PetaBricks

A Language and Compiler for Algorithmic Choice

Jason Ansel Cy Chan Yee Lok Wong Marek Olszewski
Qin Zhao Alan Edelman Saman Amarasinghe

MIT - CSAIL

June 16, 2009

Outline

1. Introduction
   - Motivating Example
   - Language & Compiler Overview
   - Why choices
2. PetaBricks Language
   - Key Ideas
   - Compilation Example
   - Other Language Features
3. Results
   - Benchmarks
   - Scalability
   - Variable Accuracy
4. Conclusion
   - Final thoughts

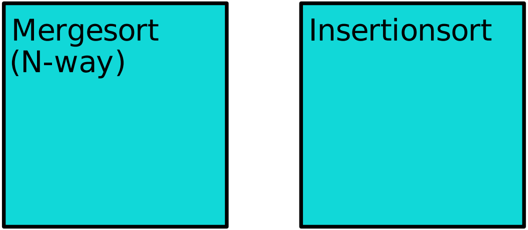
Algorithmic choice

Mergesort
(N-way)

Algorithmic choice

Mergesort
(N-way)

Algorithmic choice



Algorithmic choice

Mergesort (N-way)

Insertionsort

Radixsort

Algorithmic choice

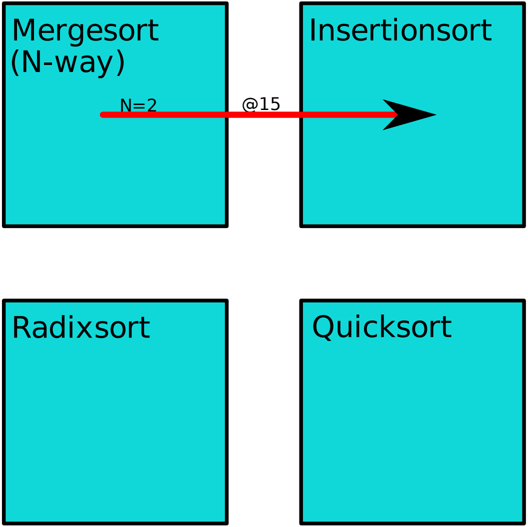
Mergesort (N-way)

Insertionsort

Radixsort

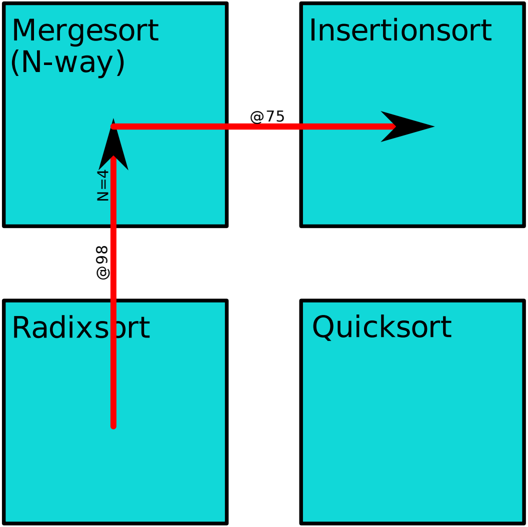
Quicksort

Algorithmic choice



STL Algorithm

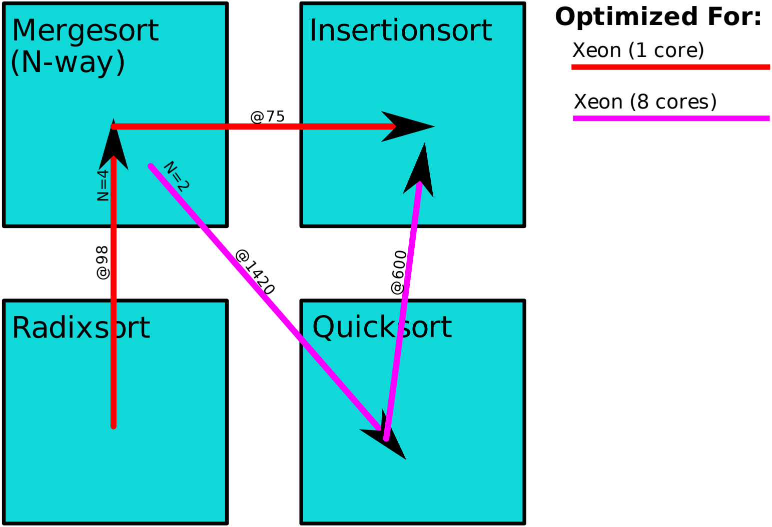
Algorithmic choice



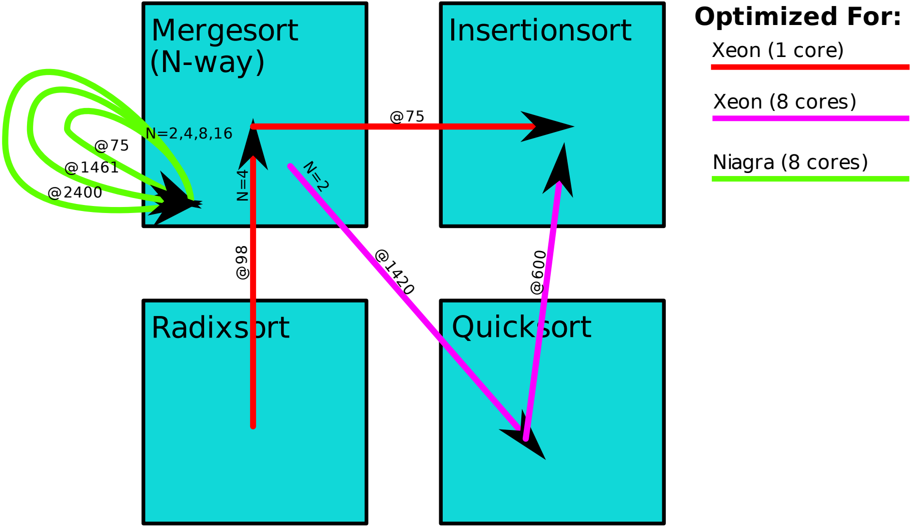
Optimized For:

Xeon (1 core)

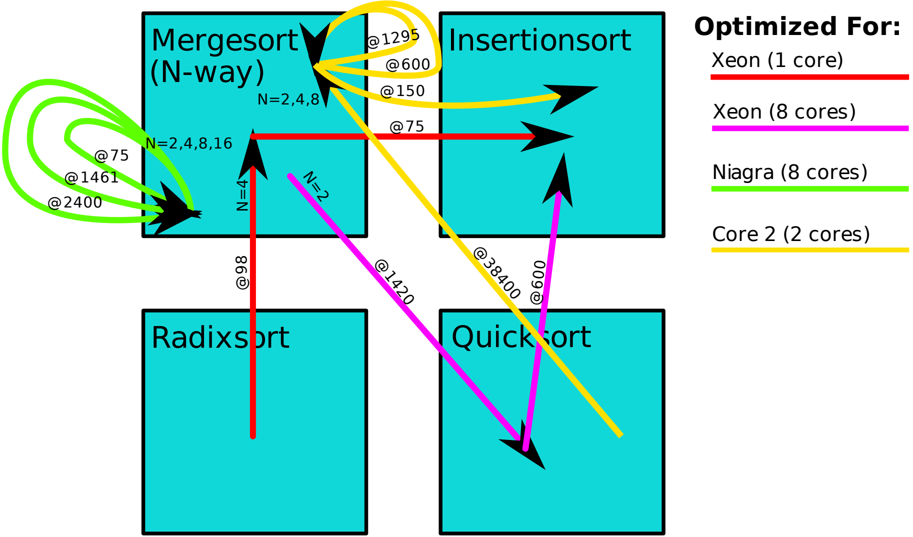
Algorithmic choice



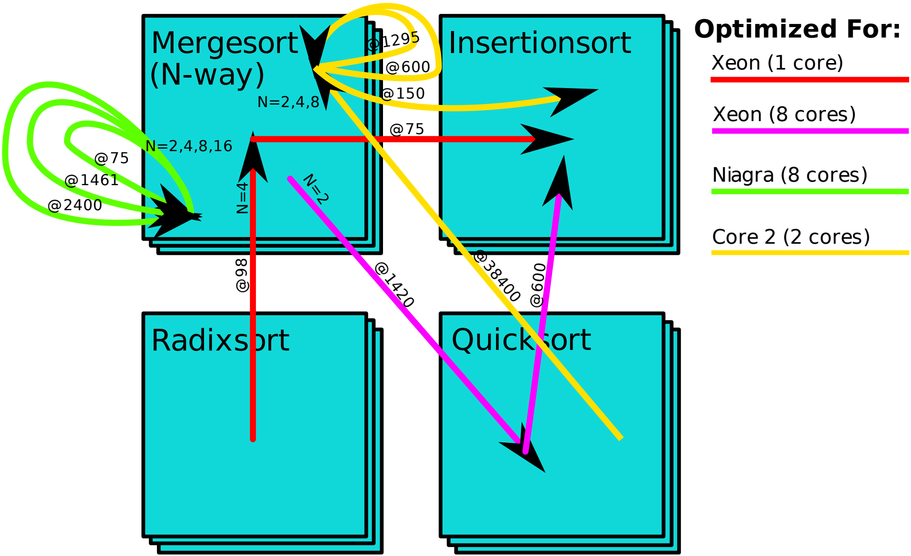
Algorithmic choice



Algorithmic choice



Algorithmic choice



The PetaBricks language

- The case for autotuning is obvious

The PetaBricks language

- The case for autotuning is obvious
- How should the programmer represent choice?

The PetaBricks language

- The case for autotuning is obvious
- How should the programmer represent choice?

- We present the **PetaBricks** programming language and compiler:

The PetaBricks language

- The case for autotuning is obvious
- How should the programmer represent choice?

- We present the **PetaBricks** programming language and compiler:
  - Choice as a fundamental language construct

The PetaBricks language

- The case for autotuning is obvious
- How should the programmer represent choice?

- We present the **PetaBricks** programming language and compiler:
  - Choice as a fundamental language construct
  - Autotuning performed by the compiler

The PetaBricks language

- The case for autotuning is obvious
- How should the programmer represent choice?

