How to Write Fast Numerical Code

Spring 2015

Lecture: Autotuning and Machine Learning

Instructor: Markus Püschel

TA: Gagandeep Singh, Daniele Spampinato, Alen Stojanov

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Overview

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

search used

no search used

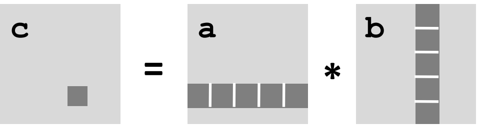
time of implementation

time of installation
platform known

time of use
problem parameters known

PhiPac/ATLAS: MMM Generator

Whaley, Bilmes, Demmel, Dongarra, …

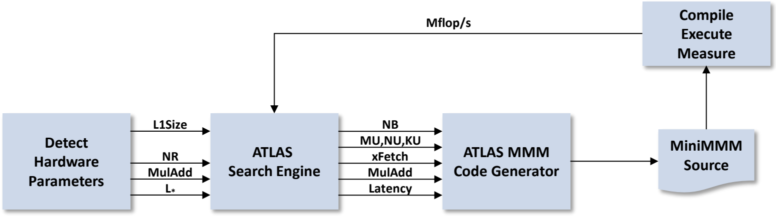


Blocking improves locality

```
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



Compile
Execute
Measure

Mflop/s

Detect
Hardware
Parameters

L1Size

NR

MulAdd

L_*

ATLAS
Search Engine

NB

MU,NU,KU

xFetch

MulAdd

Latency

ATLAS MMM
Code Generator

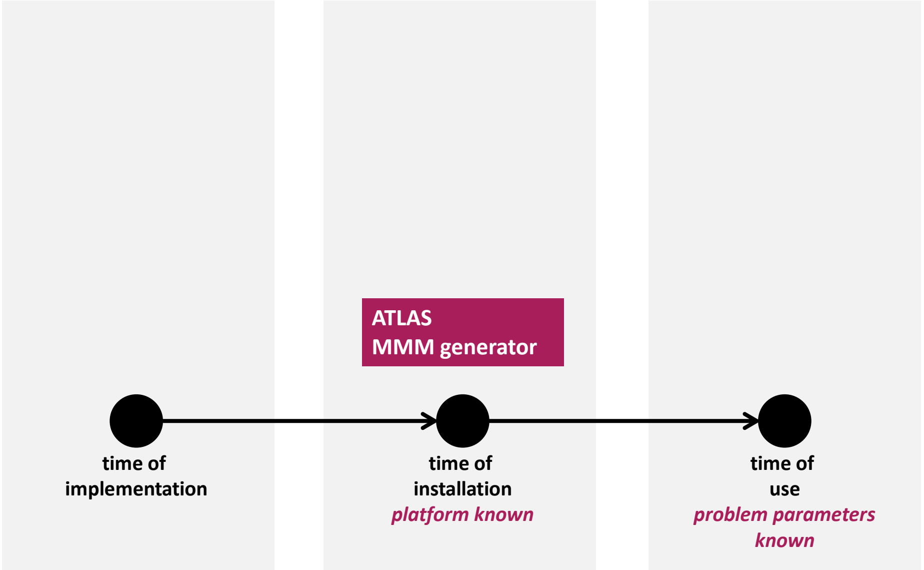
MiniMMM
Source

source: Pingali, Yotov, Cornell U.

---

search used

no search used



ATLAS
MMM generator

time of
implementation

time of
installation
platform known

time of
use
problem parameters known

FFTW: Discrete Fourier Transform (DFT)

Frigo, Johnson

Installation

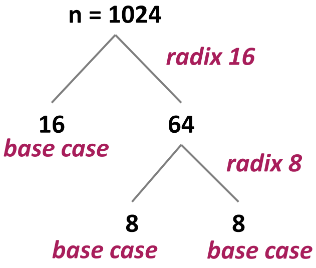
```
configure/make
```

Usage

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

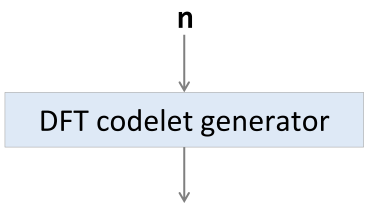
Twiddles

Search for fastest computation strategy



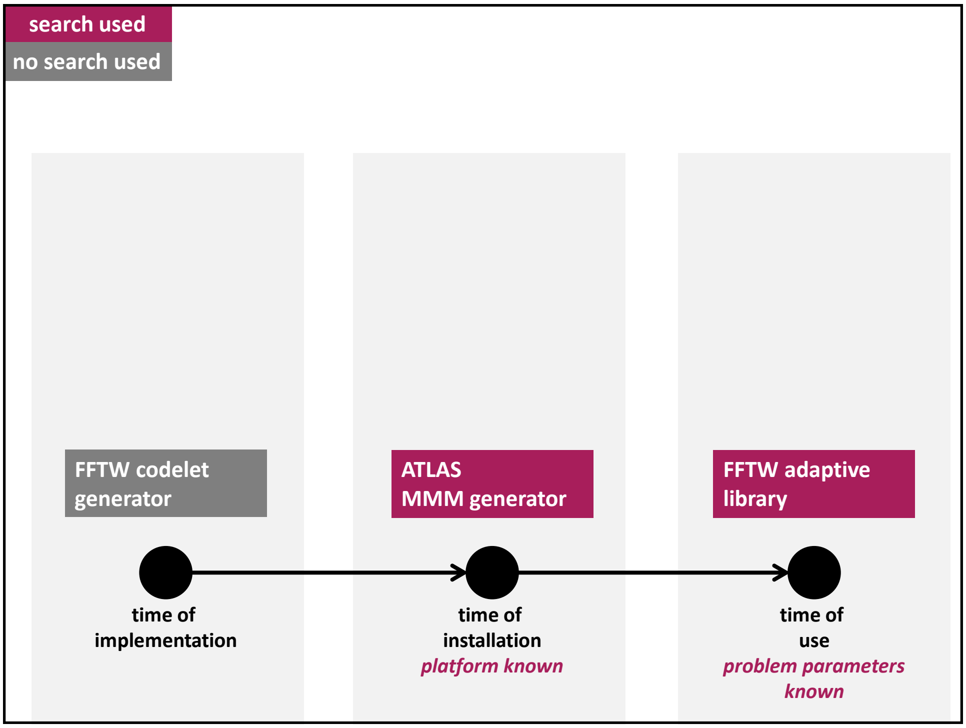
FFTW: Codelet Generator

Frigo



