PetaBricks

A language introduction

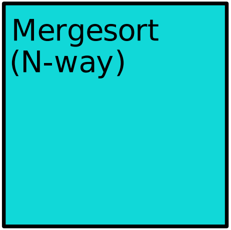
Jason Ansel

MIT - CSAIL

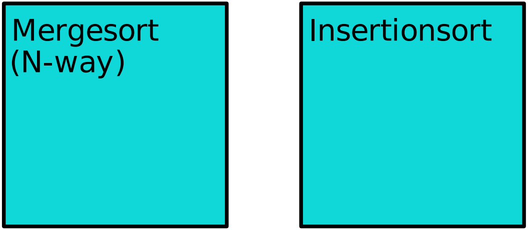
December 15, 2010

Outline

Algorithmic choice



Algorithmic choice



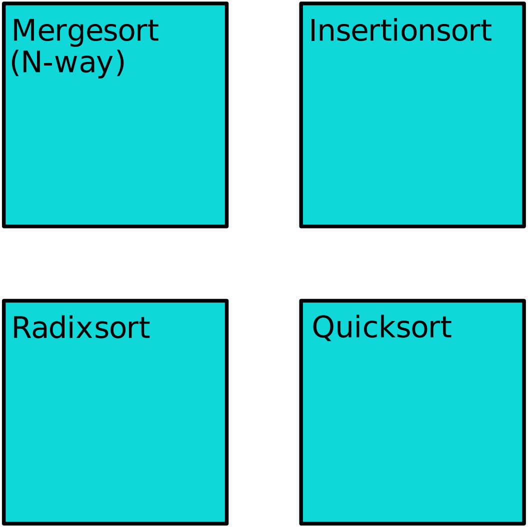
Algorithmic choice

Mergesort (N-way)