- We present the **PetaBricks** programming language and compiler:
    - Choice as a fundamental language construct
    - Autotuning performed by the compiler
    - Automatically parallelized

Sort in PetaBricks

```
transform Sort
from A[n]
to B[n]
{
  from(A a) to(B b) {
    tunable WAYS;
    /* Mergesort */
  } or {
    /* Insertionsort */
  } or {
    /* Radixsort */
  } or {
    /* Quicksort */
  }
}
```

Sort in PetaBricks

```
transform Sort
from A[n]
to B[n]
{
  from(A a) to(B b) {
    tunable WAYS;
    /* Mergesort */
  } or {
    /* Insertionsort */
  } or {
    /* Radixsort */
  } or {
    /* Quicksort */
  }
}
```

Sort in PetaBricks

```
transform Sort
from A[n]
to B[n]
{
  from(A a) to(B b) {
    tunable WAYS;
    /* Mergesort */
  } or {
    /* Insertionsort */
  } or {
    /* Radixsort */
  } or {
    /* Quicksort */
  }
}
```

Sort in PetaBricks

```
transform Sort
from A[n]
to B[n]
{
  from(A a) to(B b) {
    tunable WAYS;
    /* Mergesort */
  } or {
    /* Insertionsort */
  } or {
    /* Radixsort */
  } or {
    /* Quicksort */
  }
}
```

Sort in PetaBricks

```
transform Sort
from A[n]
to B[n]
{
  from(A a) to(B b) {
    tunable WAYS;
    /* Mergesort */
  } or {
    /* Insertionsort */
  } or {
    /* Radixsort */
  } or {
    /* Quicksort */
  }
}
```

Sort in PetaBricks

```
transform Sort
from A[n]
to B[n]
{
  from(A a) to(B b) {
    tunable WAYS;
    /* Mergesort */
  } or {
    /* Insertionsort */
  } or {
    /* Radixsort */
  } or {
    /* Quicksort */
  }
}
```

Sort in PetaBricks

```
transform Sort
from A[n]
to B[n]
{
  from(A a) to(B b) {
    tunable WAYS;
    /* Mergesort */
  } or {
    /* Insertionsort */
  } or {
    /* Radixsort */
  } or {
    /* Quicksort */
  }
}
```

Outline

1. Introduction
    - Motivating Example
    - Language & Compiler Overview
    - Why choices
2. PetaBricks Language
    - Key Ideas
    - Compilation Example
    - Other Language Features
3. Results
    - Benchmarks
    - Scalability
    - Variable Accuracy
4. Conclusion
    - Final thoughts

The PetaBricks compiler

- Sort is compiled into a autotuning binary

The PetaBricks compiler

- Sort is compiled into a autotuning binary
- Trained on target architecture

The PetaBricks compiler

- Sort is compiled into a autotuning binary
- Trained on target architecture
    - Structured genetic tuner

The PetaBricks compiler

- Sort is compiled into a autotuning binary
- Trained on target architecture
    - Structured genetic tuner
    - Trained with full number of threads

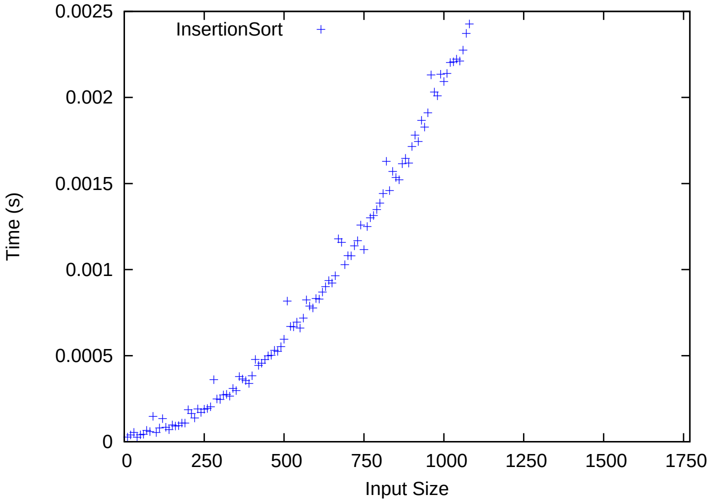
The PetaBricks compiler

- Sort is compiled into a autotuning binary
- Trained on target architecture
    - Structured genetic tuner
    - Trained with full number of threads
    - Under 1 minute for Sort

The PetaBricks compiler

- Sort is compiled into a autotuning binary
- Trained on target architecture
    - Structured genetic tuner
    - Trained with full number of threads
    - Under 1 minute for Sort
- Results fed back into the compiler
- Final binary created

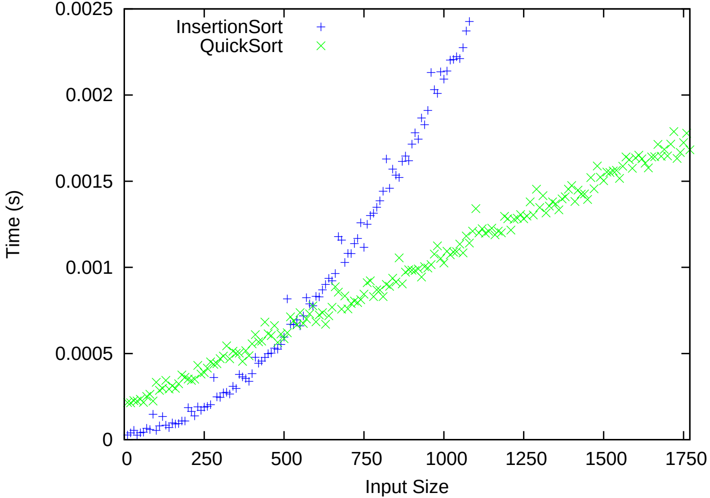
Sort algorithm timings[1]



---

[1] On an 8-way Xeon E7340 system

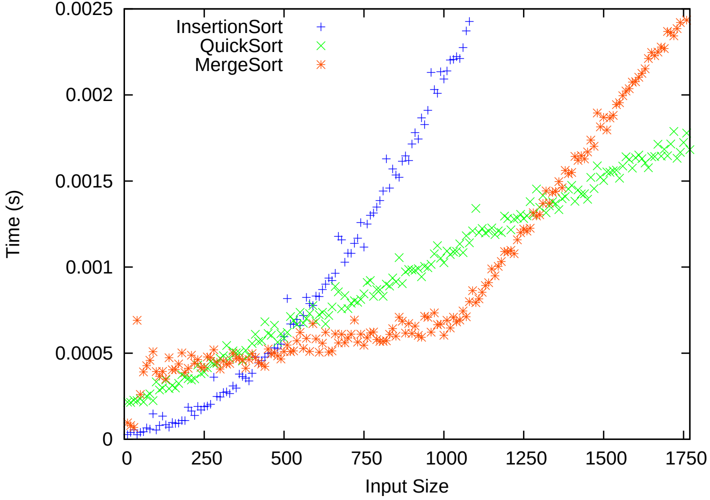
Sort algorithm timings[1]



---

[1] On an 8-way Xeon E7340 system

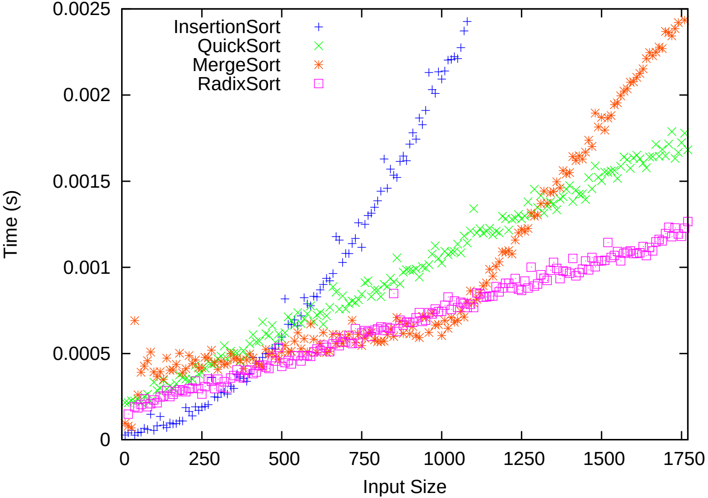
Sort algorithm timings[1]



---

[1] On an 8-way Xeon E7340 system

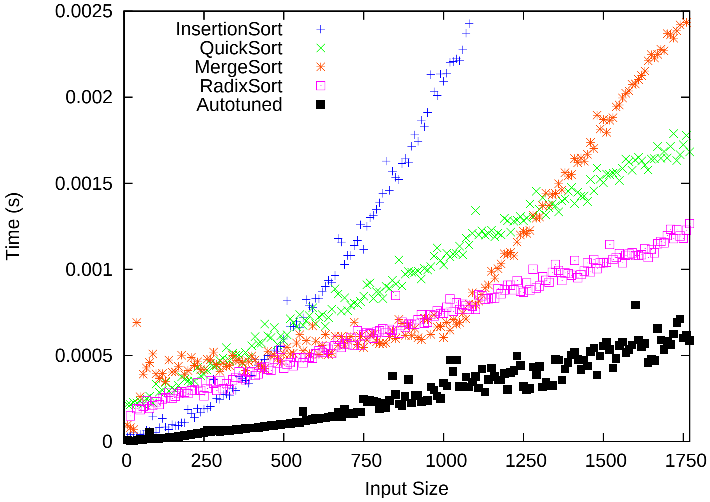
Sort algorithm timings[1]



---

[1] On an 8-way Xeon E7340 system

Sort algorithm timings[1]



---

[1] On an 8-way Xeon E7340 system

Timings on different architectures

		Trained on			
		Mobile	Xeon 1-way	Xeon 8-way	Niagara
Run on	Mobile	-	1.09x	1.67x	1.47x
	Xeon 1-way	1.61x	-	2.08x	2.50x
	Xeon 8-way	1.59x	2.14x	-	2.35x
	Niagara	1.12x	1.51x	1.08x	-

Timings on different architectures

		Trained on			
		Mobile	Xeon 1-way	Xeon 8-way	Niagara
Run on	Mobile	-	1.09x	1.67x	**1.47x**
	Xeon 1-way	1.61x	-	2.08x	**2.50x**
	Xeon 8-way	1.59x	2.14x	-	**2.35x**
	Niagara	1.12x	1.51x	1.08x	-

