PetaBricks
A language introduction

Jason Ansel

MIT - CSAIL

December 15, 2010
Outline

1 Motivating Example
2 Language features
3 Offline evolutionary algorithm
4 SiblingRivalry: Online evolutionary algorithm
5 Conclusions
Algorithmic choice

Mergesort (N-way)
Algorithmic choice

Mergesort (N-way)  Insertionsort
Algorithmic choice

- Mergesort (N-way)
- Insertionsort
- Radixsort
Algorithmic choice

Mergesort (N-way)

Insertionsort

Radixsort

Quicksort
Algorithmic choice

Mergesort (N-way)

N=2

@15

Insertionsort

Radixsort

Quicksort

STL Algorithm
Algorithmic choice

Mergesort (N-way)

Insertionsort

Optimized For:
Xeon (1 core)

Radixsort

Quicksort
Algorithmic choice

Optimized For:
- Xeon (1 core)
- Xeon (8 cores)

Mergesort (N-way)
- N=4
- N=2

Insertionsort
- @75

Radixsort
- @98

Quicksort
- @1420
- @600
Algorithmic choice

Optimized For:
- Xeon (1 core)
- Xeon (8 cores)
- Niagara (8 cores)

Mergesort (N-way) 
- N=2, 4, 8, 16 
- @75
- @1461
- @2400

Insertion sort 
- N=4
- @75

Radixsort 
- @98
- @1420

Quicksort 
- @600
Algorithmic choice

Mergesort (N-way)

N=2,4,8

N=2,4,8,16

Insertionsort

N=2

N=4

Radixsort

Quick sort

Optimized For:
- Xeon (1 core)
- Xeon (8 cores)
- Niagara (8 cores)
- Core 2 (2 cores)
Algorithmic choice

Optimized For:
- Xeon (1 core)
- Xeon (8 cores)
- Niagara (8 cores)
- Core 2 (2 cores)
The PetaBricks language

- Choices expressed in the language
  - High level algorithmic choices
  - Low level ordering choices
  - Parallelization strategy
  - Quality of service trade-offs

- Programs automatically adapt to their environment
  - Tuned using our bottom-up evaluation algorithm
  - Offline autotuner, or
  - Always-on online autotuner
1 Motivating Example

2 Language features

3 Offline evolutionary algorithm

4 SiblingRivalry: Online evolutionary algorithm

5 Conclusions
Algorithmic choices

Language

either
{
    InsertionSort(out, in);
}
or
{
    QuickSort(out, in);
}
or
{
    MergeSort(out, in);
}
or
{
    RadixSort(out, in);
}
Algorithmic choices

Language

either
  { InsertionSort(out, in); }
  or
  { QuickSort(out, in); }
  or
  { MergeSort(out, in); }
  or
  { RadixSort(out, in); }

⇒

Representation

Decision tree synthesized by our evolutionary algorithm (EA)
Language

```plaintext
transform Add
from A[n], B[n]
to AB[n]
{
  from (A. cell(i) a, B. cell(i) b)
to (AB. cell(i) ab) {
    ab = a + b;
  }
}
```
Iteration order choices (part 1)

Language

```
transform Add
from A[n], B[n]
to AB[n]
{
    from (A.cell(i) a, B.cell(i) b)
to (AB.cell(i) ab) {
    ab = a + b;
}
```

Representation

⇒ Algorithmic choices over parallel/sequential blocking strategies
Language

```plaintext
transform PrefixSum
from A[n]
to AB[n]
{
  from (A.cell(i) a, AB.cell(i-1) left)
to (AB.cell(i) ab) {
    ab = a + left;
  }
}
```
Language

```plaintext
transform PrefixSum
from A[n]
to AB[n]
{
    from (A.cell(i) a,
        AB.cell(i - 1) left)
to (AB.cell(i) ab) {
        ab = a + left;
    }

    from (A.cell(0) a)
to (AB.cell(0) ab) {
        ab = a;
    }
}
```
Language

\texttt{transform PrefixSum}

\texttt{from A[n] to AB[n]}

\begin{verbatim}
{ from(A.cell(i) a, AB.cell(i-1) left) to(AB.cell(i) ab) {
  ab=a+left;
}

from(A.cell(0) a) to(AB.cell(0) ab) {
  ab=a;
}
\end{verbatim}

⇒

Representation

Single sequential ordering
Combined iteration order and algorithmic choices

Language

```
transform PrefixSum
from A[n]
to AB[n] { 
    from (A.cell(i) a, 
          AB.cell(i-1) left)
    to (AB.cell(i) ab) {
        ab = a + left;
    }

    from (A.cell(0) a)
    to (AB.cell(0) ab) {
        ab = a;
    }
}
```

