Autotuning Programs with Algorithmic Choice

Jason Ansel

MIT - CSAIL

December 18, 2013
High Performance Search Problem

- Parallelism
High Performance Search Problem

- Parallelism Performance
  - Exploiting parallelism is necessary but not sufficient
High Performance Search Problem

Performance search space:

- Parallelism Performance
  - Exploiting parallelism is necessary but not sufficient
- Performance is a multi-dimensional search problem
- Normally done by expert programmers
- Optimization decisions often change program results
High Performance Search Problem

Goal of this work
To automate the process of program optimization to create programs that can adapt to changing environments and goals.
High Performance Search Problem

Goal of this work
To automate the process of program optimization to create programs that can adapt to changing environments and goals.

- Language level solutions for concisely representing algorithmic choice spaces.
- Processes and compilation techniques to manage and explore these spaces.
- Autotuning techniques to efficiently solve these search problems.
Research Covered in This Talk

- The PetaBricks programming language: algorithmic choice at the language level [PLDI’09]
- Language level support for variable accuracy [CGO’11]
- Automated construction of multigrid V-cycles [SC’09]
- Code generation and autotuning for heterogeneous CPU/GPU mix of parallel processing units [ASPLOS’13]
- Solution for input sensitivity based on adaptive overhead-aware classifiers [Under review]
- OpenTuner: an extensible framework for program autotuning [Under review]
Research Covered in This Talk

• The PetaBricks programming language: algorithmic choice at the language level [PLDI’09]
• Language level support for variable accuracy [CGO’11]
• Automated construction of multigrid V-cycles [SC’09]
• Code generation and autotuning for heterogeneous CPU/GPU mix of parallel processing units [ASPLOS’13]
• Solution for input sensitivity based on adaptive overhead-aware classifiers [Under review]
• OpenTuner: an extensible framework for program autotuning [Under review]
• Won’t be talking about work in: ASPLOS’09, ASPLOS’12, GECCO’11, IPDPS’09, PLDI’11, and many others
A Motivating Example for Algorithmic Choice

- How would you write a *fast* sorting algorithm?
A Motivating Example for Algorithmic Choice

- How would you write a fast sorting algorithm?
  - Insertion sort
  - Quick sort
  - Merge sort
  - Radix sort
A Motivating Example for Algorithmic Choice

- How would you write a *fast* sorting algorithm?
  - Insertion sort
  - Quick sort
  - Merge sort
  - Radix sort
- Poly-algorithms
std::stable_sort

/usr/include/c++/4.5.2/bits/stl_algo.h lines 3350-3367
/// This is a helper function for the stable sorting routines.

```cpp
template<typename RandomAccessIterator>
void
__inplace_stable_sort(RandomAccessIterator __first, RandomAccessIterator __last)
{
    if (__last - __first < 15)
    {
        std::__insertion_sort(__first, __last);
        return;
    }
    RandomAccessIterator __middle = __first + (__last - __first) / 2;
    std::__inplace_stable_sort(__first, __middle);
    std::__inplace_stable_sort(__middle, __last);
    std::__merge_without_buffer(__first, __middle, __last, __middle - __first,
                                 __last - __middle);
}
```
std::stable_sort

/usr/include/c++/4.5.2/bits/stl_algo.h lines 3350-3367
/// This is a helper function for the stable sorting routines.

```cpp
template<typename RandomAccessIterator>
void
__inplace_stable_sort(RandomAccessIterator __first,
                      RandomAccessIterator __last)
{
    if (__last - __first < 15)
    {
        std::__insertion_sort(__first, __last);
        return;
    }
    RandomAccessIterator __middle = __first + (__last - __first) / 2;
    std::__inplace_stable_sort(__first, __middle);
    std::__inplace_stable_sort(__middle, __last);
    std::__merge_without_buffer(__first, __middle, __last,
                                 __middle - __first,
                                 __last - __middle);
}
```
Why 15?

- Why 15?
Why 15?

- Why 15?
- Dates back to at least 2000 (June 2000 SGI release)
- Still in current C++ STL shipped with GCC
- cutoff = 15 survived 10+ years
- In the source code for millions\(^1\) of C++ programs
- There is nothing the compiler can do about it

\(^1\)Any C++ program with "#include <algorithm>", conservative estimate based on: http://c2.com/cgi/wiki?ProgrammingLanguageUsageStatistics
Is 15 The Right Number?