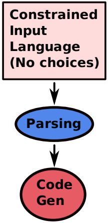
Timings on different architectures

Run on		Trained on			
		Mobile	Xeon 1-way	Xeon 8-way	Niagara
Run on	Mobile	-	1.09x	1.67x	1.47x
	Xeon 1-way	1.61x	-	**2.08x**	2.50x
	Xeon 8-way	1.59x	**2.14x**	-	2.35x
	Niagara	1.12x	1.51x	1.08x	-

Outline

1. Introduction
   - Motivating Example
   - Language & Compiler Overview
   - Why choices
2. PetaBricks Language
   - Key Ideas
   - Compilation Example
   - Other Language Features
3. Results
   - Benchmarks
   - Scalability
   - Variable Accuracy
4. Conclusion
   - Final thoughts

Early compilers



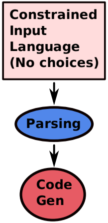
Constrained Input Language (No choices)

Parsing

Code Gen

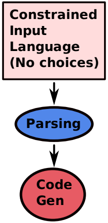
- Early computers (and compilers) were weak

Early compilers



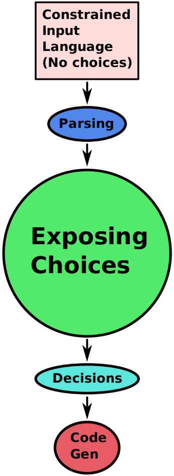
- Early computers (and compilers) were weak
- Parsing and code generation dominated compilation

Early compilers



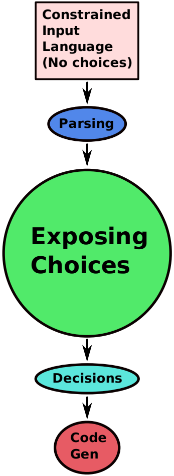
- Early computers (and compilers) were weak
- Parsing and code generation dominated compilation
- Needed a constrained input language to simplify compilation

Current compilers



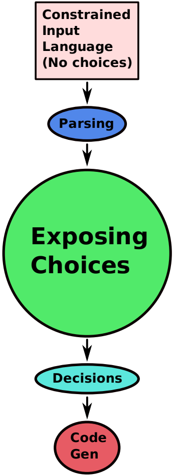
- Current computers are much more powerful
- Compilers can do a lot more

Current compilers



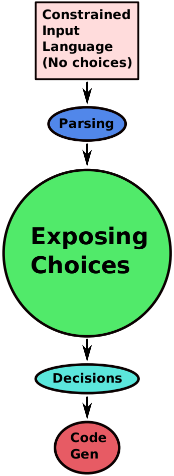
- Current computers are much more powerful
- Compilers can do a lot more
- Input language is still constraining

Current compilers



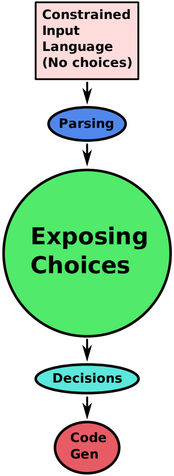
- Current computers are much more powerful
- Compilers can do a lot more
- Input language is still constraining
- Compilation dominated by exposing choices

Current compilers



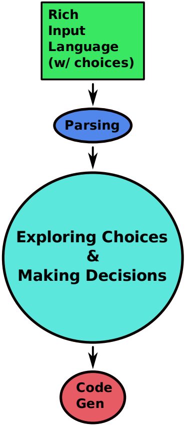
- Current computers are much more powerful
- Compilers can do a lot more
- Input language is still constraining
- Compilation dominated by exposing choices
- Input language specifies only **one**
  - Algorithmic choice
  - Iteration order choice
  - Parallelism strategy choice
  - Data layout choice

Current compilers



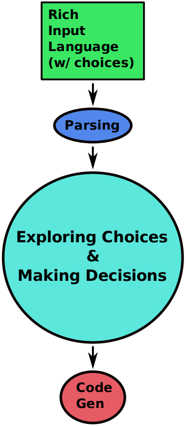
- Current computers are much more powerful
- Compilers can do a lot more
- Input language is still constraining
- Compilation dominated by exposing choices
- Input language specifies only **one**
  - Algorithmic choice
  - Iteration order choice
  - Parallelism strategy choice
  - Data layout choice
- Compiler must perform heroic analysis to reconstruct other choices

PetaBricks compiler



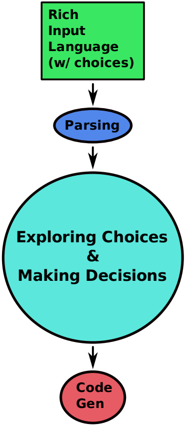
- We propose explicit choices in the language

PetaBricks compiler



- We propose explicit choices in the language
- The programmer defines the space of legal
  - Algorithmic choices
  - Iteration orders (include parallel)
  - Data layouts

PetaBricks compiler



- We propose explicit choices in the language
- The programmer defines the space of legal
  - Algorithmic choices
  - Iteration orders (include parallel)
  - Data layouts
- Allow compilers to focus on exploring choices
- Compiler no longer needs to reconstruct choices

Future-proof programs

- The result: programs can adapt to their environment

Future-proof programs

- The result: programs can adapt to their environment
- Choices make programs less brittle

Future-proof programs

- The result: programs can adapt to their environment
- Choices make programs less brittle
- Programs change with architecture, available cores, inputs, etc

Outline

Algorithmic choice in the language

- Algorithmic choice is the key aspect of PetaBricks

Algorithmic choice in the language

- Algorithmic choice is the key aspect of PetaBricks
- Programmer can define multiple rules to compute the same data

Algorithmic choice in the language

- Algorithmic choice is the key aspect of PetaBricks
- Programmer can define multiple rules to compute the same data
- Compiler re-use rules to create hybrid algorithms

Algorithmic choice in the language

- Algorithmic choice is the key aspect of PetaBricks
- Programmer can define multiple rules to compute the same data
- Compiler re-use rules to create hybrid algorithms
- Can express choices at many different granularities

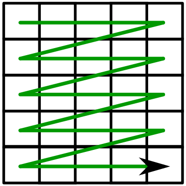
Synthesized outer control flow

- Outer control flow synthesized by compiler

Synthesized outer control flow

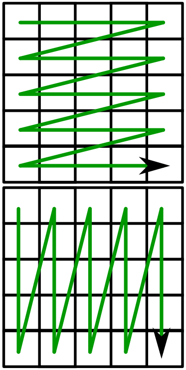
- Outer control flow synthesized by compiler
- Another choice that the programmer should not make

Synthesized outer control flow



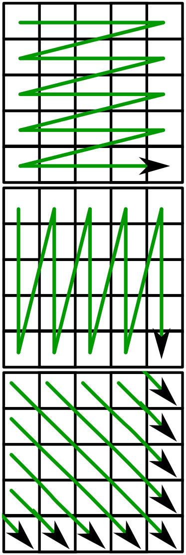
- Outer control flow synthesized by compiler
- Another choice that the programmer should not make
    - By rows?

Synthesized outer control flow



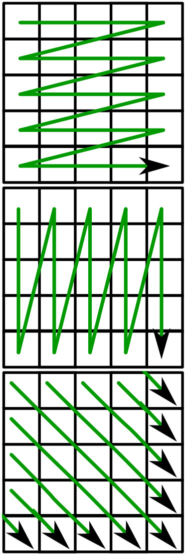
- Outer control flow synthesized by compiler
- Another choice that the programmer should not make
  - By rows?
  - By columns?

Synthesized outer control flow



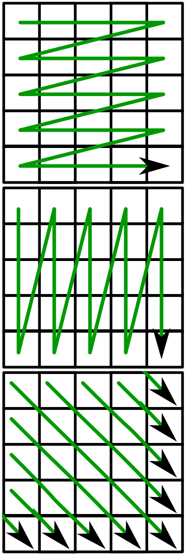
- Outer control flow synthesized by compiler
- Another choice that the programmer should not make
  - By rows?
  - By columns?
  - Diagonal? Reverse order? Blocked?
  - Parallel?

Synthesized outer control flow



- Outer control flow synthesized by compiler
- Another choice that the programmer should not make
  - By rows?
  - By columns?
  - Diagonal? Reverse order? Blocked?
  - Parallel?
- Instead programmer provides explicit producer-consumer relations

Synthesized outer control flow



- Outer control flow synthesized by compiler
- Another choice that the programmer should not make
  - By rows?
  - By columns?
  - Diagonal? Reverse order? Blocked?
  - Parallel?
- Instead programmer provides explicit producer-consumer relations
- Allows compiler to explore choice space

Outline

1. Introduction
   - Motivating Example
   - Language & Compiler Overview
   - Why choices
2. PetaBricks Language
   - Key Ideas
   - Compilation Example
   - Other Language Features
3. Results
   - Benchmarks
   - Scalability
   - Variable Accuracy
4. Conclusion
   - Final thoughts

Simple example program

```
transform RollingSum
from A[n]
to B[n]
{
  //rule 0: use the previously computed value
  B.cell(i) from(A.cell(i) a,
                 B.cell(i-1) leftSum) {
    return a+leftSum;
  }

  //rule 1: sum all elements to the left
  B.cell(i) from(A.region(0, i) in) {
    return sum(in);
  }
}
```

Simple example program

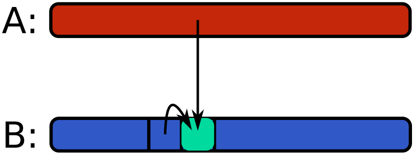
```
transform RollingSum
from A[n]
to B[n]
{
  //rule 0: use the previously computed value
  B.cell(i) from(A.cell(i) a,
                 B.cell(i-1) leftSum) {
    return a+leftSum;
  }

  //rule 1: sum all elements to the left
  B.cell(i) from(A.region(0, i) in) {
    return sum(in);
  }
}
```

Simple example program

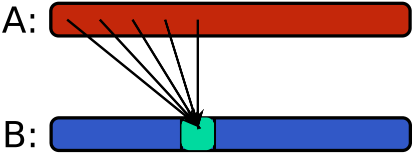
...

```
  //rule 0: use the previously computed value
  B.cell(i) from(A.cell(i) a,
                 B.cell(i-1) leftSum) {
    return a+leftSum;
  }
```

...