```
dft_n(*x, *y, …)
```

fixed size DFT functions
straightline code



search used

no search used

FFTW codelet generator

time of implementation

ATLAS MMM generator

time of installation

platform known

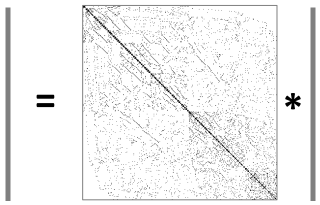
FFTW adaptive library

time of use

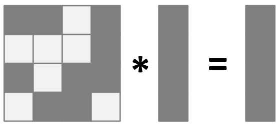
problem parameters known

OSKI: Sparse Matrix-Vector Multiplication

Vuduc, Im, Yelick, Demmel

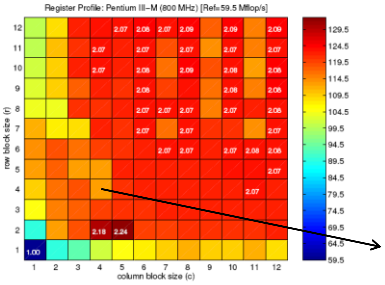


- **Blocking for registers:**
  - Improves locality (reuse of input vector)
  - But creates overhead (zeros in block)



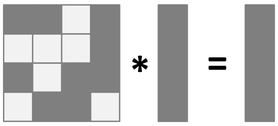
OSKI: Sparse Matrix-Vector Multiplication

Gain by blocking (dense MVM)



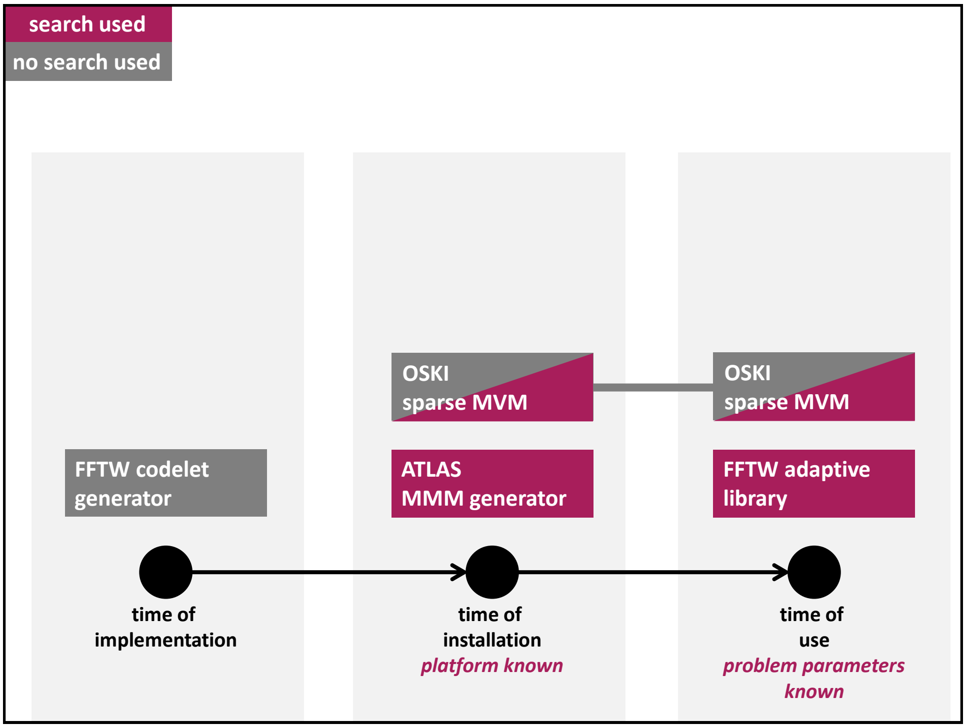
1.4

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

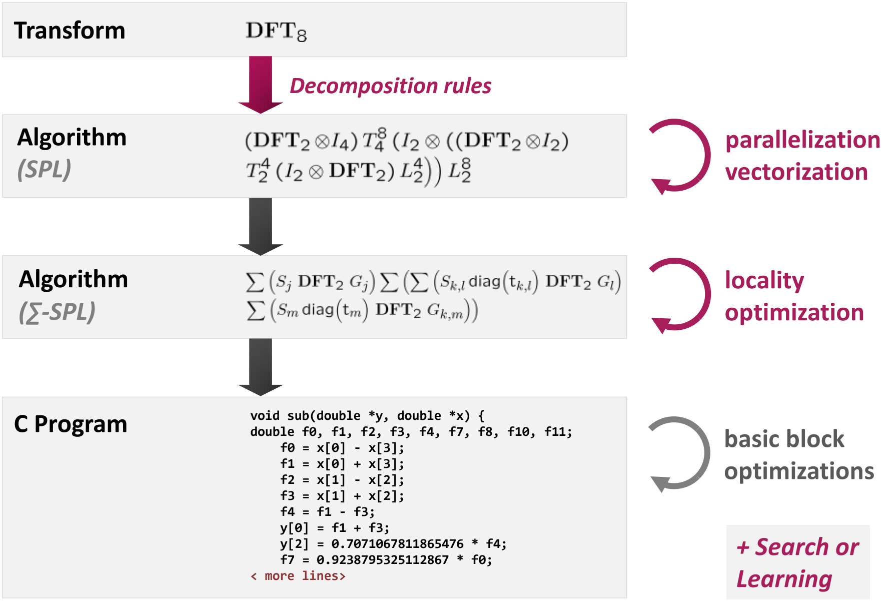
FFTW adaptive
library

time of
implementation

time of
installation
platform known

time of
use
*problem parameters
known*

Program Generation in Spiral



Transform $\mathbf{DFT}_8$

Decomposition rules

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
(∑-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>
```