⇒ Representation
Decision tree synthesized by our EA

Jason Ansel (MIT)
Combined iteration order and algorithmic choices

Language

```plaintext
transform PrefixSum
from A[n]
to AB[n] {
    from (A.cell(i) a, AB.cell(i-1) left)
to (AB.cell(i) ab) {
    ab = a + left;
}

from (A.cell(0) a)
to (AB.cell(0) ab) {
    ab = a;
}

from (A a)
to (AB ab) {
    ParallelPrefixSum(ab, a);
}
```
Combined iteration order and algorithmic choices

Language

\[
\begin{align*}
\text{\texttt{transform} PrefixSum} \\
\text{\texttt{from A[n]} } \\
\text{\texttt{to AB[n]} } \\
\quad \{ \\
\quad \quad \text{\texttt{from (A.cell(i) a,}} \\
\quad \quad \quad \text{\texttt{AB.cell(i-1) left)}} \\
\quad \quad \text{\texttt{to (AB.cell(i) ab)} } \\
\quad \quad \quad \{ \\
\quad \quad \quad \quad \text{\texttt{ab=a+left;}} \\
\quad \quad \} \\
\quad \text{\texttt{from (A.cell(0) a)}} \\
\quad \text{\texttt{to (AB.cell(0) ab) } } \\
\quad \quad \{ \\
\quad \quad \quad \text{\texttt{ab=a;}} \\
\quad \quad \} \\
\quad \text{\texttt{from (A a)}} \\
\quad \text{\texttt{to (AB ab)} } \\
\quad \quad \{ \\
\quad \quad \quad \text{\texttt{ParallelPrefixSum(ab, a);}} \\
\quad \quad \}
\end{align*}
\]

⇒

Representation

Decision tree synthesized by our EA
Spawn/sync parallelism

Language

```cpp
spawn Sort(tmp.region(0, n/2));
spawn Sort(tmp.region(n/2, n));
sync;

Merge(out,
    tmp.region(0, n/2),
    tmp.region(n/2, n));
```
Spawn/sync parallelism

Language

```c
spawn Sort(tmp.region(0, n/2));
spawn Sort(tmp.region(n/2, n));
sync;

Merge(out,
    tmp.region(0, n/2),
    tmp.region(n/2, n));
```

Representation

Choice of which input sizes to run parallel and which to run sequential.
Variable accuracy (quality of service) choices

Language

\texttt{accuracy\_metric} MyRMSErr
Variable accuracy (quality of service) choices

Language

```plaintext
accuracy_metric MyRMSEError

... for_enough {
    SORIteration(tmp);
}
```
Variable accuracy (quality of service) choices

Language

```
accuracy_metric MyRMSError

for_enough {
    SORIteration(tmp);
}
```

Representation

Function from problem size to number of iterations synthesized by our EA
User parameters

Language

**tunable** $N$

... 

`MergeSortNWay(out, in, N);`
User parameters

Language

\texttt{tunable \ N}

\ldots

\texttt{MergeSortNWay(out, in, N)};

⇒

Representation

A single value chosen by our EA
User parameters

Language

\texttt{tunable \ N}

\ ...

\texttt{MergeSortNWay(out, in, N)};

\Rightarrow

Representation

A single value chosen by our EA

Language

\texttt{tunable\_array \ N}

\ ...

\texttt{MergeSortNWay(out, in, N)};

\Rightarrow

Representation

Function from input size to a value synthesized by our EA
Other choices

- Work first vs. scheduling first work stealing algorithm
- Lazy vs. aggressive dependency resolution
- Not yet explored:
  - “low level” choices
  - Compiler parameters / pragmas
  - Loop unrolling / inlining / prefetching
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Variable accuracy</th>
<th>Search space dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin Packing</td>
<td>Yes</td>
<td>117</td>
</tr>
<tr>
<td>Clustering</td>
<td>Yes</td>
<td>91</td>
</tr>
<tr>
<td>Eigenproblem</td>
<td>No</td>
<td>35</td>
</tr>
<tr>
<td>Helmholtz</td>
<td>Yes</td>
<td>61</td>
</tr>
<tr>
<td>Image Compression</td>
<td>Yes</td>
<td>163</td>
</tr>
<tr>
<td>LU Factorization</td>
<td>No</td>
<td>140</td>
</tr>
<tr>
<td>Matrix Multiply</td>
<td>No</td>
<td>108</td>
</tr>
<tr>
<td>Poisson</td>
<td>Yes</td>
<td>64</td>
</tr>
<tr>
<td>Preconditioner</td>
<td>Yes</td>
<td>159</td>
</tr>
<tr>
<td>Sort</td>
<td>No</td>
<td>33</td>
</tr>
<tr>
<td>Average</td>
<td>-</td>
<td>97.1</td>
</tr>
</tbody>
</table>
Outline