Simple example program



```
...

  //rule 1: sum all elements to the left
  B.cell(i) from(A.region(0, i) in) {
    return sum(in);
  }

...
```

Applicable regions

Compilation Process		
Applicable regions	Choice grids	Choice dependency graph

Applicable regions

Compilation Process		
Applicable regions	Choice grids	Choice dependency graph

```
//rule 0: use the previously computed value
B.cell(i) from(A.cell(i) a,
               B.cell(i-1) leftSum) {
  return a+leftSum;
}
```

- Applicable where $1 \leq i < n$

Applicable regions

Compilation Process		
Applicable regions	Choice grids	Choice dependency graph

```
//rule 0: use the previously computed value
B.cell(i) from(A.cell(i) a,
               B.cell(i-1) leftSum) {
  return a+leftSum;
}
```

- Applicable where $1 \leq i < n$

```
//rule 1: sum all elements to the left
B.cell(i) from(A.region(0, i) in) {
  return sum(in);
}
```

- Applicable where $0 \leq i < n$

Choice grids

Compilation Process		
Applicable regions	Choice grids	Choice dependency graph

Choice grids

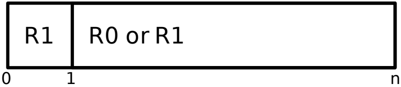
Compilation Process		
Applicable regions	Choice grids	Choice dependency graph

- Divide data space into symbolic regions with common sets of choices

Choice grids

Compilation Process		
Applicable regions	Choice grids	Choice dependency graph

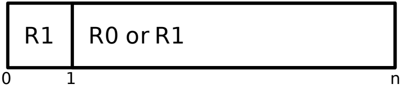
- Divide data space into symbolic regions with common sets of choices
- In this simple example:
  - A: Input (no choices)
  - B: $[0, 1)$ = rule 1
  - B: $[1, n)$ = rule 0 *or* rule 1



Choice grids

Compilation Process		
Applicable regions	Choice grids	Choice dependency graph

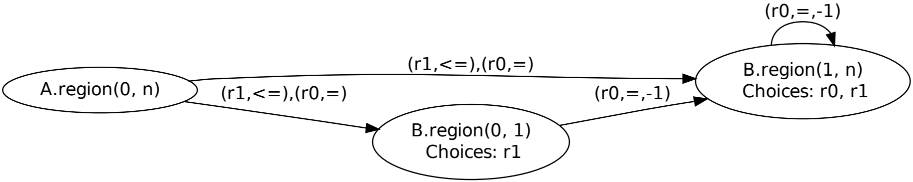
- Divide data space into symbolic regions with common sets of choices
- In this simple example:
    - A: Input (no choices)
    - B: $[0, 1)$ = rule 1
    - B: $[1, n)$ = rule 0 *or* rule 1



- Applicable regions map rules → symbolic data
- Choice grids map symbolic data → rules

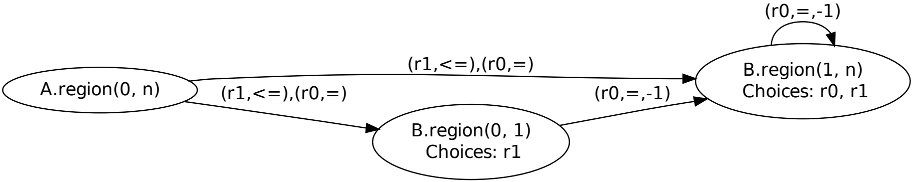
Choice dependency graph

Compilation Process		
Applicable regions	Choice grids	Choice dependency graph



Choice dependency graph

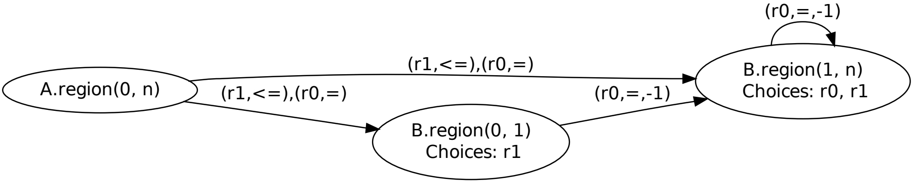
Compilation Process		
Applicable regions	Choice grids	Choice dependency graph



- Adds dependency edges between symbolic regions

Choice dependency graph

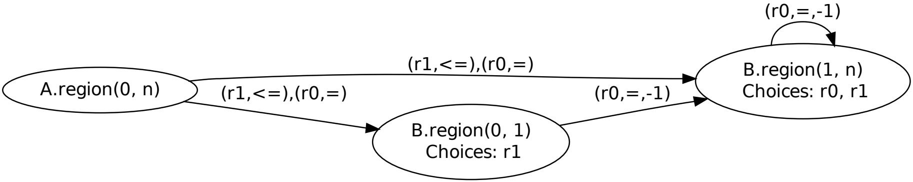
Compilation Process		
Applicable regions	Choice grids	Choice dependency graph



- Adds dependency edges between symbolic regions
- Edges annotated with directions and rules

Choice dependency graph

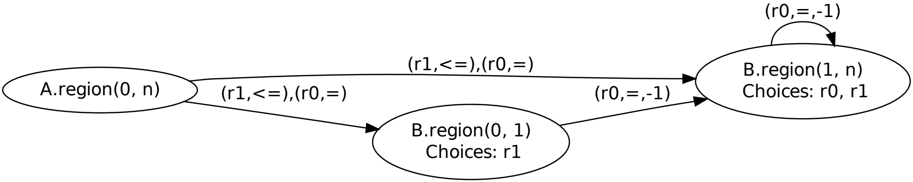
Compilation Process		
Applicable regions	Choice grids	Choice dependency graph



- Adds dependency edges between symbolic regions
- Edges annotated with directions and rules
- Many compiler passes on this IR to:

Choice dependency graph

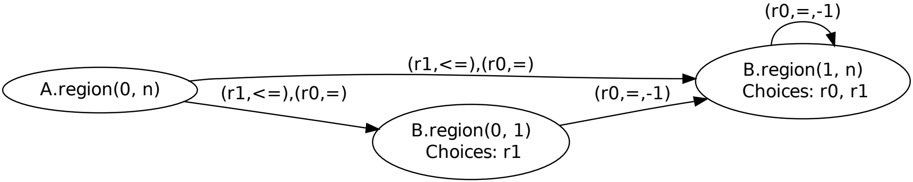
Compilation Process		
Applicable regions	Choice grids	Choice dependency graph



- Adds dependency edges between symbolic regions
- Edges annotated with directions and rules
- Many compiler passes on this IR to:
    - Simplify complex dependency patterns

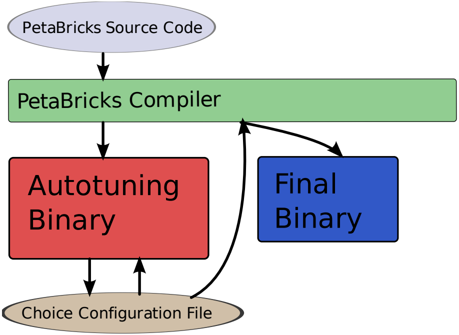
Choice dependency graph

Compilation Process		
Applicable regions	Choice grids	Choice dependency graph



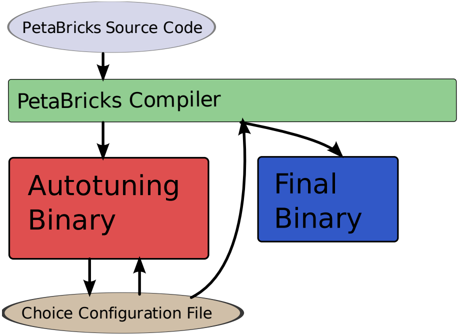
- Adds dependency edges between symbolic regions
- Edges annotated with directions and rules
- Many compiler passes on this IR to:
    - Simplify complex dependency patterns
    - Add choices

Code generation



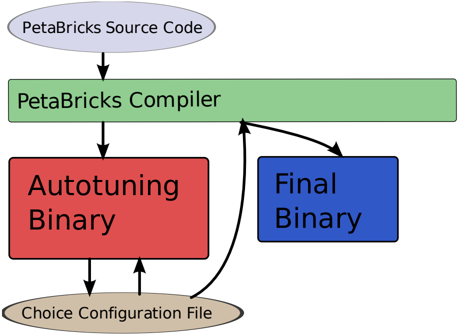
1. PetaBricks source code is compiled

Code generation



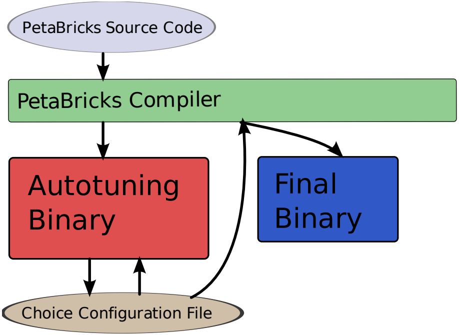
1. PetaBricks source code is compiled
2. An autotuning binary is created

Code generation



1. PetaBricks source code is compiled
2. An autotuning binary is created
3. Autotuning occurs creating a choice configuration file

Code generation



1. PetaBricks source code is compiled
2. An autotuning binary is created
3. Autotuning occurs creating a choice configuration file
4. Choices are fed back into the compiler to create a final binary

Autotuning

- Based on two building blocks:
    - A genetic tuner
    - An *n*-ary search algorithm

Autotuning

- Based on two building blocks:
    - A genetic tuner
    - An *n*-ary search algorithm
- Flat parameter space
- Compiler generates a dependency graph describing this parameter space

Autotuning

- Based on two building blocks:
  - A genetic tuner
  - An *n*-ary search algorithm
- Flat parameter space
- Compiler generates a dependency graph describing this parameter space
- Entire program tuned from bottom up