basic block
optimizations

+ Search or Learning

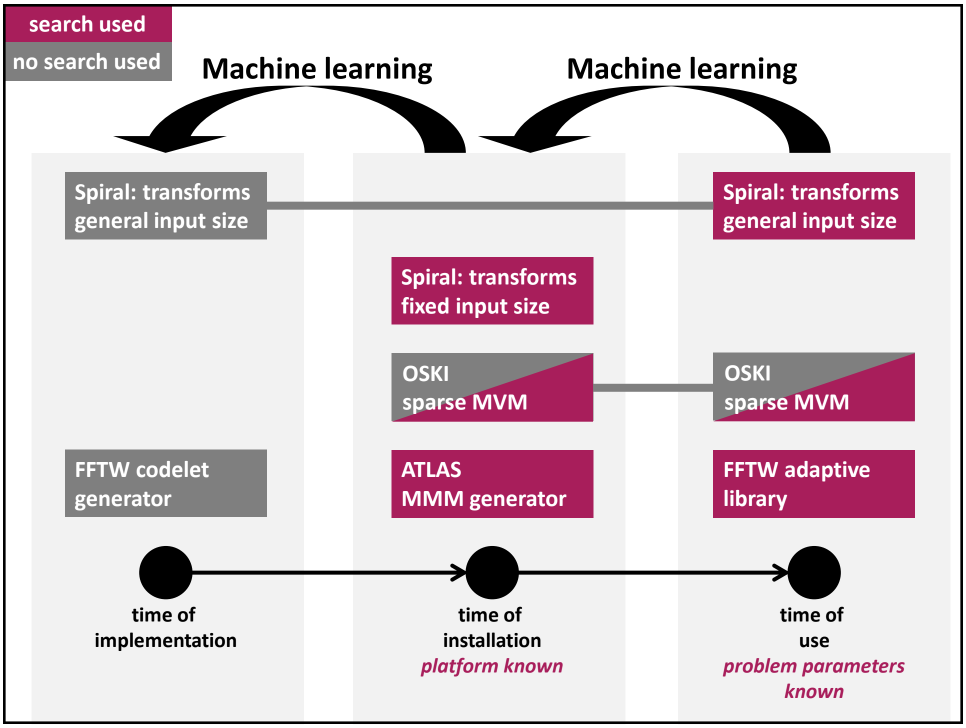
Spiral: Complete Automation for Transforms