1. Motivating Example
2. Language features
3. Offline evolutionary algorithm
4. SiblingRivalry: Online evolutionary algorithm
5. Conclusions
Offline autotuner

- Evolutionary algorithm
- Smart mutation operators created by compiler analysis
- “Bottom-up”
  - Uses smaller input performance to form initial population for larger inputs
- Adaptive number of trials
  - Based on statistical hypothesis testing
- Multi-objective
Problems with offline learning

- Offline-tuning workflow burdensome
  - Programs often not re-autotuned when they should be
    - (e.g. `apt-get install fftw` does not re-autotune)
  - Hardware upgrades / large deployments
  - Transparent migration in the cloud

- Can’t adapt to dynamic phenomenas
  - System load
  - Input types
Effect of architecture on autotuning

Train Remotely

Offline Training
Development machine (N cores)

Deploy tuned application

Online Execution
Production machine (M cores)

Train Natively

Offline Training
Production machine (M cores)
## Effect of architecture on autotuning

<table>
<thead>
<tr>
<th>Run on</th>
<th>Mobile</th>
<th>Xeon1</th>
<th>Xeon8</th>
<th>Niagara</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile</td>
<td>-</td>
<td>1.09x</td>
<td>1.67x</td>
<td>1.47x</td>
</tr>
<tr>
<td>Xeon1</td>
<td>1.61x</td>
<td>-</td>
<td>2.08x</td>
<td>2.50x</td>
</tr>
<tr>
<td>Xeon8</td>
<td>1.59x</td>
<td>2.14x</td>
<td>-</td>
<td>2.35x</td>
</tr>
<tr>
<td>Niagara</td>
<td>1.12x</td>
<td>1.51x</td>
<td>1.08x</td>
<td>-</td>
</tr>
</tbody>
</table>
Challenges for online learning

- Search space is difficult to model
  - High-dimensional
  - Non-linear
  - Irregular
  - Complex dependencies

- Dangerous configurations exist
  - Exponential algorithms
  - Infinite loops
  - Poor quality of service
SiblingRivalry (online autotuner)

Processor
SiblingRivalry (online autotuner)

Processor

Safe Configuration

Experimental Configuration
SiblingRivalry (online autotuner)

- Split available resources in half
- Process identical requests on both halves
- Race two candidate configurations (safe and experimental) and terminate slower algorithm
- Initial slowdown (from duplicating the request) can be overcome by autotuner
- Surprisingly, reduces average power consumption per request
Learning technique

- Maintain population of candidate algorithms
- Each candidate must be pareto-optimal in 3D objective space:
  - Performance
  - Quality of service
  - Confidence
- Pick safe and experimental configurations from population
- Mutate the experimental configuration
- Add the new configuration to the population if it wins the race
Adaptive mutator selection

- Extension of bandit-based differential evolution [DaCosta et al.]
- Deterministically chooses mutation operators
- Requires only relative performance information
- Considers trade-off between exploitation and exploration

$$\text{arg max}_i \left( AUC_i + C \sqrt{\frac{2 \log \sum_k n_k}{n_i}} \right)$$
Experimental setup

**Offline Training**
Development machine (N cores)

**Baseline**
No Change
Production machine (M cores)
Run using M threads

**Sibling Rivalry**
Online Training
Production machine (M cores)
Race M/2 threads vs M/2 threads
Sibling Rivalry: throughput

The diagram shows normalized throughput for various tasks. The tasks include Bin Packing, Clustering, Helmholtz, Image Compression, LU Factorization, Matrix Multiply, Poisson, Sort, and GeoMean.

The results for offline: Xeon 8, online: AMD 48 are represented by blue bars, while offline: AMD 48, online: Xeon 32 are represented by black bars.