Parallel Runtime Library

- Task-based parallel runtime
- Thread-local decks of runnable tasks

Parallel Runtime Library

- Task-based parallel runtime
- Thread-local decks of runnable tasks
- Use a work-stealing algorithm similar to that of Cilk

Outline

1. Introduction
   - Motivating Example
   - Language & Compiler Overview
   - Why choices
2. PetaBricks Language
   - Key Ideas
   - Compilation Example
   - **Other Language Features**
3. Results
   - Benchmarks
   - Scalability
   - Variable Accuracy
4. Conclusion
   - Final thoughts

More PetaBricks features

- Automatic consistency checking
- The *tunable* keyword
- Call external code
- Custom training data generators
- Matrix *versions* for iterative algorithms
- Rule priorities
- `where` (clause for limiting applicable regions)
- Template transforms

More PetaBricks features

- Automatic consistency checking
- The *tunable* keyword
- Call external code
- Custom training data generators
- Matrix *versions* for iterative algorithms
- Rule priorities
- `where` (clause for limiting applicable regions)
- Template transforms

More PetaBricks features

- Automatic consistency checking
- The *tunable* keyword
- Call external code
- Custom training data generators
- Matrix *versions* for iterative algorithms
- Rule priorities
- `where` (clause for limiting applicable regions)
- Template transforms

More PetaBricks features

- Automatic consistency checking
- The *tunable* keyword
- Call external code
- Custom training data generators
- Matrix *versions* for iterative algorithms
- Rule priorities
- `where` (clause for limiting applicable regions)
- Template transforms

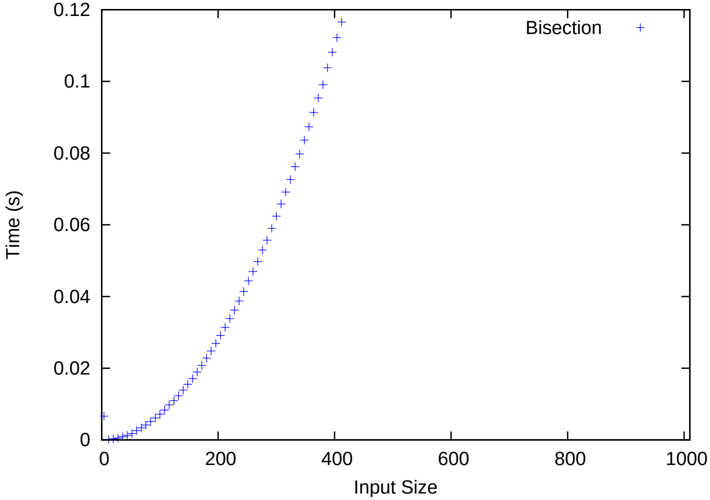
Outline

1. Introduction
   - Motivating Example
   - Language & Compiler Overview
   - Why choices
2. PetaBricks Language
   - Key Ideas
   - Compilation Example
   - Other Language Features
3. Results
   - Benchmarks
   - Scalability
   - Variable Accuracy
4. Conclusion
   - Final thoughts

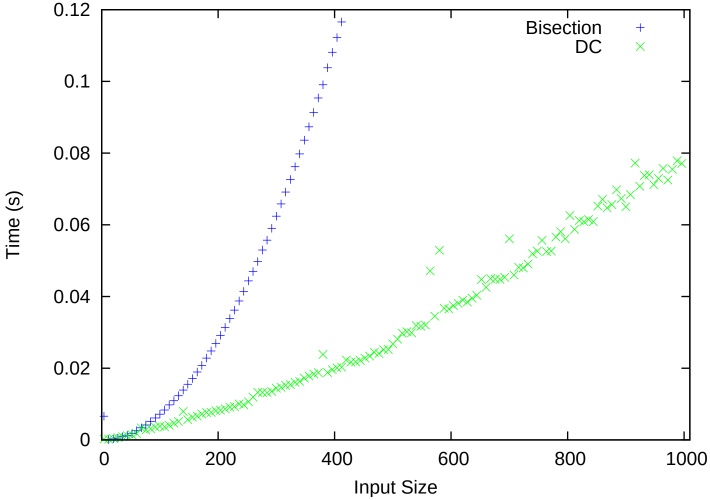
Eigenvector Solve

- Bisection
- QR decomposition
- Divide and conquer

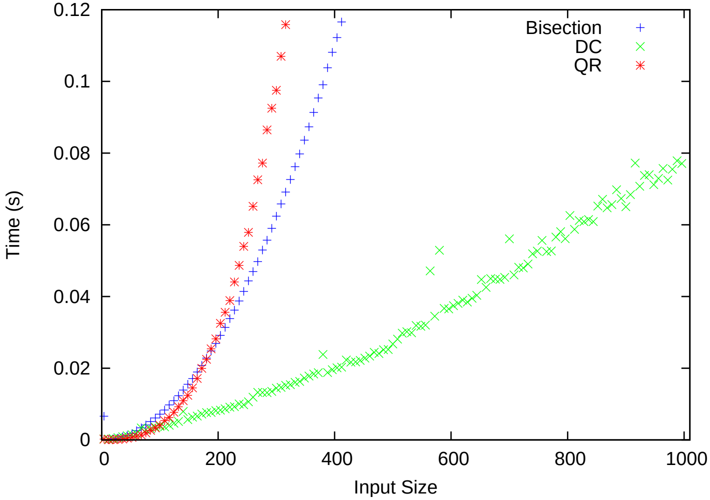
Eigenvector Solve



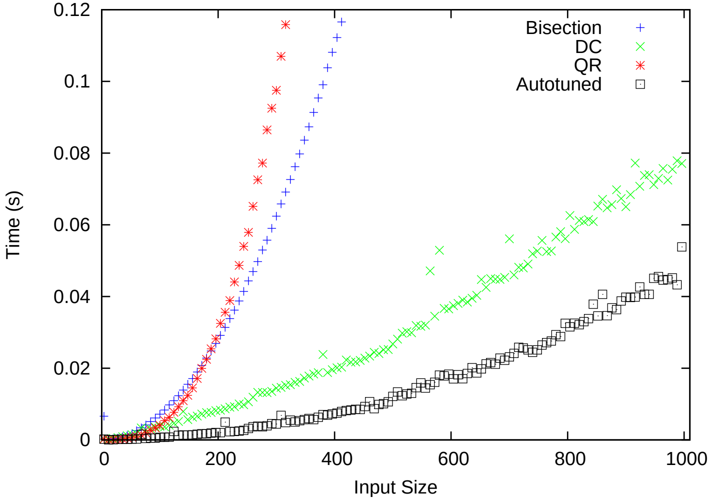
Eigenvector Solve



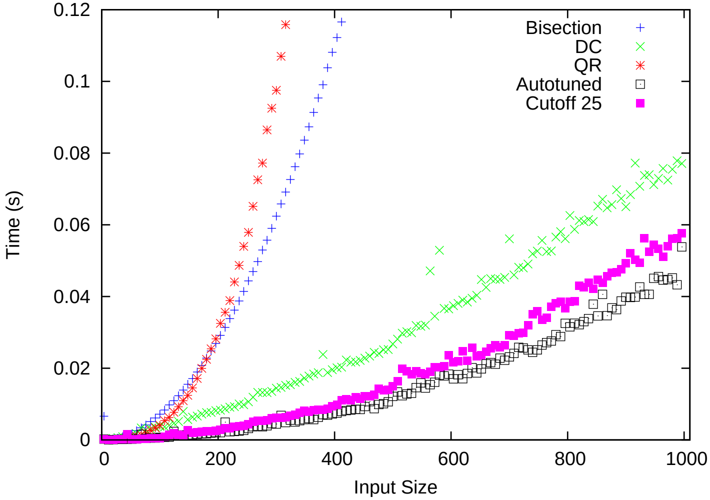
Eigenvector Solve



Eigenvector Solve



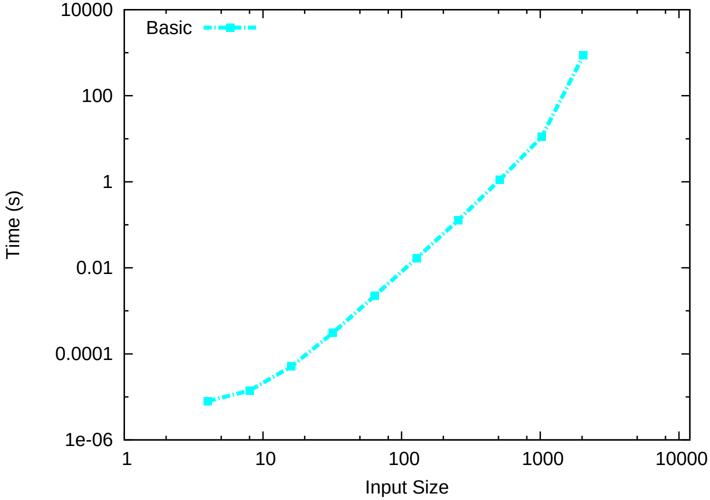
Eigenvector Solve



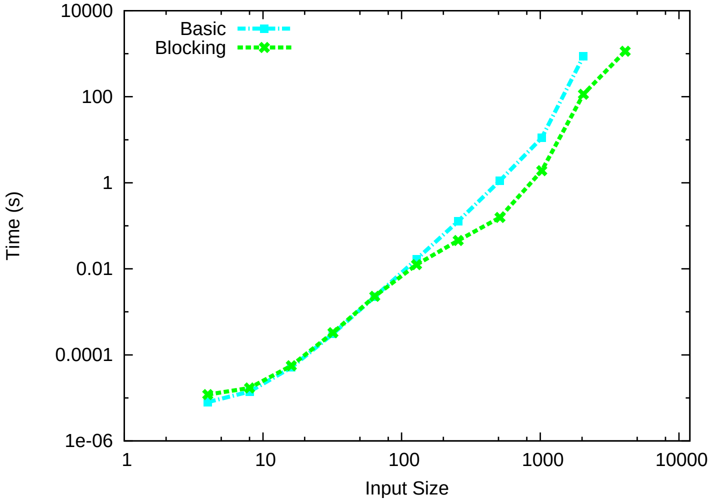
Matrix Multiply

- Basic
- Recursive decompositions
- Strassen's algorithm
- Iteration order (blocking)
- Transpose