- The best cutoff (CO) changes
- Depends on competing costs:
  - Cost of computation (< operator, call overhead, etc)
  - Cost of communication (swaps)
  - Cache behavior (misses, prefetcher, locality)
Is 15 The Right Number?

- The best cutoff (CO) changes
- Depends on competing costs:
  - Cost of computation (< operator, call overhead, etc)
  - Cost of communication (swaps)
  - Cache behavior (misses, prefetcher, locality)

- Sorting 100000 doubles with std::stable_sort:
  - $CO \approx 200$ optimal on a Phenom 905e (15% speedup)
  - $CO \approx 400$ optimal on a Opteron 6168 (15% speedup)
  - $CO \approx 500$ optimal on a Xeon E5320 (34% speedup)
  - $CO \approx 700$ optimal on a Xeon X5460 (25% speedup)

- If the best cutoff has changed, perhaps best algorithm has also changed
Algorithmic Choice

- Compiler’s hands are tied, it is stuck with 15
- Need a better way to represent algorithmic choices
- PetaBricks is the first language with support for algorithmic choice
Sort in PetaBricks

Language

```c
function Sort to out[n]
from in[n]
{
    either {
        InsertionSort(out, in);
    } or {
        QuickSort(out, in);
    } or {
        MergeSort(out, in);
    } or {
        RadixSort(out, in);
    }
}
```
Sort in PetaBricks

Language

```plaintext
function Sort
  to out[n]
  from in[n]
{
  either {
    InsertionSort(out, in);
  } or {
    QuickSort(out, in);
  } or {
    MergeSort(out, in);
  } or {
    RadixSort(out, in);
  }
}

⇒ Representation

⇒ Decision tree
synthesized by our autotuner
Decision Trees

Optimized for a Xeon E7340 (8 cores):

- $N < 600$: Insertion Sort
- $N < 1420$: Quick Sort
- $N < 1420$: Merge Sort (2-way)
Decision Trees

Optimized for Sun Fire T200 Niagara (8 cores):

- $N < 1461$
- $N < 2400$
- Merge Sort (4-way)
- Merge Sort (2-way)
- Merge Sort (8-way)
- Merge Sort (16-way)
- $N < 75$
Sort Algorithm Timings

On an 8-way Xeon E7340 system

---

2 On an 8-way Xeon E7340 system
Iteration Order Choices

- Many other choices related to execution order
  - By rows?
  - By columns?
  - Diagonal? Reverse order? Blocked?
  - Parallel?

- Choices both within a single (possibly parallel) task and between different tasks
Iteration Order Choices

- Many other choices related to execution order
  - By rows?
  - By columns?
  - Diagonal? Reverse order? Blocked?
  - Parallel?

- Choices both within a single (possibly parallel) task and between different tasks

- This is main motivation for a new language as opposed to a library
Synthesized Outer Control Flow

- PetaBricks programs have synthesized outer control flow
  - Declarative (data flow like) outer syntax
  - Imperative inner code
- Programs start as completely parallel
- Added dependencies restrict the space of legal executions
- May only access data explicitly depended on

Parallel loop

\[
X.\text{cell}(i) \text{ from } () \{ \ldots \}
\]

Sequential loop

\[
X.\text{cell}(i) \text{ from } (X.\text{cell}(i-1) \text{ left}) \{ \ldots \}
\]
Matrix Multiply

```plaintext
transform MatrixMultiply to AB[w,h]
from A[c,h], B[w,c]
{
    AB.cell(x, y) from (A.row(y) a, B.column(x) b) {
        return dot(a, b);
    }
}
```
Matrix Multiply

```
transform MatrixMultiply
to AB[w,h]
from A[c,h], B[w,c]
{
  AB.cell(x, y) from (A.row(y) a, B.column(x) b) {
    return dot(a, b);
  }
}

to (AB.region(x, y, x + 4, y + 4) out)
from (A.region(0, y, c, y + 4) a,
      B.region(x, 0, x + 4, c) b) {
    // ... compute 4 x 4 block ...
  }
}
```
**Strassen Matrix Multiply**