- **Memory hierarchy optimization**
  Rewriting and search for algorithm selection
  Rewriting for loop optimizations
- **Vectorization**
  Rewriting
- **Parallelization**
  Rewriting

fixed input size code

- **Derivation of library structure**
  Rewriting
  Other methods

general input size library



search used

no search used

Machine learning

Machine learning

Spiral: transforms general input size

FFTW codelet generator

Spiral: transforms fixed input size

OSKI sparse MVM

ATLAS MMM generator

Spiral: transforms general input size

OSKI sparse MVM

FFTW adaptive library

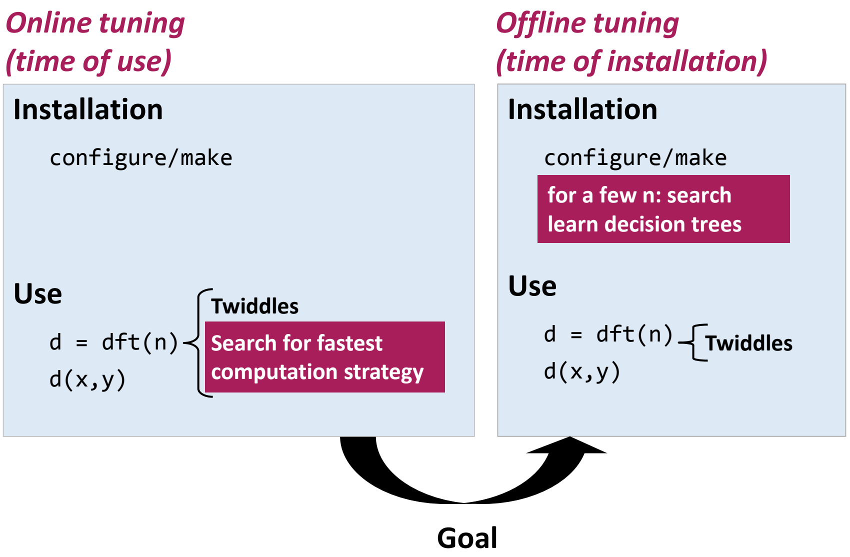
time of implementation

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]
- Use of machine learning II



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

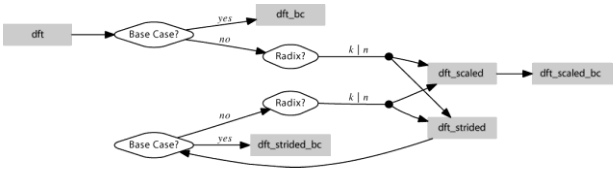
Use

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

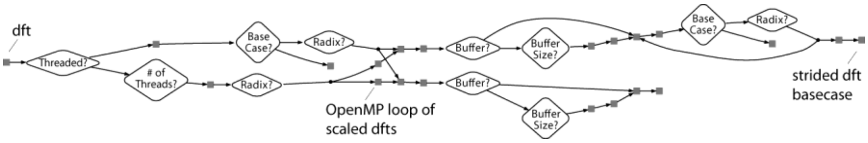
Goal

Library Structure: Examples

DFT: scalar code

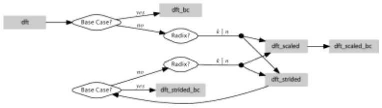


DFT: full-fledged (vectorized and parallel code)



Library Structure: Examples

DFT: scalar code

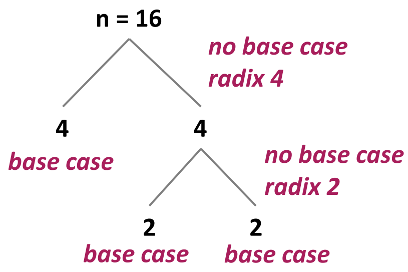


Recursive choice:

n = 2^k

base case?
radix?