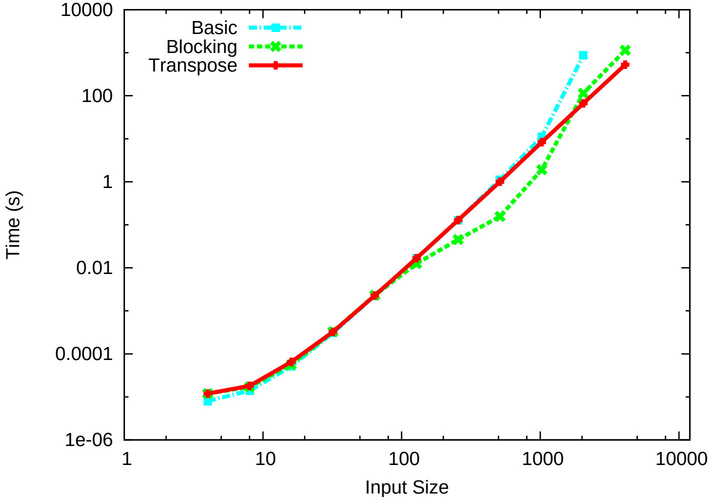
Matrix Multiply



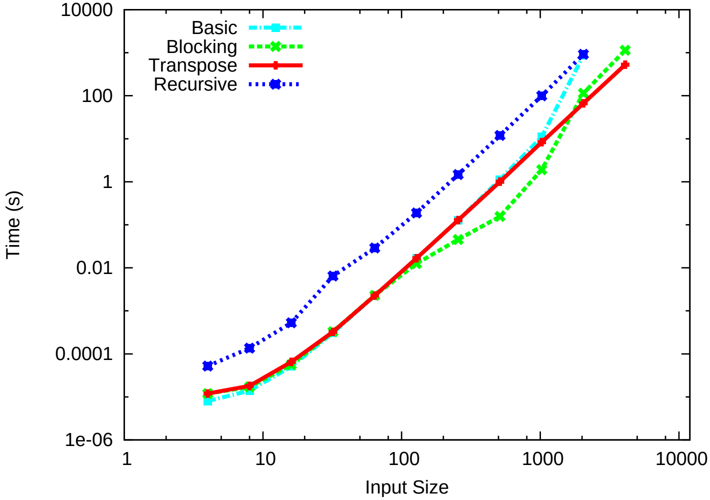
Matrix Multiply



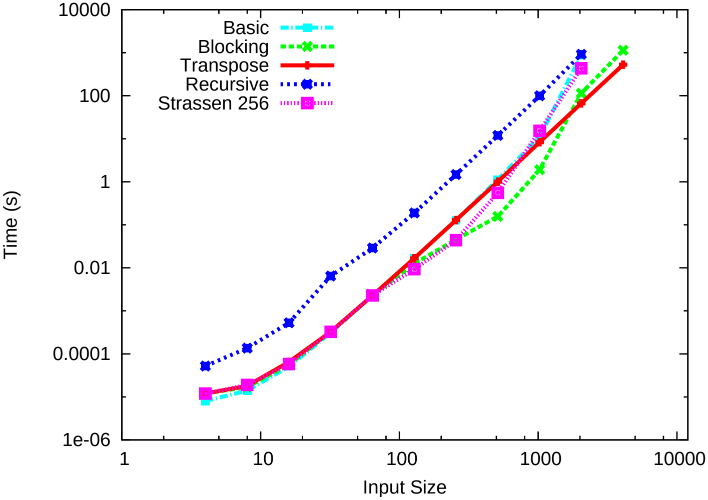
Matrix Multiply



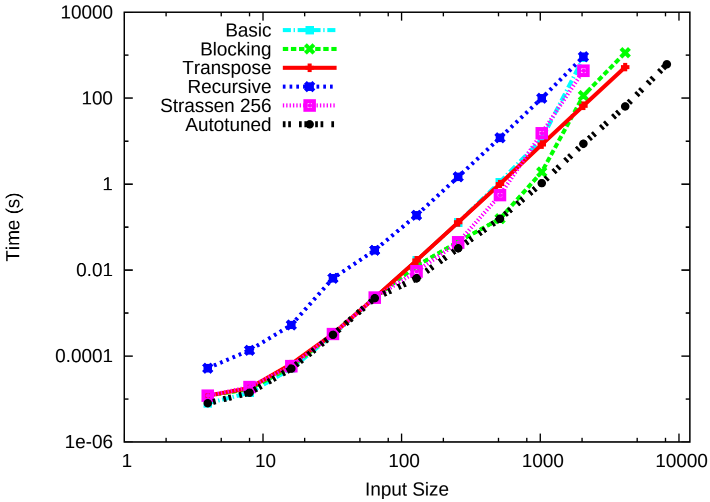
Matrix Multiply



Matrix Multiply



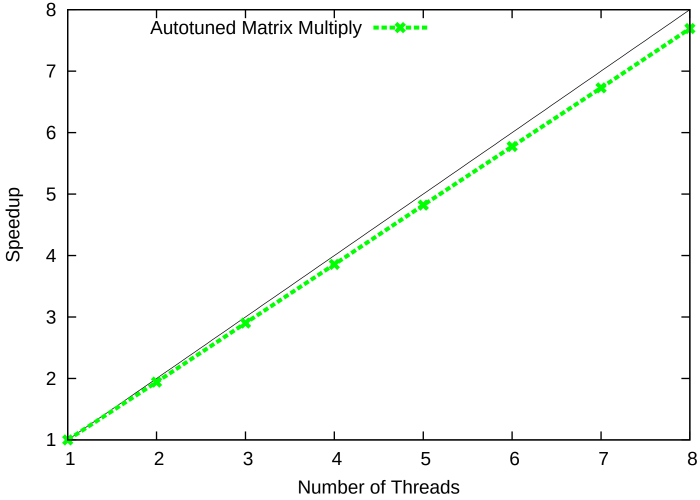
Matrix Multiply



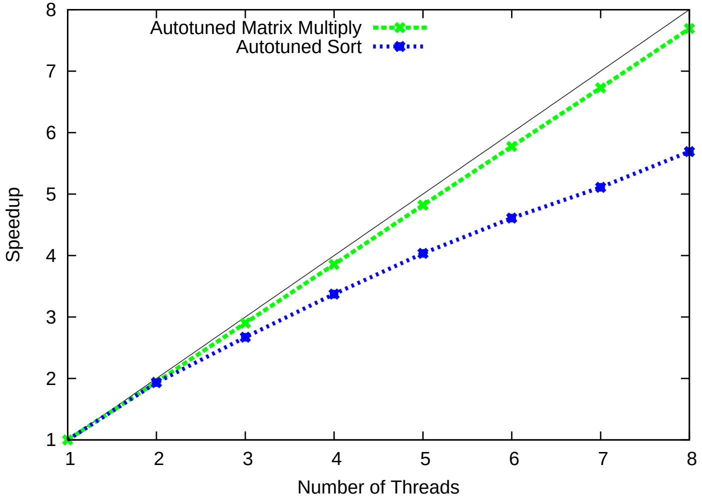
Outline

1. Introduction
   - Motivating Example
   - Language & Compiler Overview
   - Why choices
2. PetaBricks Language
   - Key Ideas
   - Compilation Example
   - Other Language Features
3. Results
   - Benchmarks
   - Scalability
   - Variable Accuracy
4. Conclusion
   - Final thoughts

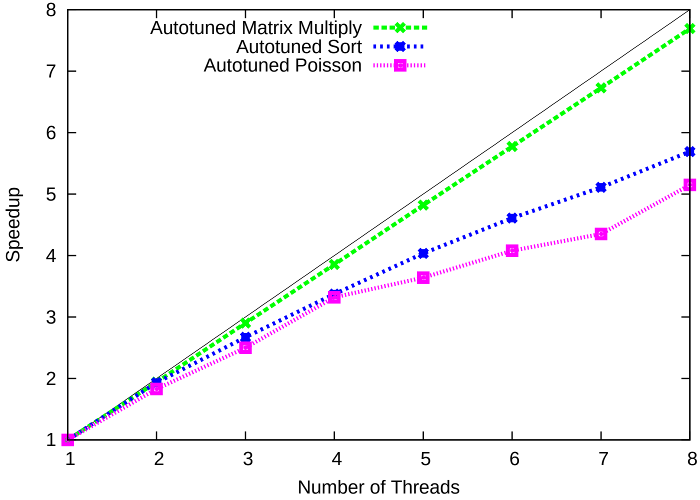
Scalability



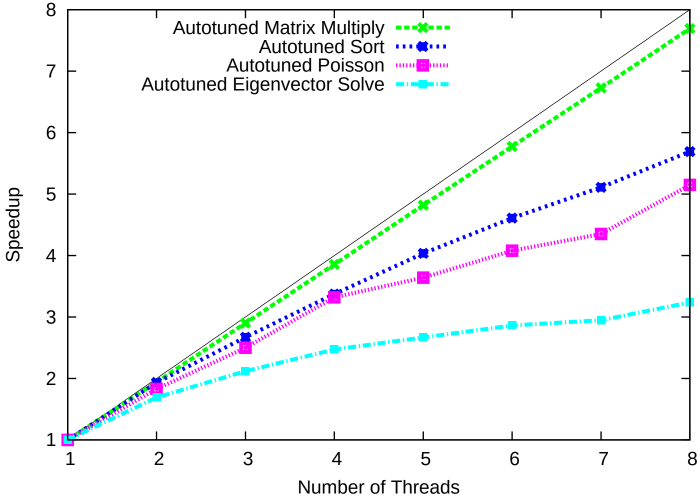
Scalability



Scalability



Scalability



Outline

1. Introduction
   - Motivating Example
   - Language & Compiler Overview
   - Why choices
2. PetaBricks Language
   - Key Ideas
   - Compilation Example
   - Other Language Features
3. Results
   - Benchmarks
   - Scalability
   - Variable Accuracy
4. Conclusion
   - Final thoughts

Variable accuracy

- Most algorithms produce exact solutions

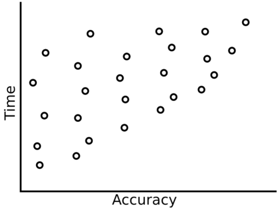
Variable accuracy

- Most algorithms produce exact solutions
- Large class of algorithms can produce approximate solutions