```plaintext
transform Strassen
to AB[n,n]
from A[n,n], B[n,n]
using M1[n/2, n/2], M2[n/2, n/2], M3[n/2, n/2], M4[n/2, n/2],
     M5[n/2, n/2], M6[n/2, n/2], M7[n/2, n/2]
{
    to(M1 m1)
    from(A.region(0, 0, n/2, n/2) a11,
         A.region(n/2, n/2, n, n) a22,
         B.region(0, 0, n/2, n/2) b11,
         B.region(n/2, n/2, n, n) b22)
    using(t1[n/2, n/2], t2[n/2, n/2]) {
        spawn MatrixAdd(t1, a11, a22);
        spawn MatrixAdd(t2, b11, b22);
        sync;
        Strassen(m1, t1, t2);
    }
    ...
    // Compute one quadrant of output with strassen decomposition
to(AB.region(n/2, 0, n, n/2) c12) from(M3 m3, M5 m5){
    MatrixAdd(c12, m3, m5);
}
    ...
    // Or, compute element in output directly (same as last slide)
    AB.cell(x, y) from(A.row(y) a, B.column(x) b){
        return dot(a, b);
    }
}
```
Variable Accuracy Algorithms

- Many problems don’t have a single correct answer, optimizations often trade-off accuracy and performance.
  - Soft computing
  - DSP algorithms
  - Iterative algorithms
Variable Accuracy Algorithms

- Many problems don’t have a single correct answer, optimizations often trade-off accuracy and performance.
  - Soft computing
  - DSP algorithms
  - Iterative algorithms

- Variable accuracy, supported in the PetaBricks language, is a fundamental part of algorithmic choice which enables new classes of programs to be represented.
K-Means Example

```python
transform kmeans
from Points[n,2] // Array of points (each column
    // stores x and y coordinates)
using Centroids[sqrt(n),2]
to Assignments[n]
{
    // Rule 1:
    // One possible initial condition: Random
    // set of points
    to(Centroids .column(i) c) from(Points p) {
        c=p .column(rand(0,n))
    }

    // Rule 2:
    // Another initial condition: Centerplus initial
    // centers (kmeans++)
    to(Centroids c) from(Points p) {
        CenterPlus(c, p);
    }

    // Rule 3:
    // The kmeans iterative algorithm
    to(Assignments a) from(Points p, Centroids c) {
        while (true) {
            int change;
            AssignClusters(a, change, p, c, a);
            if (change==0) return; // Reached fixed point
            NewClusterLocations(c, p, a);
        }
    }
}
K-Means Example (Variable Accuracy)

```plaintext
transform kmeans
accuracy_metric kmeansaccuracy
accuracy_variable k
from Points[n,2] // Array of points (each column
  // stores x and y coordinates)
using Centroids[k,2]
to Assignments[n]

... // Rule 3:
  // The kmeans iterative algorithm
to(Assignments a) from(Points p, Centroids c) {
  for_enough {
    int change;
    AssignClusters(a, change, p, c, a);
    if (change==0) return; // Reached fixed point
    NewClusterLocations(c, p, a);
  }
}
transform kmeansaccuracy
from Assignments[n], Points[n,2]
to Accuracy
{
  Accuracy from(Assignments a, Points p){
    return sqrt(2*n/SumClusterDistanceSquared(a,p));
  }
}
```
Semantics of Variable Accuracy

Running the *accuracy_metric* on the output will return a value that, in expectation, exceeds the *accuracy target* more than $P$ percent of the time.
Semantics of Variable Accuracy

Running the `accuracy_metric` on the output will return a value that, in expectation, exceeds the `accuracy target` more than $P$ percent of the time.

- Expected distribution of accuracy measured during autotuning time, not at runtime.
- When *fixed accuracy* code calls *variable accuracy* code, an accuracy target must be specified.
- When *variable accuracy* code call code containing *variable accuracy* components, only the outer most accuracy target will be honored.
A Brief Multigrid Intro

- Used to iteratively solve PDEs over a gridded domain
- Relaxations update points using neighboring values (stencil computations)
- Restrictions and Interpolations compute new grid with coarser or finer discretization
Standard Cycle Shapes

- Cycle shapes effect accuracy and performance
  - Equation, accuracy target, data, and execution platform effect efficacy of different shapes
- Entire papers published about new cycle shapes!
Standard Cycle Shaps

- Cycle shapes effect accuracy and performance
  - Equation, accuracy target, data, and execution platform effect efficacy of different shapes
- Entire papers published about new cycle shapes!
- We fundamentally change the status quo in this domain
  - Define the search space of cycle shapes once
  - Autotune to find a cycle shape tailored to your problem
Choice Space of Multigrid