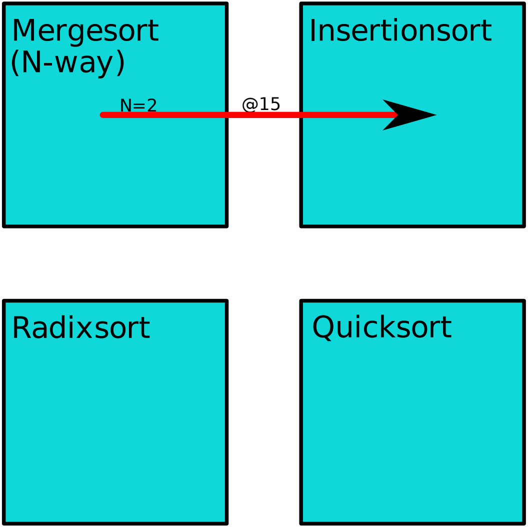
Insertionsort

Radixsort

Algorithmic choice

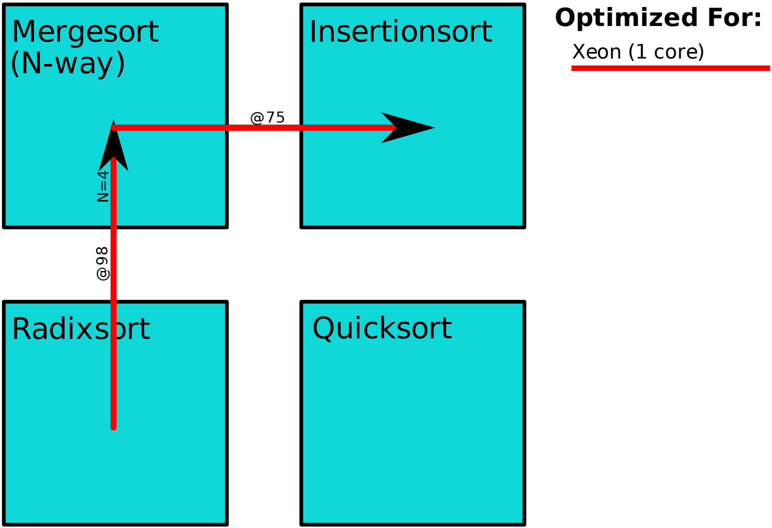


Algorithmic choice

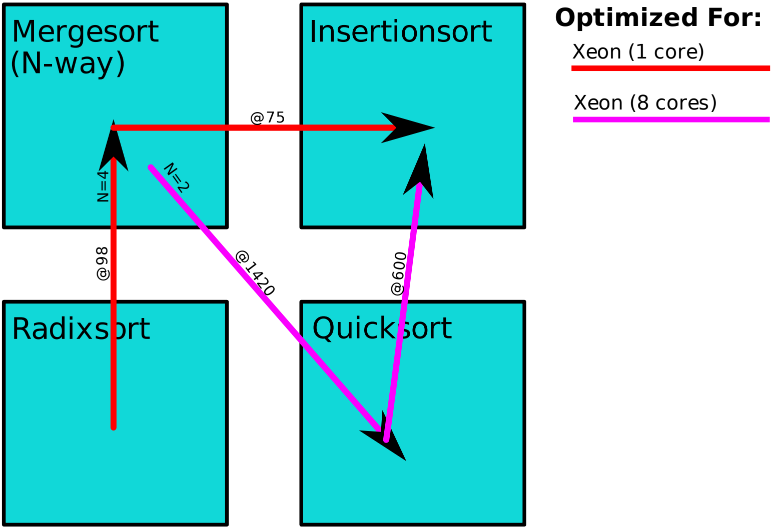


STL Algorithm

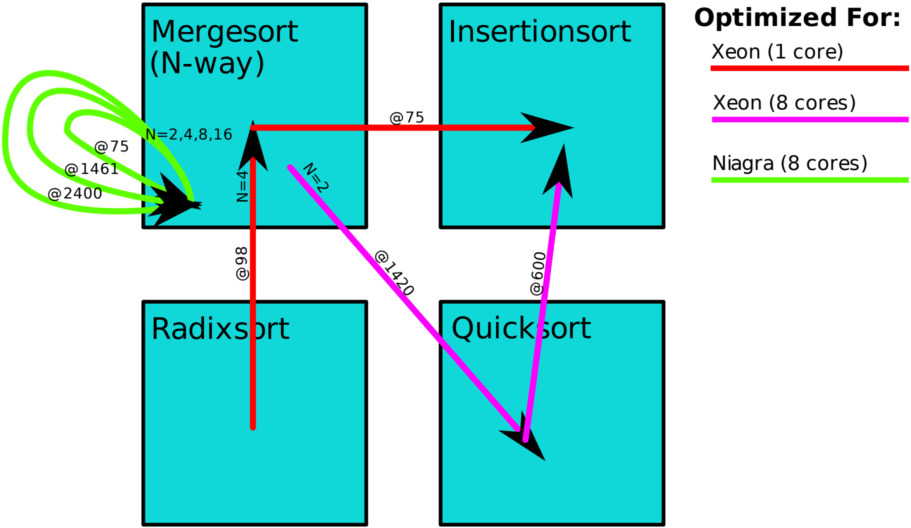
Algorithmic choice



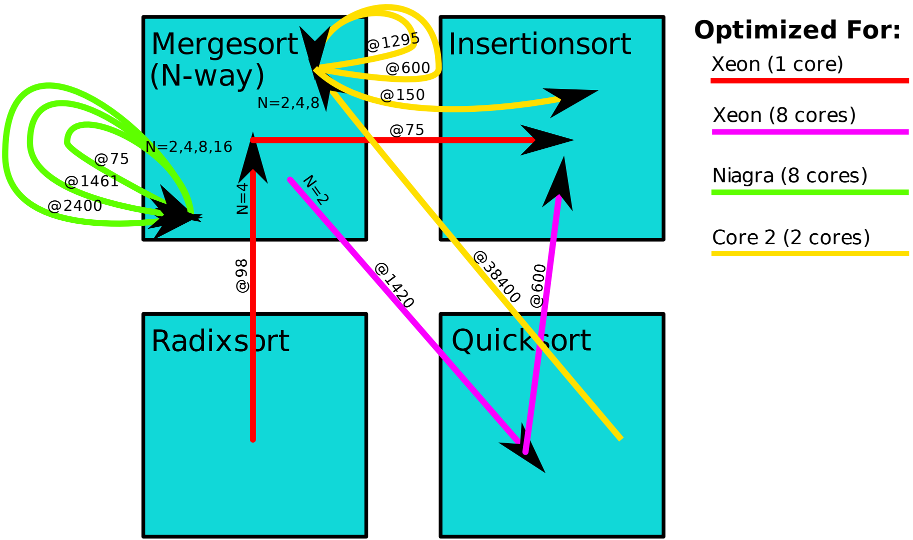
Algorithmic choice



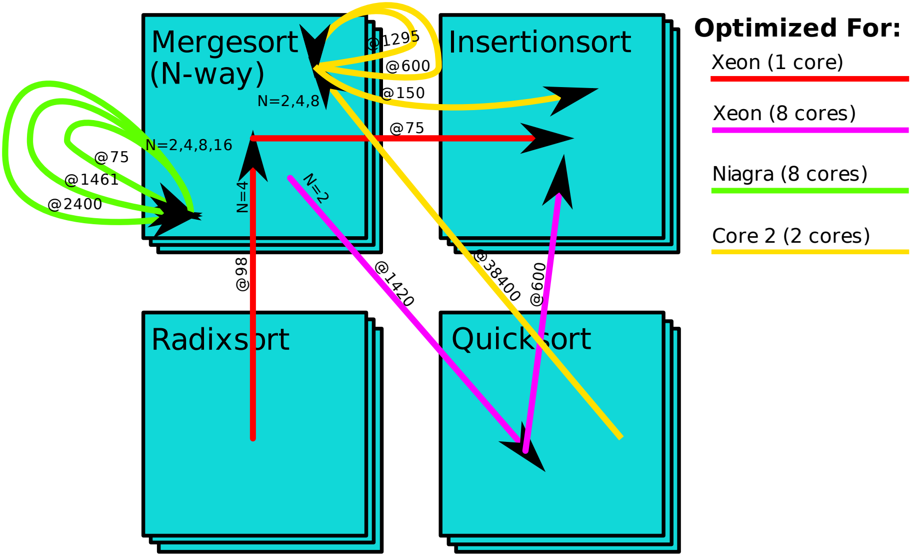
Algorithmic choice



Algorithmic choice



Algorithmic choice



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

Outline

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)

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;
  }
}
```

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

Iteration order choices (part 2)

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;
  }
```

Iteration order choices (part 2)

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;
  }
}
```

Iteration order choices (part 2)

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

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;
  }
```

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;
  }

  from(A a)
  to(AB ab) {
    ParallelPrefixSum(ab, a);
  }
}
```

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;
  }

  from(A a)
  to(AB ab) {
    ParallelPrefixSum(ab, a);
  }
}
```

⇒

Representation

Decision tree synthesized by our EA

Spawn/sync parallelism

Language

```
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

```
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));
```

$\Rightarrow$

Representation

Choice of which input sizes to run parallel and which to run sequential.

Variable accuracy (quality of service) choices

Language

accuracy_metric MyRMSError

Variable accuracy (quality of service) choices

Language

```
accuracy_metric MyRMSError
...
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

```
tunable N
...
MergeSortNWay(out, in, N);
```

⇒

Representation

A single value chosen by our EA

User parameters

Language

```
tunable N
...
MergeSortNWay(out, in, N);
```

⇒

Representation

A single value chosen by our EA

Language

```
tunable_array N
...
MergeSortNWay(out, in, N);
```

⇒

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

Large choice space

Benchmark	Variable accuracy	Search space dimensions
Bin Packing	Yes	117
Clustering	Yes	91
Eigenproblem	No	35
Helmholtz	Yes	61
Image Compression	Yes	163
LU Factorization	No	140
Matrix Multiply	No	108
Poisson	Yes	64
Preconditioner	Yes	159
Sort	No	33
Average	-	97.1