Example selections for n = 16:



n = 16

no base case
radix 4

4
base case

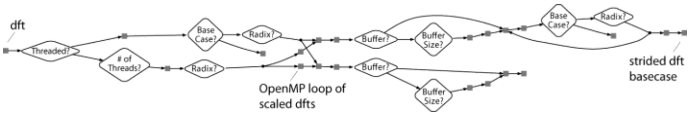
4
no base case
radix 2

2
base case

2
base case

Library Structure: Examples

DFT: full-fledged (vectorized and parallel code)



OpenMP loop of scaled dfts

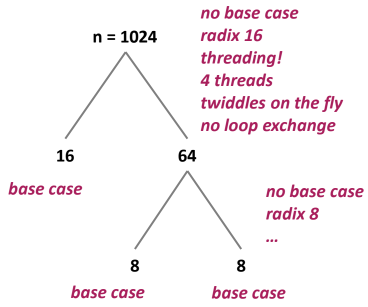
strided dft basecase

Recursive choice:

n = 2^k

base case?
radix?
threading?
#threads?
twiddles?
loop exchange?

Example selections for n = 1024:



n = 1024

no base case
radix 16
threading!
4 threads
twiddles on the fly
no loop exchange

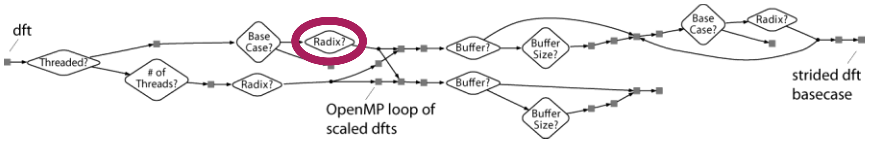
16
base case

64
no base case
radix 8
…

8
base case

8
base case

Our Work



Upon installation, generate decision trees for each choice

Example:

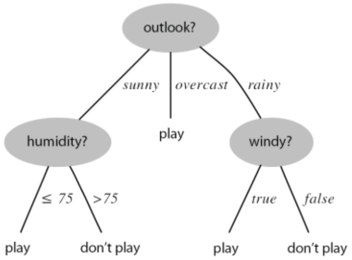
```
if ( n <= 65536 ) {
   if ( n <= 32 ) {
      if ( n <= 4 ) {return 2;}
      else {return 4;}
   }
   else {
      if ( n <= 1024 ) {
         if ( n <= 256 ) {return 8;}
         else {return 32;}
      }
      else {
      .................
```

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

C4.5



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)**

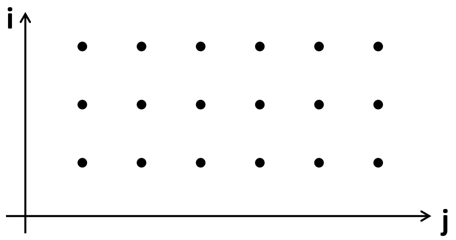
```
dft(int n, cpx *y, cpx *x)
dft_strided(int n, int istr, cpx *y, cpx *x)
dft_scaled(int n, int str, cpx *d, cpx *y, cpx *x)
```

- **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

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**

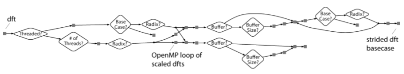
$n = 2^i 3^j$ = 2, 3, 4, 6, 9, 12, 16, 18, 24, 27, 32, …