- Direct
  - Estimate Phase
  - Interpolate

- Iterative
  - Multigrid Recursion
  - Interpolate
  - Restrict
Autotuned V-cycle Shapes
Dynamic Programming Technique for Autotuning Multigrid
Dynamic Programming Technique for Autotuning Multigrid
Dynamic Programming Technique for Autotuning Multigrid

- Partition accuracy space into discrete levels
Dynamic Programming Technique for Autotuning Multigrid

- Partition accuracy space into discrete levels
Dynamic Programming Technique for Autotuning Multigrid

- Partition accuracy space into discrete levels
- Base space of candidate algorithms on optimal algorithms from coarser level
2D Poisson’s Equation (uses Multigrid)
More Variable Accuracy Results

- Clustering
- Bin Packing
- Image Compression
- 3D Helmholtz
- 2D Poisson
- Preconditioner
Results on Different Systems

Test Systems

<table>
<thead>
<tr>
<th>Codename</th>
<th>CPU(s)</th>
<th>Cores</th>
<th>GPU</th>
<th>OpenCL Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop</td>
<td>Core i7 920 @2.67GHz</td>
<td>4</td>
<td>NVIDIA Tesla C2070</td>
<td>CUDA Toolkit 3.2</td>
</tr>
<tr>
<td>Server</td>
<td>4× Xeon X7550 @2GHz</td>
<td>32</td>
<td>None</td>
<td>AMD APP SDK 2.5</td>
</tr>
<tr>
<td>Laptop</td>
<td>Core i5 2520M @2.5GHz</td>
<td>2</td>
<td>AMD Radeon HD 6630M</td>
<td>Xcode 4.2</td>
</tr>
</tbody>
</table>

Benchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th># Possible Configs</th>
<th>Generated OpenCL Kernels</th>
<th>Mean Autotuning Time</th>
<th>Testing Input Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeparableConv.</td>
<td>$10^{1358}$</td>
<td>9</td>
<td>3.82 hours</td>
<td>$3520^2$</td>
</tr>
<tr>
<td>Black-Sholes</td>
<td>$10^{130}$</td>
<td>1</td>
<td>3.09 hours</td>
<td>500000</td>
</tr>
<tr>
<td>Poisson2D SOR</td>
<td>$10^{1358}$</td>
<td>25</td>
<td>15.37 hours</td>
<td>$2048^2$</td>
</tr>
<tr>
<td>Sort</td>
<td>$10^{920}$</td>
<td>7</td>
<td>3.56 hours</td>
<td>$2^{20}$</td>
</tr>
<tr>
<td>Strassen</td>
<td>$10^{1509}$</td>
<td>9</td>
<td>3.05 hours</td>
<td>$1024^2$</td>
</tr>
<tr>
<td>SVD</td>
<td>$10^{2435}$</td>
<td>8</td>
<td>1.79 hours</td>
<td>$256^2$</td>
</tr>
<tr>
<td>Tridiagonal Solver</td>
<td>$10^{1040}$</td>
<td>8</td>
<td>5.56 hours</td>
<td>$1024^2$</td>
</tr>
</tbody>
</table>
Results on Different Systems

Test Systems

<table>
<thead>
<tr>
<th>Codename</th>
<th>CPU(s)</th>
<th>Cores</th>
<th>GPU</th>
<th>OpenCL Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop</td>
<td>Core i7 920 @2.67GHz</td>
<td>4</td>
<td>NVIDIA Tesla C2070</td>
<td>CUDA Toolkit 3.2</td>
</tr>
<tr>
<td>Server</td>
<td>4× Xeon X7550 @2GHz</td>
<td>32</td>
<td>None</td>
<td>AMD APP SDK 2.5</td>
</tr>
<tr>
<td>Laptop</td>
<td>Core i5 2520M @2.5GHz</td>
<td>2</td>
<td>AMD Radeon HD 6630M</td>
<td>Xcode 4.2</td>
</tr>
</tbody>
</table>

Benchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th># Possible Configs</th>
<th>Generated OpenCL Kernels</th>
<th>Mean Autotuning Time</th>
<th>Testing Input Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeparableConv.</td>
<td>$10^{1358}$</td>
<td>9</td>
<td>3.82 hours</td>
<td>$3520^2$</td>
</tr>
<tr>
<td>Black-Sholes</td>
<td>$10^{130}$</td>
<td>1</td>
<td>3.09 hours</td>
<td>500000</td>
</tr>
<tr>
<td>Poisson2D SOR</td>
<td>$10^{1358}$</td>
<td>25</td>
<td>15.37 hours</td>
<td>$2048^2$</td>
</tr>
<tr>
<td>Sort</td>
<td>$10^{920}$</td>
<td>7</td>
<td>3.56 hours</td>
<td>$2^{20}$</td>
</tr>
<tr>
<td>Strassen</td>
<td>$10^{1509}$</td>
<td>9</td>
<td>3.05 hours</td>
<td>$1024^2$</td>
</tr>
<tr>
<td>SVD</td>
<td>$10^{2435}$</td>
<td>8</td>
<td>1.79 hours</td>
<td>$256^2$</td>
</tr>
<tr>
<td>Tridiagonal Solver</td>
<td>$10^{1040}$</td>
<td>8</td>
<td>5.56 hours</td>
<td>$1024^2$</td>
</tr>
</tbody>
</table>
Separable Convolution (width=7)

<table>
<thead>
<tr>
<th>SeparableConv.</th>
<th>Desktop Config</th>
<th>Server Config</th>
<th>Laptop Config</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1D kernel+local memory on GPU</td>
<td>1D kernel on OpenCL</td>
<td>2D kernel+local memory on GPU</td>
</tr>
</tbody>
</table>
Separable Convolution (width=7)

### Execution Time (Normalized)

<table>
<thead>
<tr>
<th></th>
<th>Desktop Config</th>
<th>Server Config</th>
<th>Laptop Config</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SeparableConv.</strong></td>
<td>1D kernel+local memory on GPU</td>
<td>1D kernel on OpenCL</td>
<td>2D kernel+local memory on GPU</td>
</tr>
<tr>
<td><strong>Desktop</strong></td>
<td><img src="Desktop" alt="Graph" /></td>
<td><img src="Server" alt="Graph" /></td>
<td><img src="Laptop" alt="Graph" /></td>
</tr>
<tr>
<td><strong>Server</strong></td>
<td><img src="Desktop" alt="Graph" /></td>
<td><img src="Server" alt="Graph" /></td>
<td><img src="Laptop" alt="Graph" /></td>
</tr>
<tr>
<td><strong>Laptop</strong></td>
<td><img src="Desktop" alt="Graph" /></td>
<td><img src="Server" alt="Graph" /></td>
<td><img src="Laptop" alt="Graph" /></td>
</tr>
</tbody>
</table>
Separable Convolution (width=7)

<table>
<thead>
<tr>
<th></th>
<th>Desktop Config</th>
<th>Server Config</th>
<th>Laptop Config</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SeparableConv.</strong></td>
<td>1D kernel+local memory on GPU</td>
<td>1D kernel on OpenCL</td>
<td>2D kernel+local memory on GPU</td>
</tr>
</tbody>
</table>
**Poisson 2D SOR**

<table>
<thead>
<tr>
<th></th>
<th>Desktop Config</th>
<th>Server Config</th>
<th>Laptop Config</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Poisson2D SOR</strong></td>
<td>Split on CPU followed by compute on GPU</td>
<td>Split some parts on OpenCL followed by compute on CPU</td>
<td>Split on CPU followed by compute on GPU</td>
</tr>
</tbody>
</table>
Singular Value Decomposition (SVD)

First phase: task parallelism between CPU/GPU; matrix multiply: 8-way parallel recursive decomposition on CPU, call LAPACK when $< 42 \times 42$

First phase: all on CPU; matrix multiply: 8-way parallel recursive decomposition on CPU, call LAPACK when $< 170 \times 170$

First phase: all on CPU; matrix multiply: 4-way parallel recursive decomposition on CPU, call LAPACK when $< 85 \times 85$
Results Takeaways

- Different configurations are required for best performance on different systems
- Not just changing block sizes
- Can not be easily solved by a simple heuristic
- Motivates the need for algorithmic choice and autotuning
Autotuning Challenges

- Evaluating quality of candidate algorithms is expensive
  - Must run the program (at least once)
  - More expensive for unfit solutions
  - Scales poorly with larger problem sizes