Outline

Offline autotuner

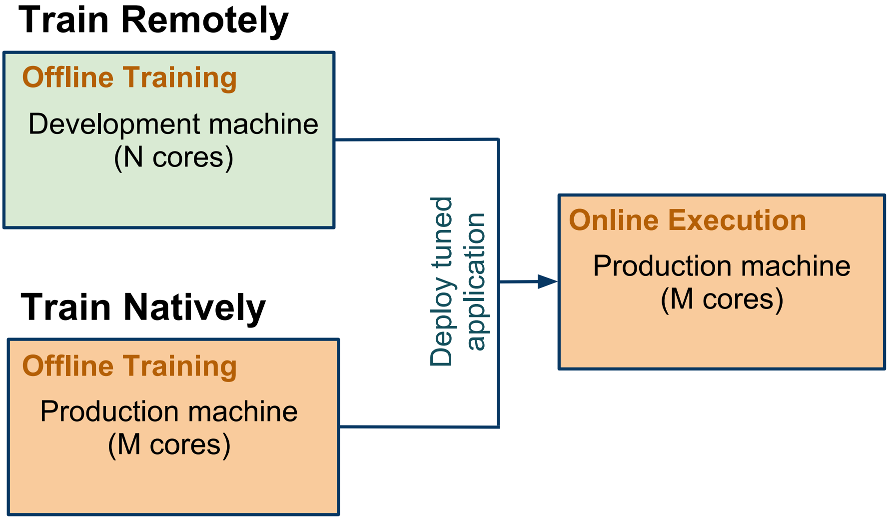
- Evolutionary algorithm
- Smart mutation operators created by compiler analysis
- "Bottom-up"
  - Uses smaller input performance to form initial population for larger inputs
- Adaptive number of trials
  - Based on statistical hypothesis testing
- Multi-objective

Outline

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



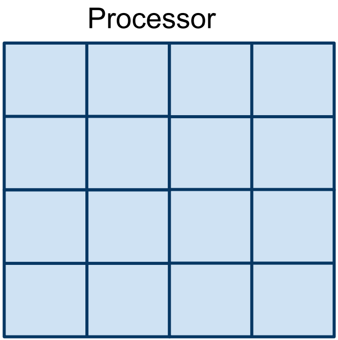
Effect of architecture on autotuning

		Trained on			
		Mobile	Xeon1	Xeon8	Niagara
Run on	Mobile	-	1.09x	1.67x	1.47x
	Xeon1	1.61x	-	2.08x	2.50x
	Xeon8	1.59x	2.14x	-	2.35x
	Niagara	1.12x	1.51x	1.08x	-

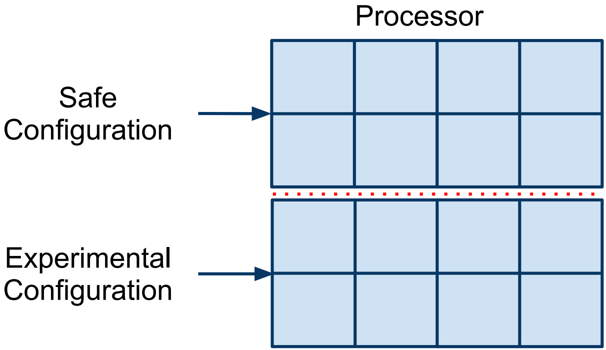
Challenges for online learning

- 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)



SiblingRivalry (online autotuner)



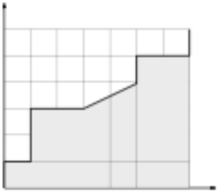
SiblingRivalry (online autotuner)

- Split available resources in half
- Process identical requests on both halfs
- 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