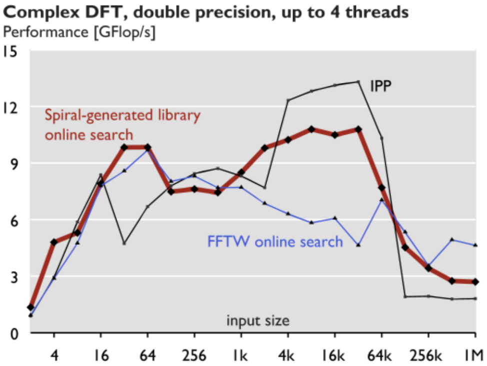


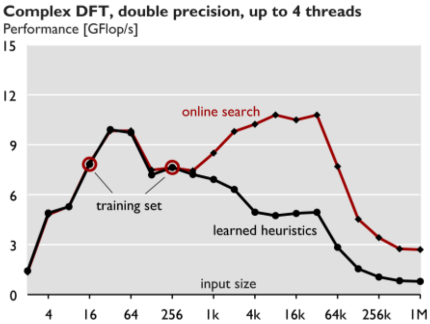
Compute i, j and add to features

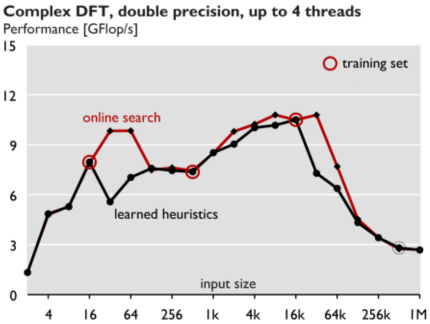
Experimental Setup

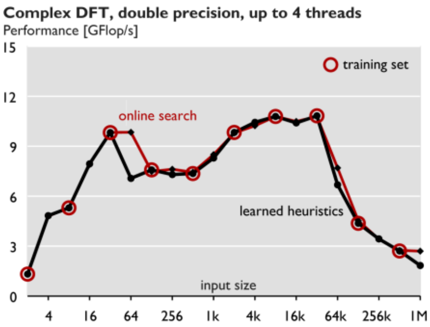
- 3GHz Intel Xeon 5160 (2 Core 2 Duos = 4 cores)
- Linux 64-bit, icc 10.1
- Libraries:
  - IPP 5.3
  - FFTW 3.2 alpha 2
  - Spiral-generated library




Complex DFT, double precision, up to 4 threads

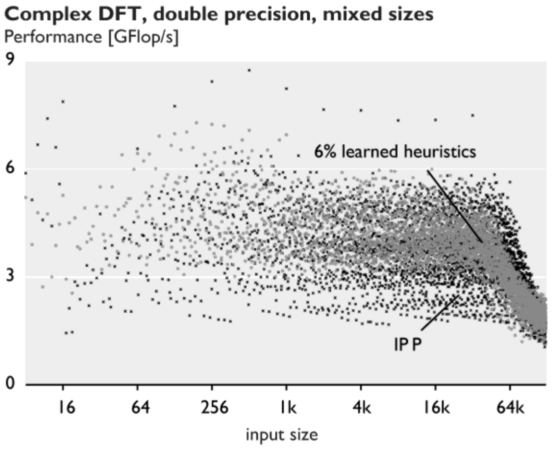

Complex DFT, double precision, up to 4 threads


Complex DFT, double precision, up to 4 threads


Complex DFT, double precision, up to 4 threads

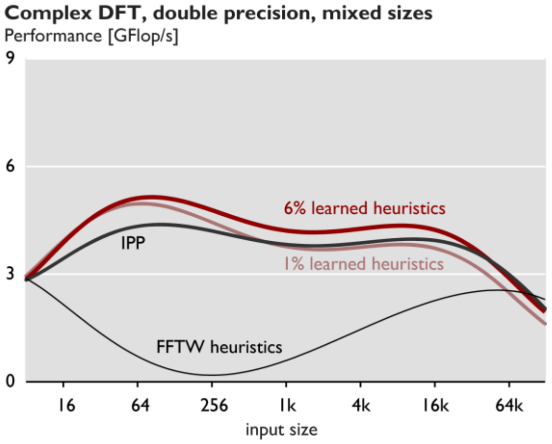
Learning works as expected

“All” Sizes



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

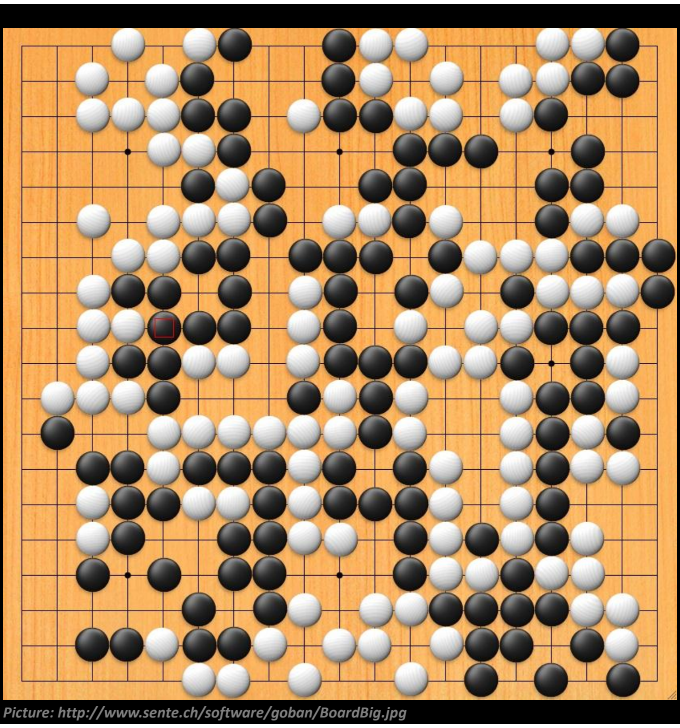
“All” Sizes



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

Overview

- Rough classification of autotuning efforts seen in course
- Use of machine learning I
- Use of machine learning II [de Mesmay et al., ICML 2009]