Tasks and their normalized throughput:

- Bin Packing: 23.5x
- Clustering: 6.9x
- Helmholtz: 9.7x
- Image Compression
- LU Factorization
- Matrix Multiply
- Poisson
- Sort
- GeoMean

The GeoMean value is not explicitly shown but can be calculated from the overall normalized throughput.
SiblingRivalry: energy usage (on AMD48)

![Graph showing energy usage for various benchmarks]

- Benchmark: Bin Packing, Clustering, Helmholtz, Image Compression, LU Factorization, Matrix Multiply, Poisson, Sort, Mean
- Energy per request (joules)
- Baseline vs SiblingRivalry comparison
- Jason Ansel (MIT)

December 15, 2010
Outline

1. Motivating Example
2. Language features
3. Offline evolutionary algorithm
4. SiblingRivalry: Online evolutionary algorithm
5. Conclusions
Current and future work

- **Publications**
  - PetaBricks: A Language and Compiler for Algorithmic Choice. [PLDI’09]
  - Autotuning Multigrid with PetaBricks. [SC’09]
  - PetaBricks: Building adaptable and more efficient programs for the multi-core era. [XRDS Vol.17]
  - Language and Compiler Support for Auto-Tuning Variable-Accuracy Algorithms.[CGO’11]

- **Submitted papers**
  - SiblingRivalry: Online Autotuning Through Local Competitions
  - An Efficient Evolutionary Algorithm for Solving Bottom Up Problems

- **Current projects**
  - Cluster/cloud back-end
  - Heterogeneous systems
  - Applications in wind-energy prediction and graphics
Backup slides
## Test systems

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Processor Type</th>
<th>Processors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mobile</strong></td>
<td>Core 2 Duo Mobile 1.6 GHz</td>
<td>1 (×2 cores)</td>
</tr>
<tr>
<td><strong>Niagara</strong></td>
<td>Sun Fire T200 Niagara 1.2 GHz</td>
<td>1 (×8 cores)</td>
</tr>
<tr>
<td><strong>Xeon1</strong></td>
<td>Intel Xeon X5460 3.16GHz</td>
<td>1 (other cores disabled)</td>
</tr>
<tr>
<td><strong>Xeon8</strong></td>
<td>Intel Xeon X5460 3.16GHz</td>
<td>2 (×4 cores)</td>
</tr>
<tr>
<td><strong>Xeon32</strong></td>
<td>Intel Xeon X7560 2.27GHz</td>
<td>4 (×8 cores)</td>
</tr>
<tr>
<td><strong>AMD48</strong></td>
<td>AMD Opteron 6168 1.9GHz</td>
<td>4 (×12 cores)</td>
</tr>
</tbody>
</table>
### Large choice space

<table>
<thead>
<tr>
<th>Benchmark name</th>
<th>Variable accuracy</th>
<th>Decision trees</th>
<th>Synthesized functions</th>
<th>Tunables</th>
<th>Search space dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin Packing</td>
<td>Yes</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>117</td>
</tr>
<tr>
<td>Clustering</td>
<td>Yes</td>
<td>1</td>
<td>2</td>
<td>10</td>
<td>91</td>
</tr>
<tr>
<td>Eigenproblem</td>
<td>No</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>35</td>
</tr>
<tr>
<td>Helmholtz</td>
<td>Yes</td>
<td>1</td>
<td>1</td>
<td>15</td>
<td>61</td>
</tr>
<tr>
<td>Image Compression</td>
<td>Yes</td>
<td>2</td>
<td>1</td>
<td>9</td>
<td>163</td>
</tr>
<tr>
<td>LU Factorization</td>
<td>No</td>
<td>4</td>
<td>0</td>
<td>13</td>
<td>140</td>
</tr>
<tr>
<td>Matrix Multiply</td>
<td>No</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>108</td>
</tr>
<tr>
<td>Poisson</td>
<td>Yes</td>
<td>1</td>
<td>1</td>
<td>21</td>
<td>64</td>
</tr>
<tr>
<td>Preconditioner</td>
<td>Yes</td>
<td>2</td>
<td>1</td>
<td>22</td>
<td>159</td>
</tr>
<tr>
<td>Sort</td>
<td>No</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>33</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>-</td>
<td><strong>1.8</strong></td>
<td><strong>0.6</strong></td>
<td><strong>10.8</strong></td>
<td><strong>97.1</strong></td>
</tr>
</tbody>
</table>
Sort timings (fixed accuracy)

InsertionSort
QuickSort
MergeSort
RadixSort
Autotuned

Time (s)
Input Size
2D Poisson (variable accuracy)

Accuracy Level $10^9$
Accuracy Level $10^7$
Accuracy Level $10^5$
Accuracy Level $10^3$
Accuracy Level $10^1$

Speedup (x)
Input Size
SiblingRivalry: convergence (Sort on AMD48)

Request per second vs. Time (s)

- SiblingRivalry (w/o offline)
- SiblingRivalry (w/ offline on Xeon8)
- SiblingRivalry (w/ offline on AMD48)