Variable accuracy

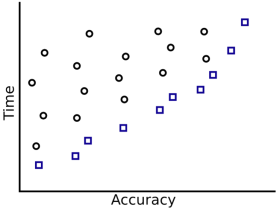
- Most algorithms produce exact solutions
- Large class of algorithms can produce approximate solutions
    - Iterative convergence
    - Grid coarsening
    - Others

Variable accuracy



- Most algorithms produce exact solutions
- Large class of algorithms can produce approximate solutions
  - Iterative convergence
  - Grid coarsening
  - Others
- Compiler/autotuner should be aware of variable accuracy

Variable accuracy

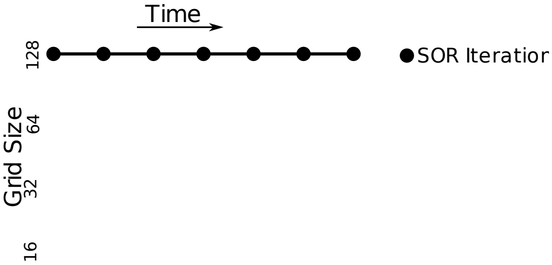


- Most algorithms produce exact solutions
- Large class of algorithms can produce approximate solutions
  - Iterative convergence
  - Grid coarsening
  - Others
- Compiler/autotuner should be aware of variable accuracy
- Compiler can examine optimal frontier of algorithms

Poisson's equation

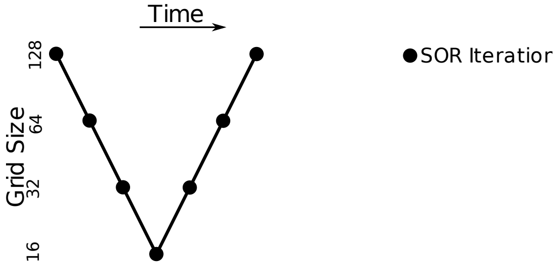
- A variable accuracy benchmark
- Accuracy level expressed as a template parameter
- Autotuner exploits variable accuracy in a general way
- Choices:
    - Direct solve
    - Jacobi iteration
    - Successive over relaxation
    - Multigrid

Choices in Multigrid



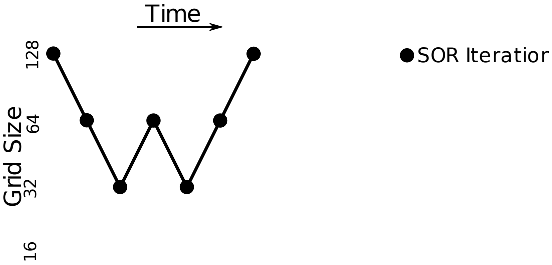
- SOR is an iterative algorithm

Choices in Multigrid



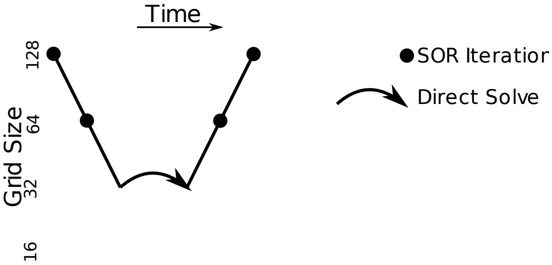
- SOR is an iterative algorithm
- Multigrid changes grid coarseness to speed up convergence
- Many standard shapes: V-Cycle,

Choices in Multigrid



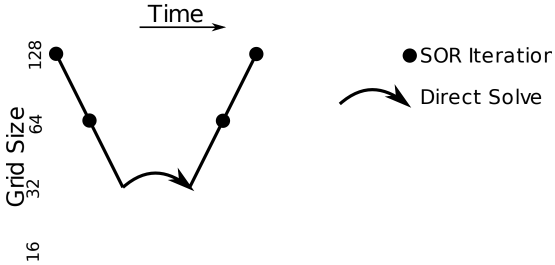
- SOR is an iterative algorithm
- Multigrid changes grid coarseness to speed up convergence
- Many standard shapes: V-Cycle, W-Cycle, etc

Choices in Multigrid



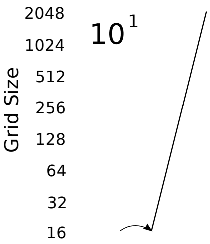
- SOR is an iterative algorithm
- Multigrid changes grid coarseness to speed up convergence
- Many standard shapes: V-Cycle, W-Cycle, etc
- Direct solver

Choices in Multigrid

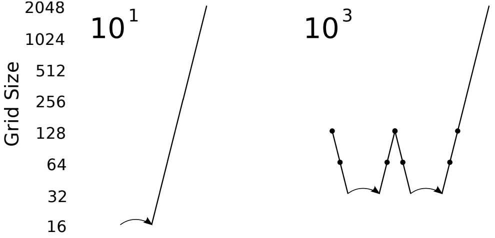


- SOR is an iterative algorithm
- Multigrid changes grid coarseness to speed up convergence
- Many standard shapes: V-Cycle, W-Cycle, etc
- Direct solver
- Different shapes = different algorithms

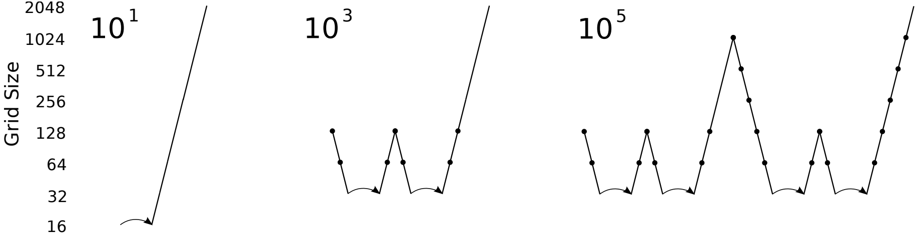
Autotuned V-cycle shapes for different accuracy requirements



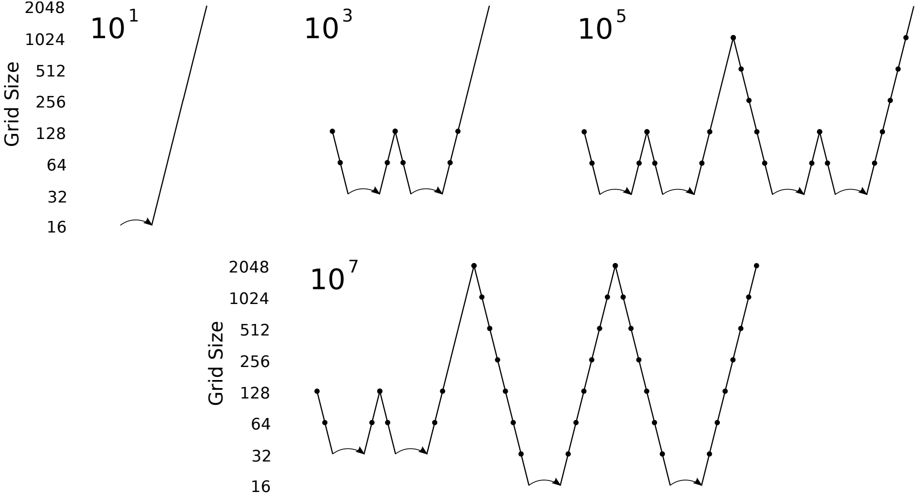
Autotuned V-cycle shapes for different accuracy requirements



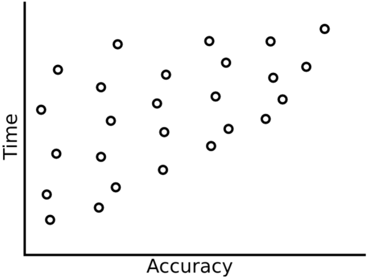
Autotuned V-cycle shapes for different accuracy requirements



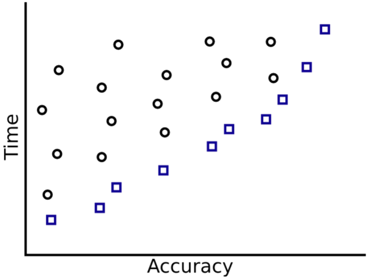
Autotuned V-cycle shapes for different accuracy requirements



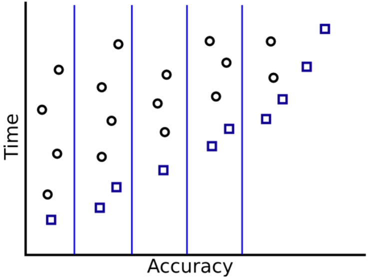
Dynamic programming technique for autotuning Multigrid



Dynamic programming technique for autotuning Multigrid

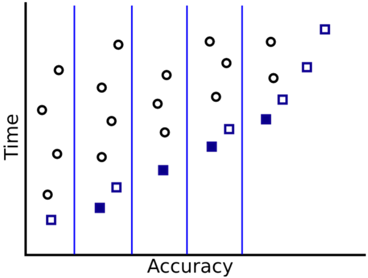


Dynamic programming technique for autotuning Multigrid



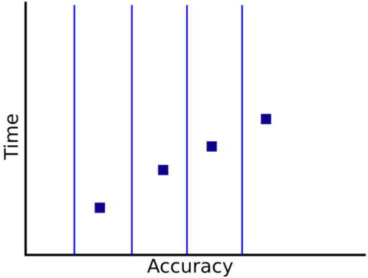
- Partition accuracy space into discrete levels

Dynamic programming technique for autotuning Multigrid



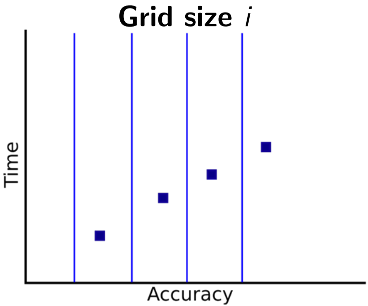
- Partition accuracy space into discrete levels

