How to Write Fast Numerical Code

Spring 2016

Lecture: Autotuning and Machine Learning

Instructor: Markus Püschel

TA: Gagandeep Singh, Daniele Spampinato, Alen Stojanov

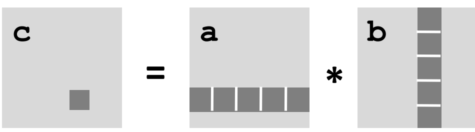
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Overview

- **Rough classification of autotuning efforts seen in course**
- **Use of machine learning**

PhiPac/ATLAS: MMM Generator

Whaley, Bilmes, Demmel, Dongarra, …

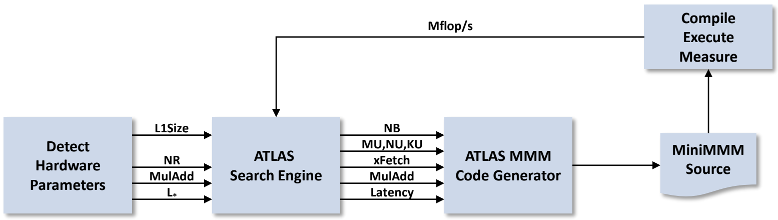


Blocking improves locality

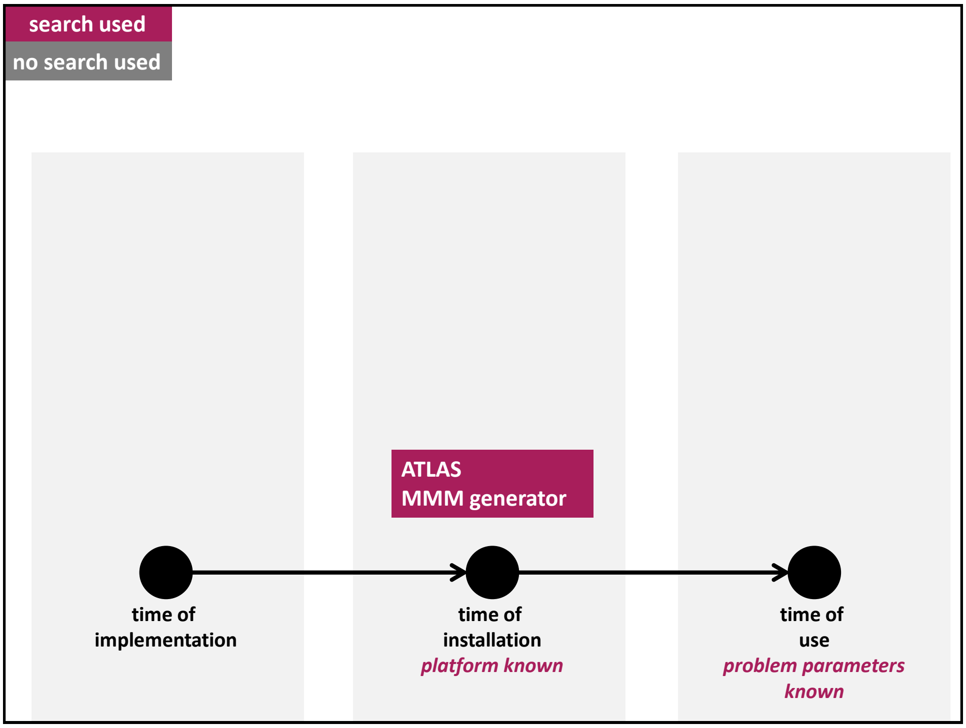
```
c = (double *) calloc(sizeof(double), n*n);

/* 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



source: Pingali, Yotov, Cornell U.