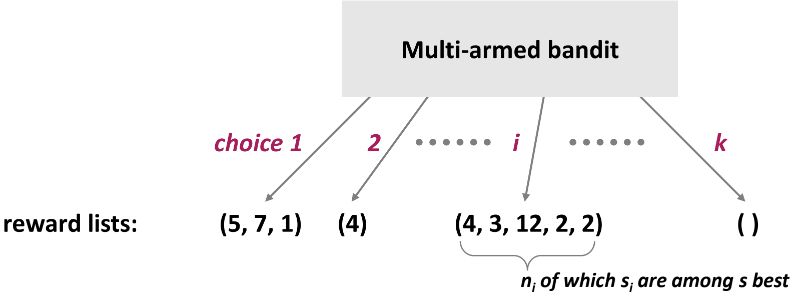
Picture: http://www.sente.ch/software/goban/BoardBig.jpg

Modeling Choice: Multi-armed Bandit

Which arm to pull next to maximize reward?

http://www.cardboardcutout.net/index.php?_a=product&product_id=115&cat_id=130

Modeling Choice: Multi-armed Bandit



Multi-armed bandit

choice 1 *2* •••••• *i* •••••• *k*

reward lists: **(5, 7, 1)** **(4)** **(4, 3, 12, 2, 2)** **( )**

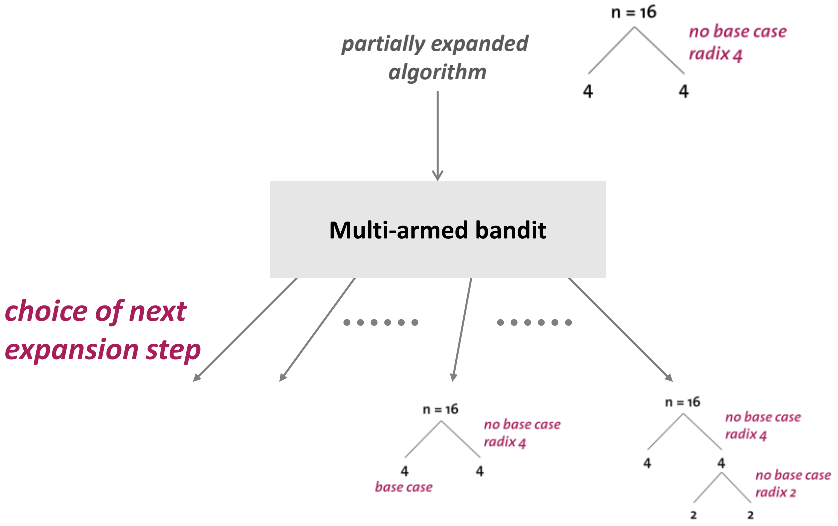
n_i of which s_i are among s best

Which arm to pull next to maximize reward?

$$i_{\text{best}} = \underset{1\le i\le k}{\operatorname{argmax}}\ h(s_i, n_i),$$

$$\text{with } h(s_i, n_i) = \begin{cases} \frac{s_i+\alpha+\sqrt{2s_i\alpha+\alpha^2}}{n_i}, & \text{if } n_i > 0 \\ \infty, & \text{else} \end{cases}$$