- Fitness is noisy
  - Randomness from parallel races and system noise
  - Testing each candidate only once often produces a worse algorithm
  - Running many trials is expensive

- Decision tree structures are complex
  - Not easy to hill-climb
  - We artificially bound them
Input Sensitivity

- Input sensitivity is a major challenge
- Different algorithms may be better for different inputs
- Use fast algorithm for easy inputs, slow algorithm for hard inputs
- Avoid pathological cases
Input Sensitivity Today

- Vast majority of programs today use a single algorithm for all inputs
  - This forces design for the “worst case” input
  - Wastes time and resources
Input Sensitivity Today

- Vast majority of programs today use a single algorithm for all inputs
  - This forces design for the “worst case” input
  - Wastes time and resources
- Related work:
  - Uses hand written heuristics to adapt to inputs
  - Rectify inputs for security [Long et al.]
- Our system automatically classifies inputs and runs a program optimized for the type of input being processed
Input Sensitivity Overview

Input Aware Learning

- Program
- Feature Extractors
- Training Inputs

Training Inputs

Insights:
- Feature Priority List
- Performance Bounds

Input Classifier

Selected Program

Input Optimized Programs

Run

Selected

Comments:
Input Features

```cpp
function Sort
to out[n]
from in[n]

input_feature Sortedness, Duplication
{
    ... 
}

function Sortedness
from in[n]
to sortedness
tunable double level (0.0, 1.0)
{
    int sortedcount = 0;
    int count = 0;
    int step = (int)(level*n);
    for(int i=0; i+step<n; i+=step) {
        if(in[i] <= in[i+step]) {
            // increment for correctly ordered
            // pairs of elements
            sortedcount += 1;
        }
    }
    count += 1;
}

if(count > 0)
    sortedness = sortedcount / (double) count;
else
    sortedness = 0.0;

}

function Duplication
from in[n]
to duplication
{
    ... 
}
```
Input Space Sampling

Duplication vs. Sortedness

- PetaBricks
- OpenTuner
- Conclusions
Input Space Sampling

Duplication vs. Sortedness
Input Space Sampling

![Input Space Sampling Diagram](image)
Input Space Sampling

- Duplication
- Sortedness
Input Space Sampling

Duplication vs Sortedness
Training
How Many Landmarks Are Enough?

![Graph showing lost speedup (L) vs. size of region (p_i) for different numbers of configurations.]

![Graph showing speedup vs. number of landmarks for 2 configurations.]

- **Lost speedup (L)** vs. **Size of region (p_i)**
  - 2 configs
  - 3 configs
  - 4 configs
  - 5 configs
  - 6 configs
  - 7 configs
  - 8 configs
  - 9 configs

- **Speedup** vs. **Landmarks**: For 2 configurations.
Input Adaptation Results

- **sort**
- **binpacking**
- **poisson2d**
- **clustering**
- **svd**
- **helmholtz3d**
# Related Projects

A small selection of many related projects:

<table>
<thead>
<tr>
<th>Package</th>
<th>Domain</th>
<th>Search Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Harmony</td>
<td>Runtime System</td>
<td>Nelder-Mead</td>
</tr>
<tr>
<td>ATLAS</td>
<td>Dense Linear Algebra</td>
<td>Exhaustive</td>
</tr>
<tr>
<td>Code Perforation</td>
<td>Compiler</td>
<td>Exhaustive + Simulated Annealing</td>
</tr>
<tr>
<td>Dynamic Knobs</td>
<td>Runtime System</td>
<td>Control Theory</td>
</tr>
<tr>
<td>FFTW</td>
<td>Fast Fourier Transform</td>
<td>Exhaustive / Dynamic Prog.</td>
</tr>
<tr>
<td>Insieme</td>
<td>Compiler</td>
<td>Differential Evolution</td>
</tr>
<tr>
<td>Milepost GCC / cTuning</td>
<td>Compiler</td>
<td>IID Model + Central DB</td>
</tr>
<tr>
<td>OSKI</td>
<td>Sparse Linear Algebra</td>
<td>Exhaustive + Heuristic</td>
</tr>
<tr>
<td>PATUS</td>
<td>Stencil Computations</td>
<td>Nelder-Mead or Evolutionary</td>
</tr>
<tr>
<td>SEEC / Heartbeats</td>
<td>Runtime System</td>
<td>Control Theory</td>
</tr>
<tr>
<td>Sepya</td>
<td>Stencil Computations</td>
<td>Random-Restart Gradient Ascent</td>
</tr>
<tr>
<td>SPIRAL</td>
<td>DSP Algorithms</td>
<td>Pareto Active Learning</td>
</tr>
</tbody>
</table>
Related Projects