FFTW: Discrete Fourier Transform (DFT)

Frigo, Johnson

Installation

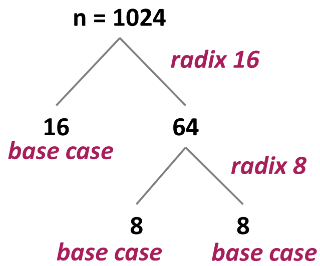
```
configure/make
```

Usage

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

Twiddles

Search for fastest computation strategy



FFTW: Codelet Generator

Frigo

n

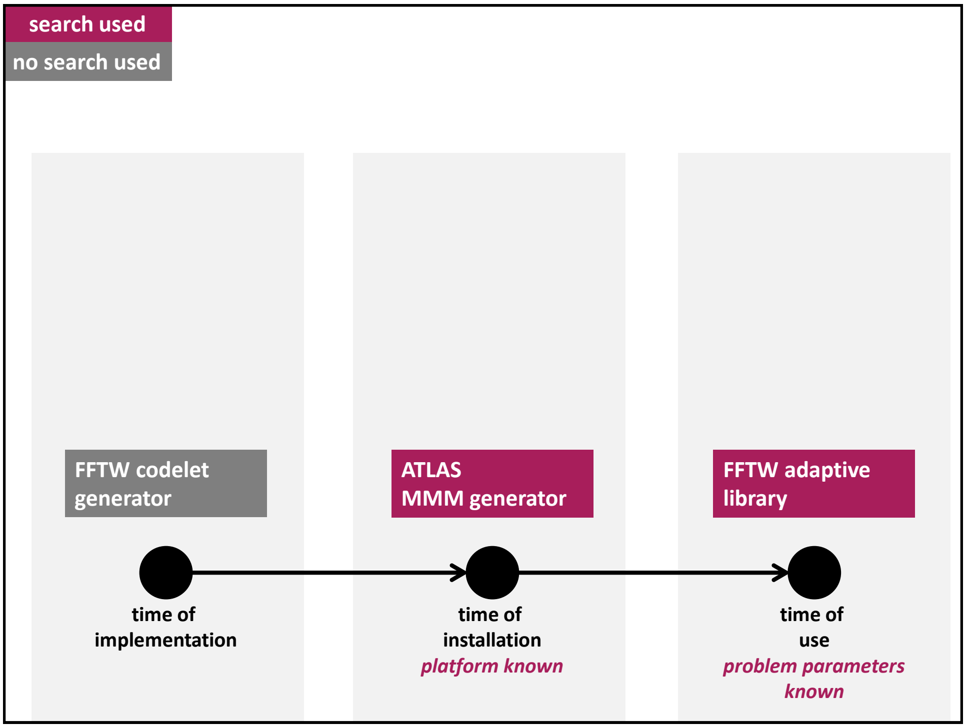
↓

DFT codelet generator

↓

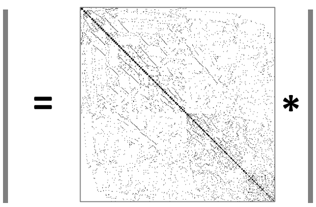
`dft_n(*x, *y, …)`

fixed size DFT functions
straightline code

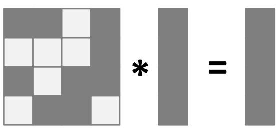


OSKI: Sparse Matrix-Vector Multiplication

Vuduc, Im, Yelick, Demmel

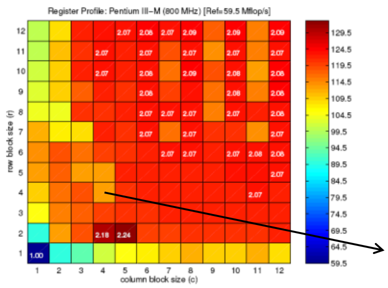


- **Blocking for registers:**
  - Improves locality (reuse of input vector)