In Our Application

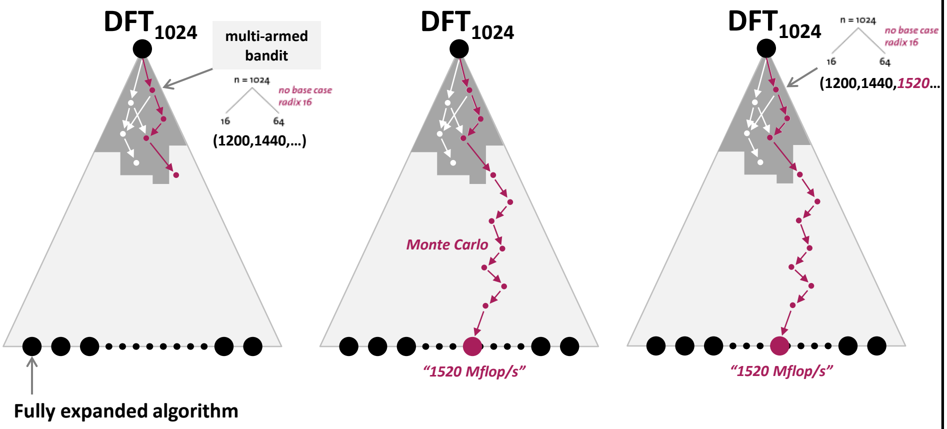


partially expanded algorithm

Multi-armed bandit

choice of next expansion step

Search Algorithm: TAG



Fully expanded algorithm

Descend *Evaluate* *Backpropagate*

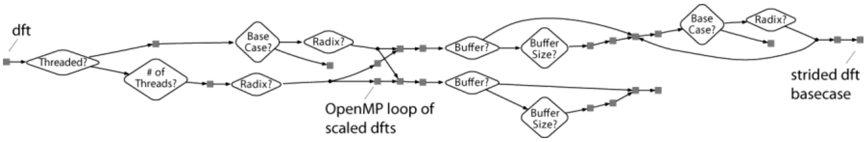
Experiments

- **Spiral-generated adaptive libraries**
  (similar to FFTW 3.x)

Recursive choice:

$n = 2^k$

base case?
radix?
threading?
#threads?
twiddles?
loop exchange?

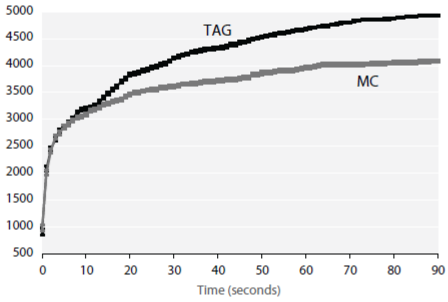


OpenMP loop of scaled dfts

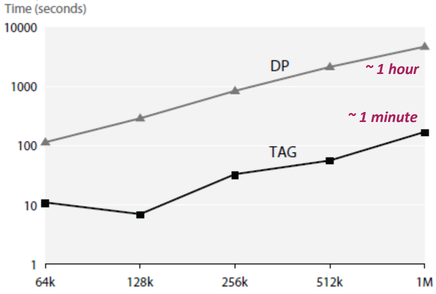
strided dft basecase

- **Intel Xeon 5160, 2 x dualcore, 3GHz**
- **Intel icc 10.1**
- **FFTW 3.2alpha, Intel IPP 5.3**

Comparison between anytime strategies for DFT 256k



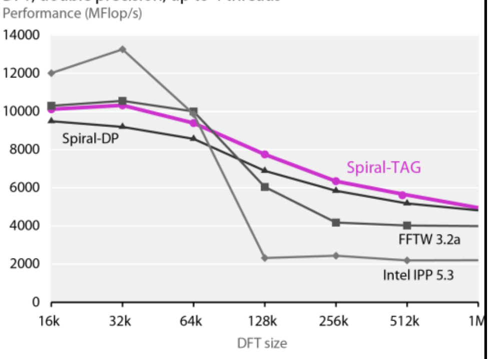
Comparison of search time between DP and TAG



~ 1 hour

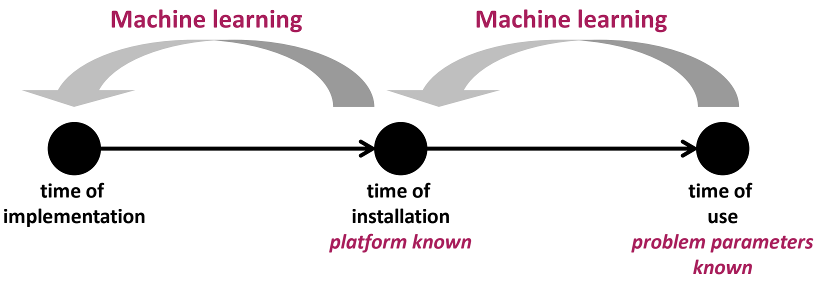
~ 1 minute

DFT, double precision, up to 4 threads



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



Machine learning

Machine learning

time of implementation

time of installation
platform known

time of use
problem parameters known

Research Questions

- **How to automate the production of fastest numerical code?**
  - *Domain-specific languages*
  - *Rewriting*
  - *Compilers*
  - *Machine Learning*
- **What program language features help with program generation?**
- **What environment should be used to build generators?**
- **How to represent mathematical functionality?**
- **How to formalize the mapping to fast code?**
- **How to handle various forms of parallelism?**
- **How to integrate into standard work flows?**