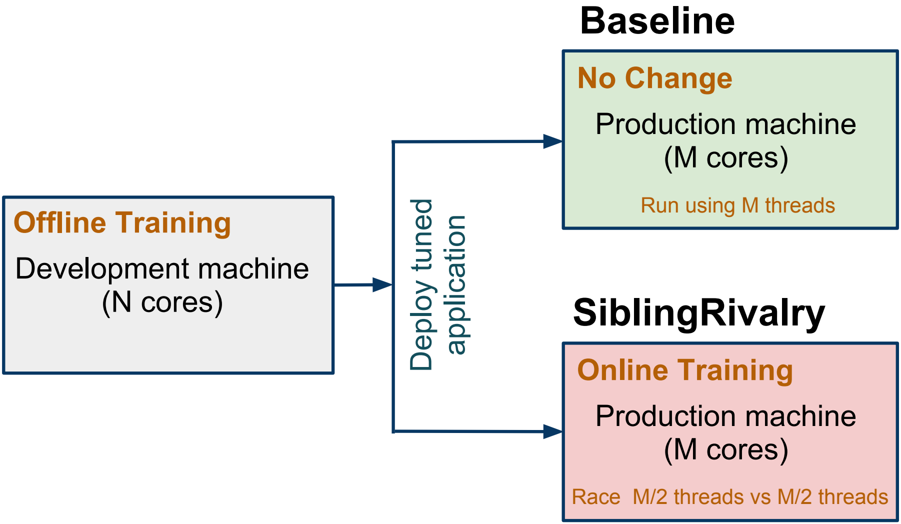
Adaptive mutator selection



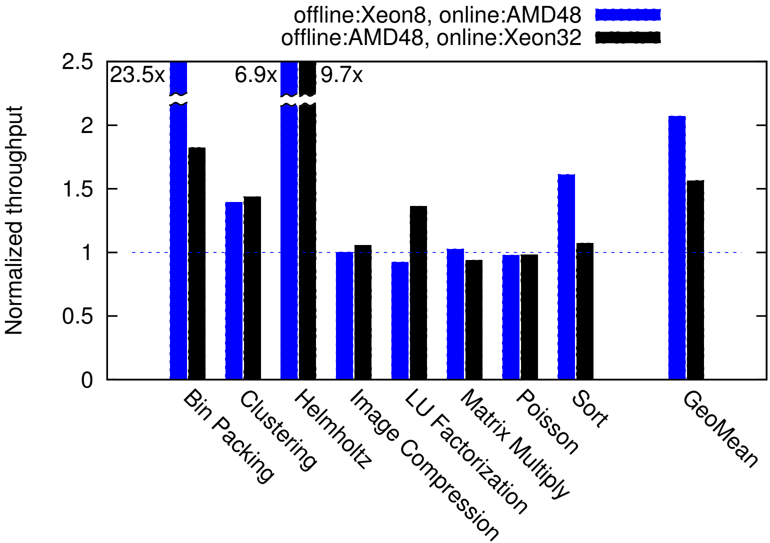
- Extension of bandit-based differential evolution [DaCosta et al.]
- Deterministically choses mutation operators
- Requires only relative performance information
- Considers trade-off between *exploitation* and *exploration*