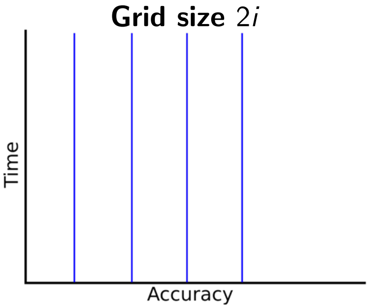
Dynamic programming technique for autotuning Multigrid



- Partition accuracy space into discrete levels

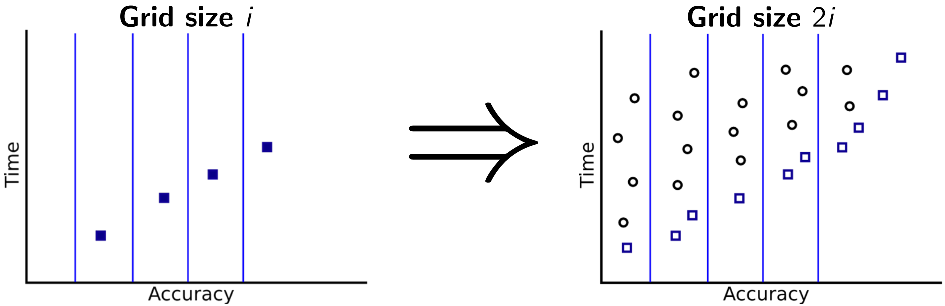
Dynamic programming technique for autotuning Multigrid





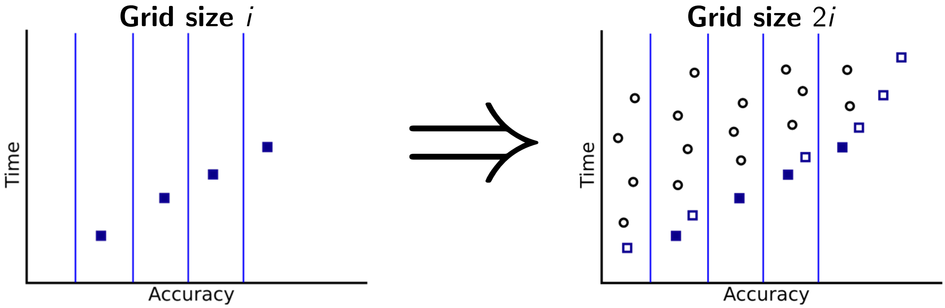
- Partition accuracy space into discrete levels

Dynamic programming technique for autotuning Multigrid



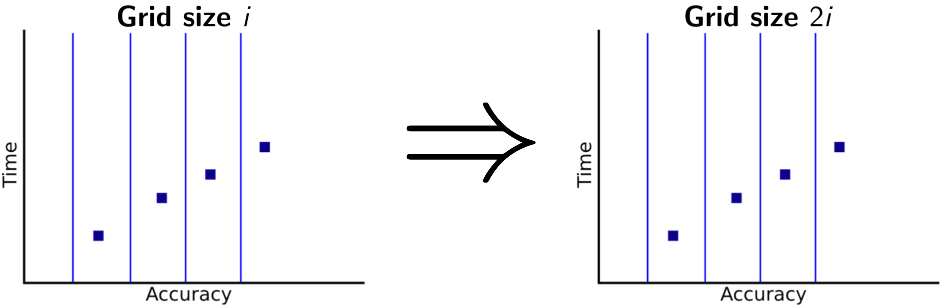
- Partition accuracy space into discrete levels
- Base space of candidate algorithms on optimal algorithms from coarser level

Dynamic programming technique for autotuning Multigrid



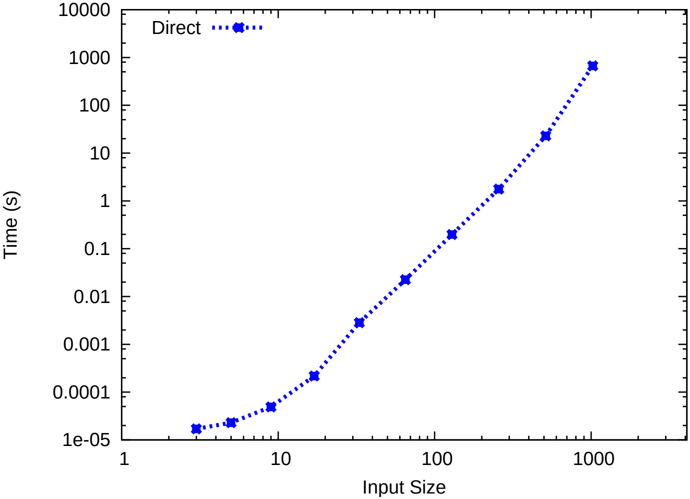
- Partition accuracy space into discrete levels
- Base space of candidate algorithms on optimal algorithms from coarser level

Dynamic programming technique for autotuning Multigrid

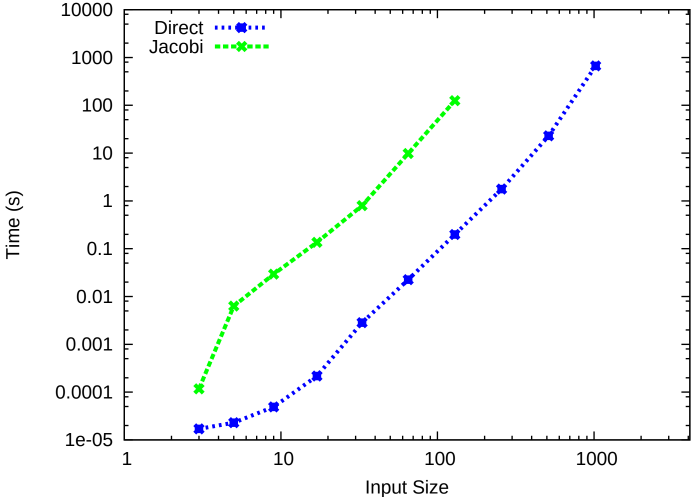


- Partition accuracy space into discrete levels
- Base space of candidate algorithms on optimal algorithms from coarser level

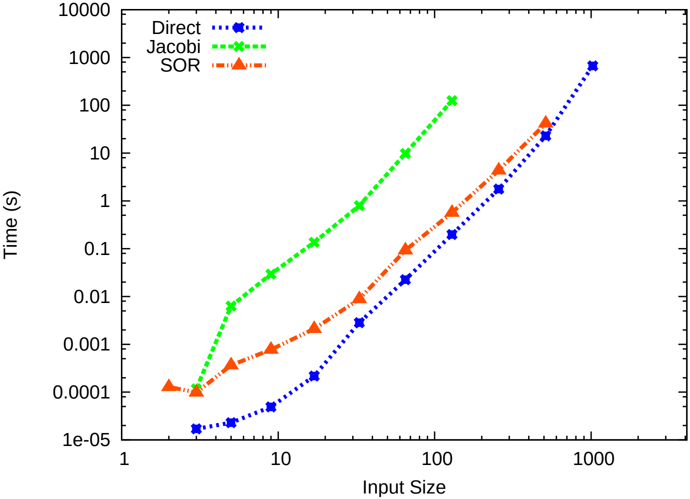
Poisson's Equation



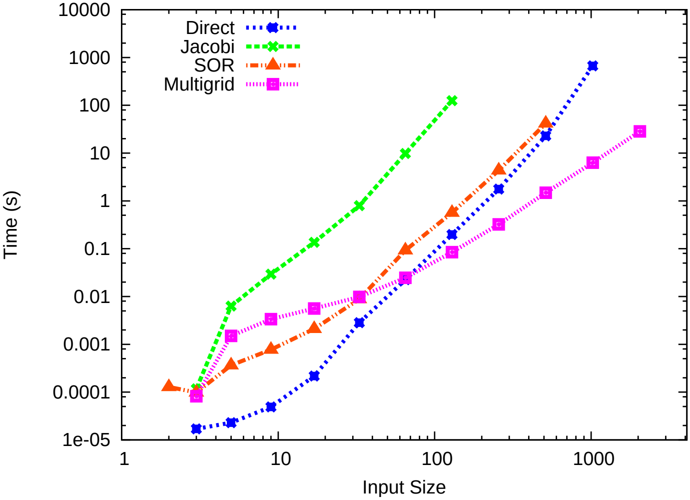
Poisson's Equation



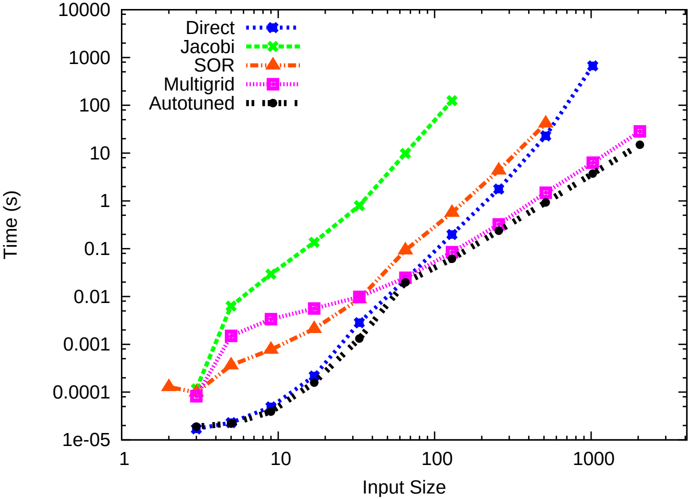
Poisson's Equation



Poisson's Equation



Poisson's Equation



Outline

1. Introduction
   - Motivating Example
   - Language & Compiler Overview
   - Why choices
2. PetaBricks Language
   - Key Ideas
   - Compilation Example
   - Other Language Features
3. Results
   - Benchmarks
   - Scalability
   - Variable Accuracy
4. Conclusion
   - Final thoughts

Related work

- Languages
    - Sequoia
- Libraries & domain specific tuners
    - STAPL
    - ATLAS
    - FFTW
    - SPARSITY
    - SPIRAL
    - ...

For more information

- PetaBricks makes programs **future-proof**, by allowing them to adapt to new architectures

- We plan to released PetaBricks at the end of summer
- Sign up for our mailing list to be notified
- For more information see: `http://projects.csail.mit.edu/petabricks/`

- Questions?

Thank you!