A small selection of many related projects:

<table>
<thead>
<tr>
<th>Package</th>
<th>Domain</th>
<th>Search Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Harmony</td>
<td>Runtime System</td>
<td>Nelder-Mead</td>
</tr>
<tr>
<td>ATLAS</td>
<td>Dense Linear Algebra</td>
<td>Exhaustive</td>
</tr>
<tr>
<td>Code Perforation</td>
<td>Compiler</td>
<td>Exhaustive + Simulated Annealing</td>
</tr>
<tr>
<td>Dynamic Knobs</td>
<td>Runtime System</td>
<td>Control Theory</td>
</tr>
<tr>
<td>FFTW</td>
<td>Fast Fourier Transform</td>
<td>Exhaustive / Dynamic Prog.</td>
</tr>
<tr>
<td>Insieme</td>
<td>Compiler</td>
<td>Differential Evolution</td>
</tr>
<tr>
<td>Milepost GCC / cTuning</td>
<td>Compiler</td>
<td>IID Model + Central DB</td>
</tr>
<tr>
<td>OSKI</td>
<td>Sparse Linear Algebra</td>
<td>Exhaustive + Heuristic</td>
</tr>
<tr>
<td>PATUS</td>
<td>Stencil Computations</td>
<td>Nelder-Mead or Evolutionary</td>
</tr>
<tr>
<td>SEEC / Heartbeats</td>
<td>Runtime System</td>
<td>Control Theory</td>
</tr>
<tr>
<td>Sepya</td>
<td>Stencil Computations</td>
<td>Random-Restart Gradient Ascent</td>
</tr>
<tr>
<td>SPIRAL</td>
<td>DSP Algorithms</td>
<td>Pareto Active Learning</td>
</tr>
</tbody>
</table>

- Simple techniques (exhaustive, hill climbers, etc) are popular
- No single technique is best for all problems
- Representations are often just integers/floats/booleans
Limits of Existing Autotuning Projects

- We believe these factors limit the scope and efficiency of autotuning.
- A hill climber works great for a block size, but completely fails at synthesizing poly-algorithms.
- Many users of autotuning work hard to prune their search spaces to fit their techniques.
Limits of Existing Autotuning Projects

- We believe these factors limit the scope and efficiency of autotuning
- A hill climber works great for a block size, but completely fails at synthesizing poly-algorithms
- Many users of autotuning work hard to prune their search spaces to fit their techniques
- OpenTuner provides extensible representations and ensembles of techniques which can solve more complex autotuning problems
OpenTuner Overview

OpenTuner: an extensible framework for program autotuning

Results Database

Search

Search Driver

Search Techniques

Configuration Manipulator

Measurement

Measurement Driver

User Defined Measurement Function

Reads: Results

Writes: Desired Results

Reads: Desired Results

Writes: Results
OpenTuner Configuration Manipulator Parameters

- Hierarchical structure of parameters, user defined parameter types can be added at any point
- Primitive parameters behave like bounded integers or floats
- Complex parameters have a set of stochastic mutation operators
- Technique-specific operators
Ensembles of Techniques

- Differential Evolution
- Particle Swarm Optimization
- Torczon Hill Climber
Ensembles of Techniques

Information sharing through ResultsDB

- Differential Evolution
- Particle Swarm Optimization
- Torczon Hill Climber
Ensembles of Techniques

Information sharing through ResultsDB

Differential Evolution

Particle Swarm Optimization

Torczon Hill Climber

AUC Bandit
Ensembles of Techniques

Information sharing through ResultsDB

Differential Evolution

Particle Swarm Optimization

Torczon Hill Climber

AUC Bandit

Which configuration should we try next?
Ensembles of Techniques

Information sharing through ResultsDB

- Differential Evolution (33%)
- Particle Swarm Optimization (33%)
- Torczon Hill Climber (33%)

