Overview

- Rough classification of autotuning efforts seen in course
- Use of machine learning I
- Use of machine learning II
**PhiPac/ATLAS: MMM Generator**

*Whaley, Bilmes, Demmel, Dongarra,* ...

Blocking improves locality

```c
double *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

Detect Hardware Parameters → ATLAS Search Engine → ATLAS MMM Code Generator → MiniMMM Source

Compile
Execute
Measure

Mflop/s

source: Pingali, Yotov, Cornell U.

search used
no search used

ATLAS MMM generator

time of implementation

platform known

problem parameters known

How to write fast numerical code

Spring 2015
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, Ilm, 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

\[ \begin{align*}
\text{Gain by blocking} & : 1.4 \\
\text{Overhead by blocking} & : 16/9 = 1.77 \\
1.4/1.77 & = 0.79 \text{ (no gain)}
\end{align*} \]

search used
no search used

- **FFTW codelet generator**
- **OSKI sparse MVM**
- **ATLAS MMM generator**
- **FFT W adaptive library**

- **time of implementation**
- **time of installation**
  - platform known
- **time of use**
  - problem parameters known
Program Generation in Spiral

Transform: $DFT_B$

Decomposition rules

Algorithm ($SPL$): $(DFT_2 \otimes I_4) T^B_2 (I_2 \otimes ((DFT_2 \otimes I_2) T^B_2 (I_2 \otimes DFT_2) L^B_2)) L^B_2$

Algorithm ($\Sigma$-$SPL$): $\sum (S_j DFT_2 G_j) \sum (S_k \text{diag}(t_k) DFT_2 G_j) \sum (S_m \text{diag}(t_m) DFT_2 G_k,m)$

C Program:

```c
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] = f2 + f3;
    y[2] = 0.7071067811865476 * f4;
    f7 = 0.9238795325112867 * f0;
    // more lines
}
```

+ 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

- Derivation of library structure
  - Rewriting
  - Other methods

fixed input size code

general input size library
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**  
\[ d = \text{dft}(n) \]
\[ d(x,y) \]

**Twiddles**
Search for fastest computation strategy

**Offline tuning**  
*(time of installation)*

**Installation**
configure/make

**Use**  
\[ d = \text{dft}(n) \]
\[ d(x,y) \]

**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 \( n = 16 \):

- \( n = 16 \)
  - no base case
  - radix 4
  - base case
  - radix 2
  - base case

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 \( n = 1024 \):

- \( n = 1024 \)
  - no base case
  - radix 16
  - threading!
  - 4 threads
  - twiddles on the fly
  - no loop exchange
  - base case
  - radix 8
  - ...
Our Work

Upon installation, generate decision trees for each choice

Example:
if \( n \leq 65536 \) {
  if \( n \leq 32 \) {
    if \( n \leq 4 \) {return 2;}
    else {return 4;}
  } else {
    if \( n \leq 1024 \) {
      if \( n \leq 256 \) {return 0;}
      else {return 32;}
    } else {
      // 
      // 
      // 
      // 
    }
  }
}

Statistical Classification: C4.5

Features (events)

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Windy</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>sunny</td>
<td>85</td>
<td>85</td>
<td>false</td>
<td>don’t play</td>
</tr>
<tr>
<td>sunny</td>
<td>80</td>
<td>90</td>
<td>true</td>
<td>don’t play</td>
</tr>
<tr>
<td>overcast</td>
<td>83</td>
<td>78</td>
<td>false</td>
<td>play</td>
</tr>
<tr>
<td>rain</td>
<td>70</td>
<td>96</td>
<td>false</td>
<td>play</td>
</tr>
<tr>
<td>rain</td>
<td>68</td>
<td>80</td>
<td>false</td>
<td>play</td>
</tr>
<tr>
<td>rain</td>
<td>65</td>
<td>70</td>
<td>true</td>
<td>don’t play</td>
</tr>
<tr>
<td>overcast</td>
<td>64</td>
<td>65</td>
<td>true</td>
<td>play</td>
</tr>
<tr>
<td>sunny</td>
<td>72</td>
<td>95</td>
<td>false</td>
<td>don’t play</td>
</tr>
<tr>
<td>sunny</td>
<td>69</td>
<td>70</td>
<td>false</td>
<td>play</td>
</tr>
<tr>
<td>sunny</td>
<td>75</td>
<td>80</td>
<td>false</td>
<td>play</td>
</tr>
<tr>
<td>sunny</td>
<td>75</td>
<td>70</td>
<td>true</td>
<td>play</td>
</tr>
<tr>
<td>overcast</td>
<td>72</td>
<td>90</td>
<td>true</td>
<td>play</td>
</tr>
<tr>
<td>overcast</td>
<td>81</td>
<td>75</td>
<td>false</td>
<td>play</td>
</tr>
<tr>
<td>rain</td>
<td>71</td>
<td>80</td>
<td>true</td>
<td>don’t play</td>
</tr>
</tbody>
</table>

\[
P(\text{play}|\text{windy}=\text{false}) = \frac{6}{8}
\]
\[
P(\text{don’t play}|\text{windy}=\text{false}) = \frac{2}{8}
\]
\[
P(\text{play}|\text{windy}=\text{true}) = \frac{1}{2}
\]
\[
P(\text{don’t play}|\text{windy}=\text{true}) = \frac{1}{2}
\]

\[
H(\text{windy}=\text{false}) = 0.81
\]
\[
H(\text{windy}=\text{true}) = 1.0
\]

Entropy of Features

\[
H(\text{windy}) = 0.89
\]
\[
H(\text{outlook}) = 0.69
\]
\[
H(\text{humidity}) = ...
\]
Application to Libraries

- Features = arguments of functions (except variable pointers)

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

$$n = 2^i3^j = 2, 3, 4, 6, 9, 12, 16, 18, 24, 27, 32, \ldots$$

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

Learning works as expected
“All” Sizes

Complex DFT, double precision, mixed sizes
Performance [GFlop/s]

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

“All” Sizes

Complex DFT, double precision, mixed sizes
Performance [GFlop/s]

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

Which arm to pull next to maximize reward?

\[ i_{best} = \arg \max_{1 \leq i \leq k} h(s_i, n_i), \]

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

choice of next expansion step

Multi-armed bandit

Search Algorithm: TAG

DFT_1024

Fully expanded algorithm

Descend

Evaluate

Backpropagate

© Markus Püschel
Computer Science
Experiments

- Spiral-generated adaptive libraries (similar to FFTW 3.x)
- Intel Xeon 5160, 2 x dualcore, 3GHz
- Intel icc 10.1
- FFTW 3.2alpha, Intel IPP 5.3

Recursive choice:

\[ n = 2^k \]

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

~ 1 hour
~ 1 minute
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

- 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?