  - But creates overhead (zeros in block)



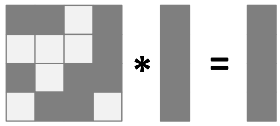
OSKI: Sparse Matrix-Vector Multiplication

Gain by blocking (dense MVM)



1.4

Overhead by blocking



16/9 = 1.77

1.4/1.77 = 0.79 (no gain)

search used

no search used

OSKI sparse MVM

OSKI sparse MVM

FFTW codelet generator

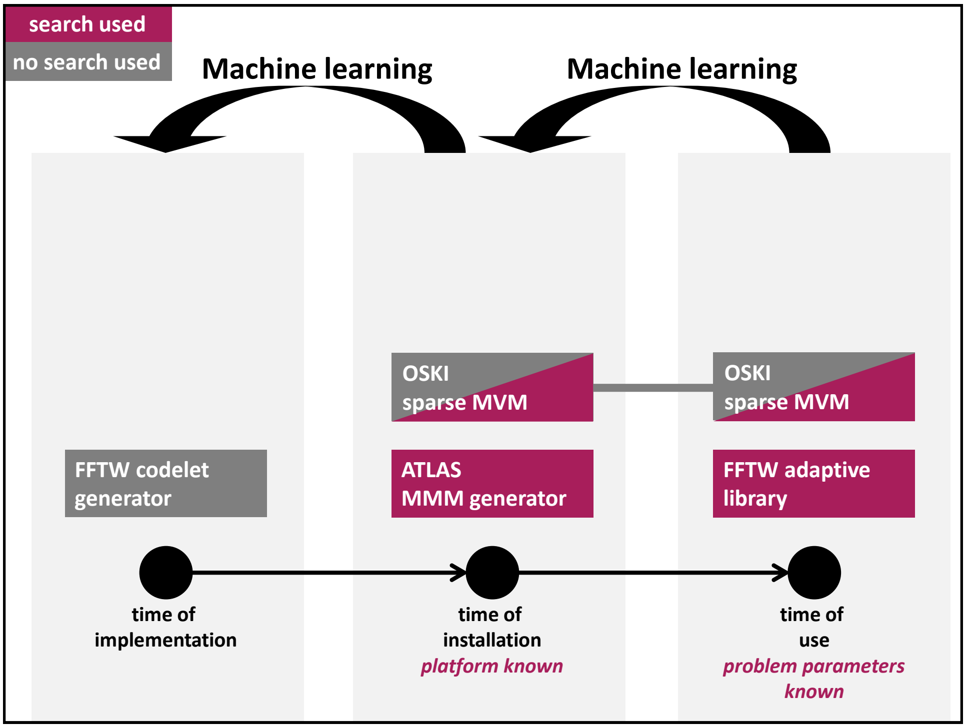
ATLAS MMM generator

FFTW adaptive library

time of implementation

time of installation
platform known

time of use
problem parameters known



Overview

- Rough classification of autotuning efforts seen in course
- Use of machine learning I [de Mesmay et al., IPDPS 2010]

Online tuning
(time of use)

Installation