Exploration

AUC Bandit

Which configuration should we try next?
Ensembles of Techniques

Information sharing through ResultsDB

Differential Evolution
Particle Swarm Optimization
Torczon Hill Climber

AUC Bandit

100%
0%
0%

Which configuration should we try next?
## OpenTuner Results

<table>
<thead>
<tr>
<th>Project</th>
<th>Benchmark</th>
<th>Possible Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC/G++ Flags</td>
<td>all</td>
<td>$10^{806}$</td>
</tr>
<tr>
<td>Halide</td>
<td>Blur</td>
<td>$10^{52}$</td>
</tr>
<tr>
<td>Halide</td>
<td>Wavelet</td>
<td>$10^{44}$</td>
</tr>
<tr>
<td>HPL</td>
<td>n/a</td>
<td>$10^{9.9}$</td>
</tr>
<tr>
<td>PetaBricks</td>
<td>Poisson</td>
<td>$10^{3657}$</td>
</tr>
<tr>
<td>PetaBricks</td>
<td>Sort</td>
<td>$10^{90}$</td>
</tr>
<tr>
<td>PetaBricks</td>
<td>Strassen</td>
<td>$10^{188}$</td>
</tr>
<tr>
<td>PetaBricks</td>
<td>TriSolve</td>
<td>$10^{1559}$</td>
</tr>
<tr>
<td>Stencil</td>
<td>all</td>
<td>$10^{6.5}$</td>
</tr>
<tr>
<td>Unitary</td>
<td>n/a</td>
<td>$10^{21}$</td>
</tr>
</tbody>
</table>
## OpenTuner Results

<table>
<thead>
<tr>
<th>Project</th>
<th>Benchmark</th>
<th>Possible Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC/G++ Flags</td>
<td>all</td>
<td>$10^{806}$</td>
</tr>
<tr>
<td>Halide</td>
<td>Blur</td>
<td>$10^{52}$</td>
</tr>
<tr>
<td>Halide</td>
<td>Wavelet</td>
<td>$10^{44}$</td>
</tr>
<tr>
<td>HPL</td>
<td>n/a</td>
<td>$10^{9.9}$</td>
</tr>
<tr>
<td>PetaBricks</td>
<td>Poisson</td>
<td>$10^{3657}$</td>
</tr>
<tr>
<td>PetaBricks</td>
<td>Sort</td>
<td>$10^{90}$</td>
</tr>
<tr>
<td>PetaBricks</td>
<td>Strassen</td>
<td>$10^{188}$</td>
</tr>
<tr>
<td>PetaBricks</td>
<td>TriSolve</td>
<td>$10^{1559}$</td>
</tr>
<tr>
<td>Stencil</td>
<td>all</td>
<td>$10^{6.5}$</td>
</tr>
<tr>
<td>Unitary</td>
<td>n/a</td>
<td>$10^{21}$</td>
</tr>
</tbody>
</table>
OpenTuner Results: GCC Flags

fft.c

raytracer.cpp

matrixmultiply.cpp

tsp_ga.cpp
**OpenTuner Results: PetaBricks**

**Poisson 2D**

![Poisson 2D Graph]

**Strassen**

![Strassen Graph]

**Sort**

![Sort Graph]

**Tridiagonal Solver**

![Tridiagonal Solver Graph]
Conclusions

- PetaBricks has pushed the limits of what can be done with algorithmic choice
  - Provides performance portability by allowing programs to adapt to their environment
  - Have shown: variable accuracy, multigrid, and input sensitivity
  - Hope that future main stream programming languages will incorporate algorithmic choice and autotuning
- OpenTuner can expand the scope of program autotuning for other projects
  - Extensible configuration representation
  - Ensembles of techniques
  - Hope that field of autotuning will expand to much more complex problems
Coauthors and Collaborators

- Saman Amarasinghe
- Cy Chan
- Yufei Ding
- Alan Edelman
- Sam Fingeret
- Sanath Jayasena
- Shoaib Kamil
- Kevin Kelley
- Erika Lee
- Deepak Narayanan
- Marek Olszewski
- Una-May O’Reilly
- Maciej Pacula
- Phitchaya Mangpo Phothilimthana
- Jonathan Ragan-Kelley
- Xipeng Shen
- Michele Tartara
- Kalyan Veeramachaneni
- Yod Watanaprakornku
- Yee Lok Wong
- Kevin Wu
- Minshu Zhan
- Qin Zhao
Thanks!

About me:
http://jasonansel.com/

http://opentuner.org/

http://projects.csail.mit.edu/petabricks/