$$\arg\max_i \left(AUC_i + C\sqrt{\frac{2\log \sum_k n_k}{n_i}} \right)$$

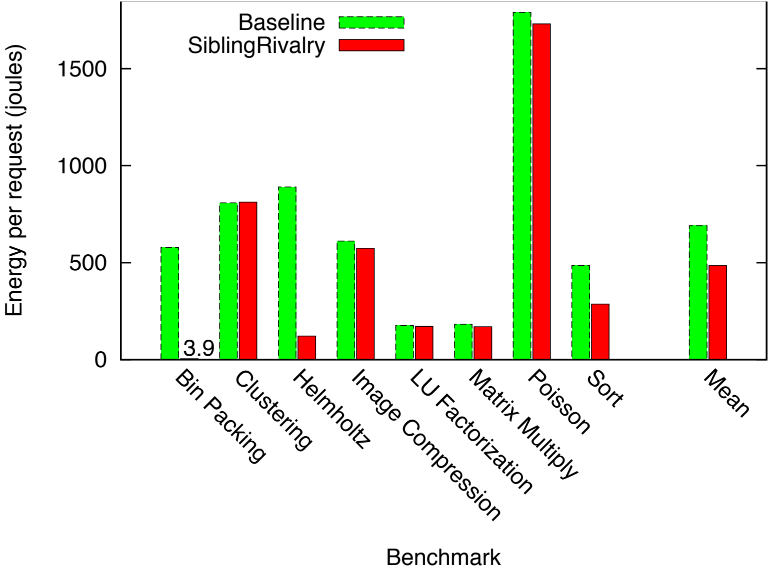
Experimental setup



SiblingRivalry: throughput



SiblingRivalry: energy usage (on AMD48)



Outline

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

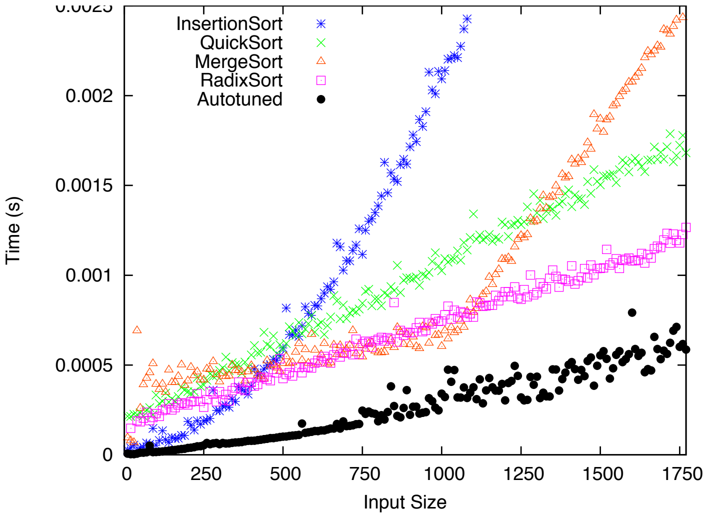
Test systems

Acronym	Processor Type	Processors
Mobile	Core 2 Duo Mobile 1.6 GHz	1 ($\times$2 cores)
Niagara	Sun Fire T200 Niagara 1.2 GHz	1 ($\times$8 cores)
Xeon1	Intel Xeon X5460 3.16GHz	1 (other cores disabled)
Xeon8	Intel Xeon X5460 3.16GHz	2 ($\times$4 cores)
Xeon32	Intel Xeon X7560 2.27GHz	4 ($\times$8 cores)
AMD48	AMD Opteron 6168 1.9GHz	4 ($\times$12 cores)

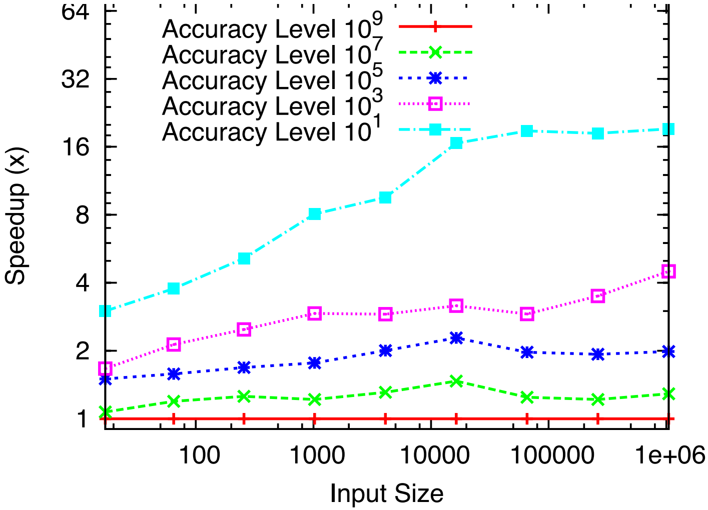
Large choice space

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

Sort timings (fixed accuracy)



2D Poisson (variable accuracy)



SiblingRivalry: convergence (Sort on AMD48)