```
configure/make
```

Use

```
d = 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

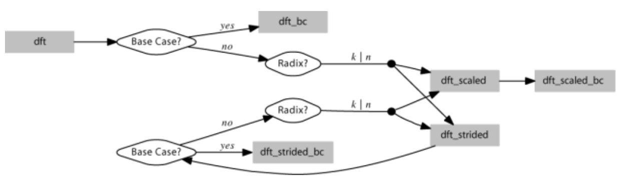
```
d = dft(n)
d(x,y)
```

Twiddles

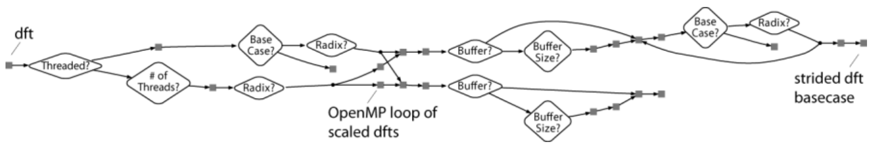
Goal

Library Structure: Examples

DFT: scalar code

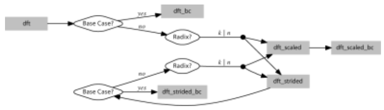


DFT: full-fledged (vectorized and parallel code)



Library Structure: Examples

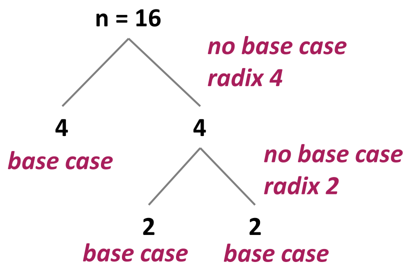
DFT: scalar code



Recursive choice:

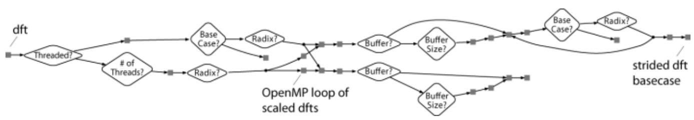
n = 2^k	*base case?* *radix?*

Example selections for n = 16:



Library Structure: Examples

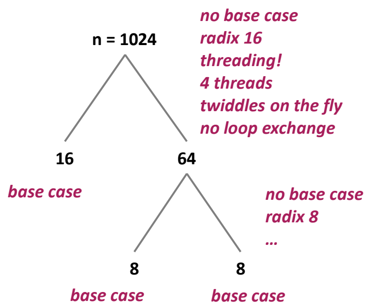
DFT: full-fledged (vectorized and parallel code)



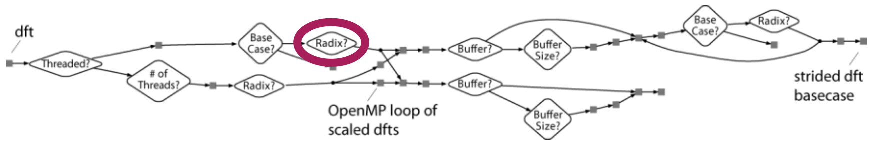
Recursive choice:

n = 2^k	*base case?* *radix?* *threading?* *#threads?* *twiddles?* *loop exchange?*

Example selections for n = 1024:



Our Work



Upon installation, generate decision trees for each choice

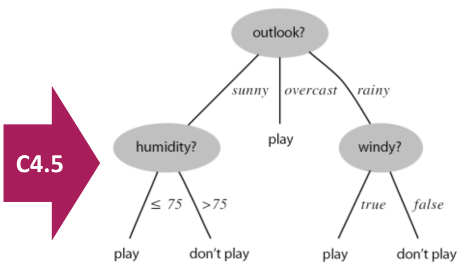
Example:

```
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(play|windy=false) = 6/8

P(don't play|windy=false) = 2/8

P(play|windy=true) = 1/2

P(don't play|windy=false) = 1/2

H(windy=false) = 0.81

H(windy=true) = 1.0

Entropy of Features

H(windy) = 0.89

H(outlook) = 0.69

H(humidity) = …

Application to Libraries

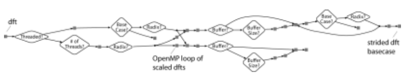
- **Features = arguments of functions (except variable pointers)**

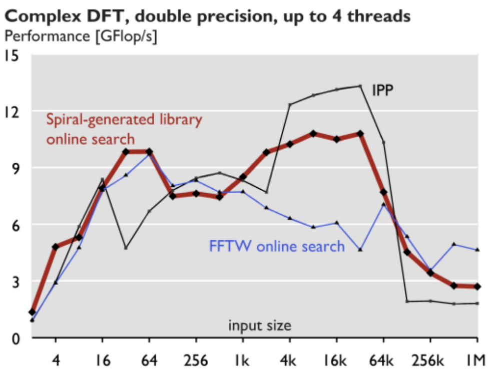
```
dft(int n, cpx *y, cpx *x)
dft_strided(int n, int istr, cpx *y, cpx *x)
dft_scaled(int n, int str, cpx *d, cpx *y, cpx *x)
```

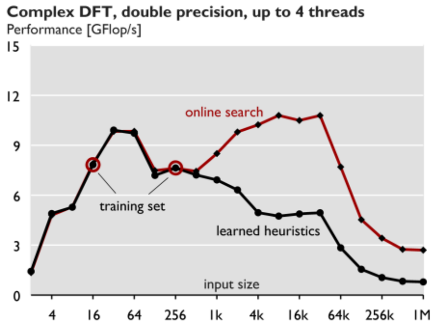
- **At installation time:**
  - Run search for a few input sizes n
  - Yields training set: features and associated decisions (several for each size)
  - Generate decision trees using C4.5 and insert into library

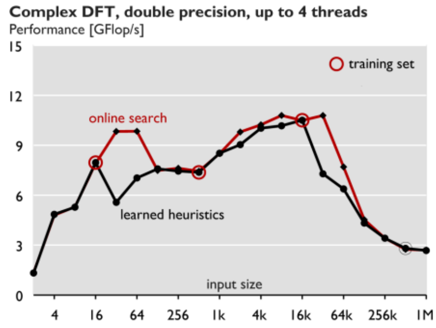
Experimental Setup

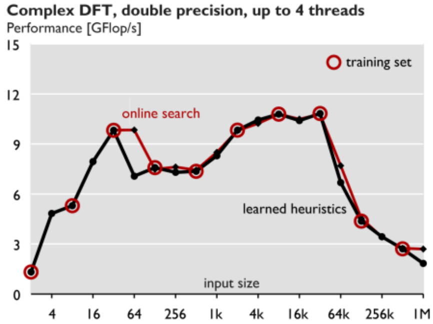
- **3GHz Intel Xeon 5160 (2 Core 2 Duos = 4 cores)**
- **Linux 64-bit, icc 10.1**
- **Libraries:**
  - IPP 5.3
  - FFTW 3.2 alpha 2
  - Spiral-generated library





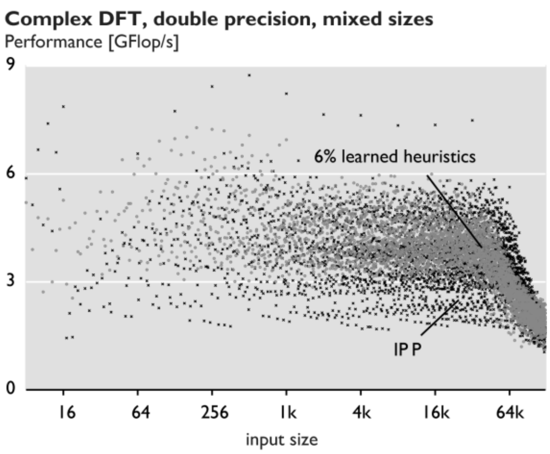






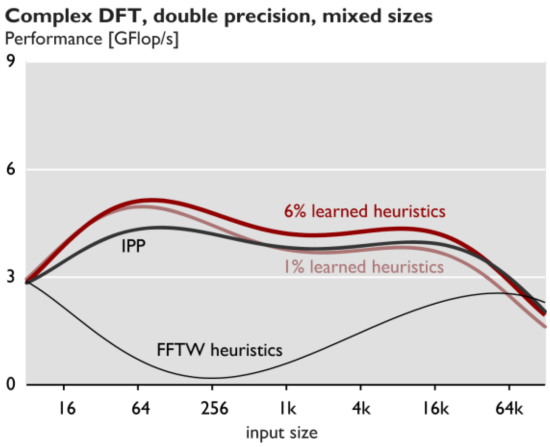
Learning works as expected

"All" Sizes



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

“All” Sizes



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

Message of Lecture

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