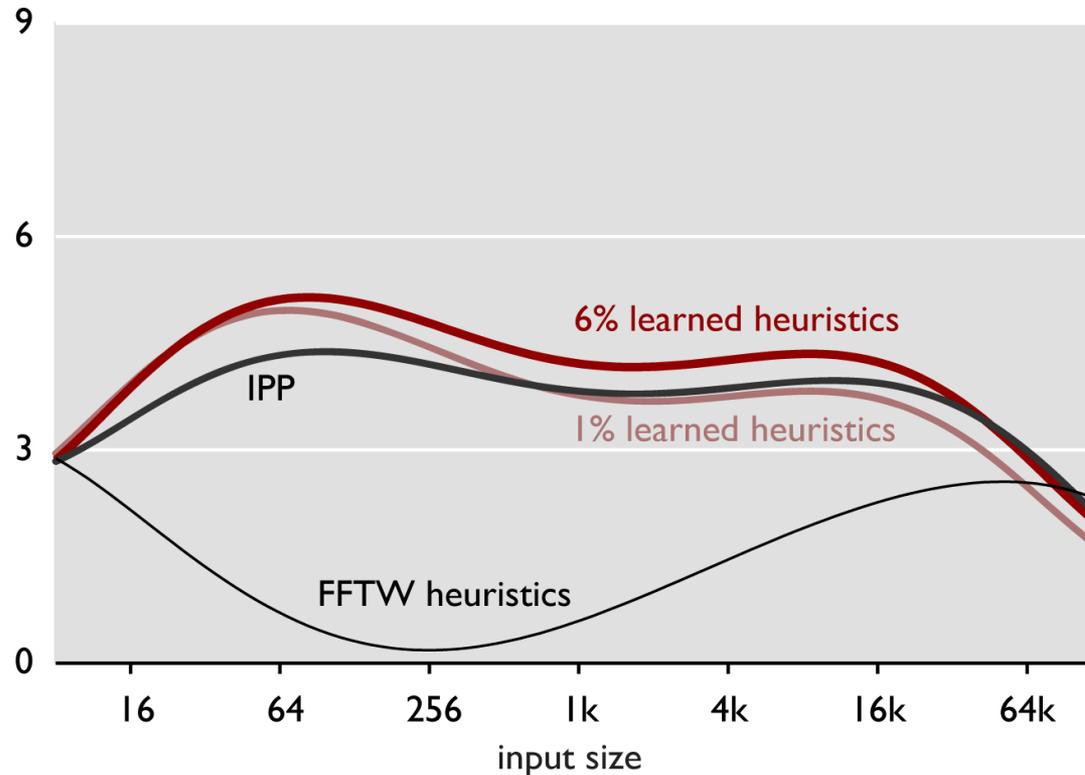


- All sizes $n \leq 2^{18}$, with prime factors ≤ 19

“All” Sizes

Complex DFT, double precision, mixed sizes

Performance [GFlop/s]



- All sizes $n \leq 2^{18}$, with prime factors ≤ 19
- Higher order fit of all sizes

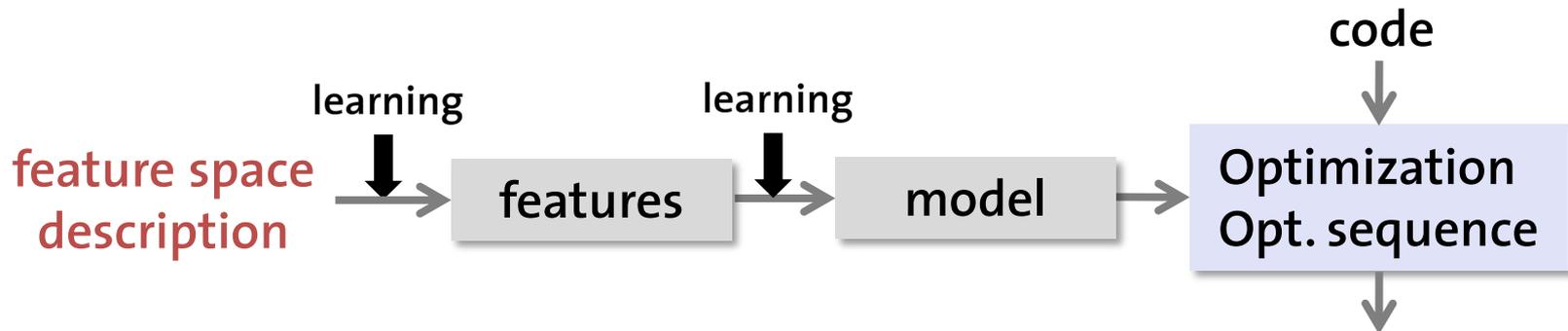
Related Work

■ Machine learning in Spiral

- Learning DFT recursions (Singer/Veloso 2001)

■ Machine learning in compilation

- Scheduling (Moss et al. 1997, Cavazos/Moss 2004)
- Branch prediction (Calder et al. 1997)
- Heuristics generation (Monsifrot/Bodin/Quiniou 2002)
- Feature generation (Leather/Bonilla/O'Boyle 2009)
- Multicores (Wang/O'Boyle 2009)



This talk

- Frédéric de Mesmay, Yevgen Voronenko and Markus Püschel
Offline Library Adaptation Using Automatically Generated Heuristics
Proc. International Parallel and Distributed Processing Symposium (IPDPS),
pp. 1-10, 2010
- Frédéric de Mesmay, Arpad Rimmel, Yevgen Voronenko and Markus Püschel
Bandit-Based Optimization on Graphs with Application to Library Performance Tuning
Proc. International Conference on Machine Learning (ICML), pp. 729-736,
2009

Message of Talk

- **Machine learning should be used in autotuning**
 - Overcomes the problem of expensive searches
 - Relatively easy to do
 - Applicable to any search-based approach

