UNIVERSITY OF CAMBRIDGE

Petabricks: A language and compiler for algorithmic choice

J. Ansel et al. ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI, 2009.

Sami (sa894) – R244 Large-Scale Data Processing and Optimisation, sess 6

Motivation – Many algorithms with various trade offs

Sorting

MergeSort	QuickSort	InsertionSort	RadixSort
Fast for medium size input	Fastest for medium size input	Fastest for small input	Fastest for largest input
Highly parallelizable	Exploit spatial locality	O(1) Memory	O(wn) runtime

- But gets more complicated!
- Mixing the algorithms yield better results
- e.g QuickSort then cut off to InsertionSort once list is small enough
  - Requires knowing an optimal cut-off point!
  - Differs from an architecture to an architecture
- Choices are left to the developer, complex, time-consuming, error prone

Hence the need for Autotuning

PetaBricks – A language and a compiler

- Self-tuning compiler for bespoke architecture
- A language that allow expressing choice in algorithms
- Implicitly parallelizable
- Auto-select the desired trade-off between accuracy and performance

Sample Code - Sorting

```
#define SORT Sort

//#include "Bitonicsort.pbcc"
#include "Insertionsort.pbcc"
#include "Mergesort2.pbcc"
#include "Quicksort2.pbcc"
#include "Selectionsort.pbcc"

function Sort
from in[n]
to out[n]
{
  Mergesort2(out, in);
} or {
  Mergesort4(out, in);
} or {
  Quicksort2(out, in);
} or {
  Insertionsort(out, in);
} or {
  Selectionsort(out, in);
}
//or {
//  Bitonic(out, in);
//}
```

- Define multiple functions
- Functions have tunable variables
- PetaBricks choose optimal combination

Code Taken from PetaBricks GitHub Repo

PetaBricks effect

- Source code is compiled into a binary
- The binary is auto-tuned on the architecture of the system
- The compiler produces a final binary contains the optimal configuration

Tuned variables - Sample

Branch: master ▾ petabricks / examples / sort / Sort.cfg.X1 Find file Copy path

Marek Olszewski New cross tuning configs 6eafee1 on Mar 30, 2009

0 contributors

52 lines (51 sloc) | 3.31 KB Raw Blame History

```
Copy1D_sequential_cutoff = 1455     # valid range: 0 to 2147483647
Copy1D_split_size = 1714     # valid range: 1 to 2147483647
Copy2D_sequential_cutoff = 568     # valid range: 0 to 2147483647
Copy2D_split_size = 910     # valid range: 1 to 2147483647
InsertionsortSubArray_sequential_cutoff = 0     # valid range: 0 to 2147483647
Insertionsort_sequential_cutoff = 0     # valid range: 0 to 2147483647
Merge16_sequential_cutoff = 1455     # valid range: 0 to 2147483647
Merge2_Parallel_Cutoff = 949     # valid range: 100 to 100000
Merge2_Parallel_sequential_cutoff = 1373     # valid range: 0 to 2147483647
Merge4_sequential_cutoff = 1334     # valid range: 0 to 2147483647
Merge8_sequential_cutoff = 569     # valid range: 0 to 2147483647
Mergesort16_sequential_cutoff = 1497     # valid range: 0 to 2147483647
Mergesort4_sequential_cutoff = 1497     # valid range: 0 to 2147483647
Mergesort8_sequential_cutoff = 1458     # valid range: 0 to 2147483647
MergesortSubArray16_sequential_cutoff = 1455     # valid range: 0 to 2147483647
MergesortSubArray4_sequential_cutoff = 1497     # valid range: 0 to 2147483647
MergesortSubArray8_sequential_cutoff = 1497     # valid range: 0 to 2147483647
Parallel_MergesortSubArray_sequential_cutoff = 945     # valid range: 0 to 2147483647
Parallel_Mergesort_sequential_cutoff = 1497     # valid range: 0 to 2147483647
QuickSort_sequential_cutoff = 1497     # valid range: 0 to 2147483647
QuicksortSubArray_sequential_cutoff = 949     # valid range: 0 to 2147483647
RadixsortSubArray_sequential_cutoff = 1714     # valid range: 0 to 2147483647
```


Code Taken from PetaBricks GitHub Repo

Sorting - Results

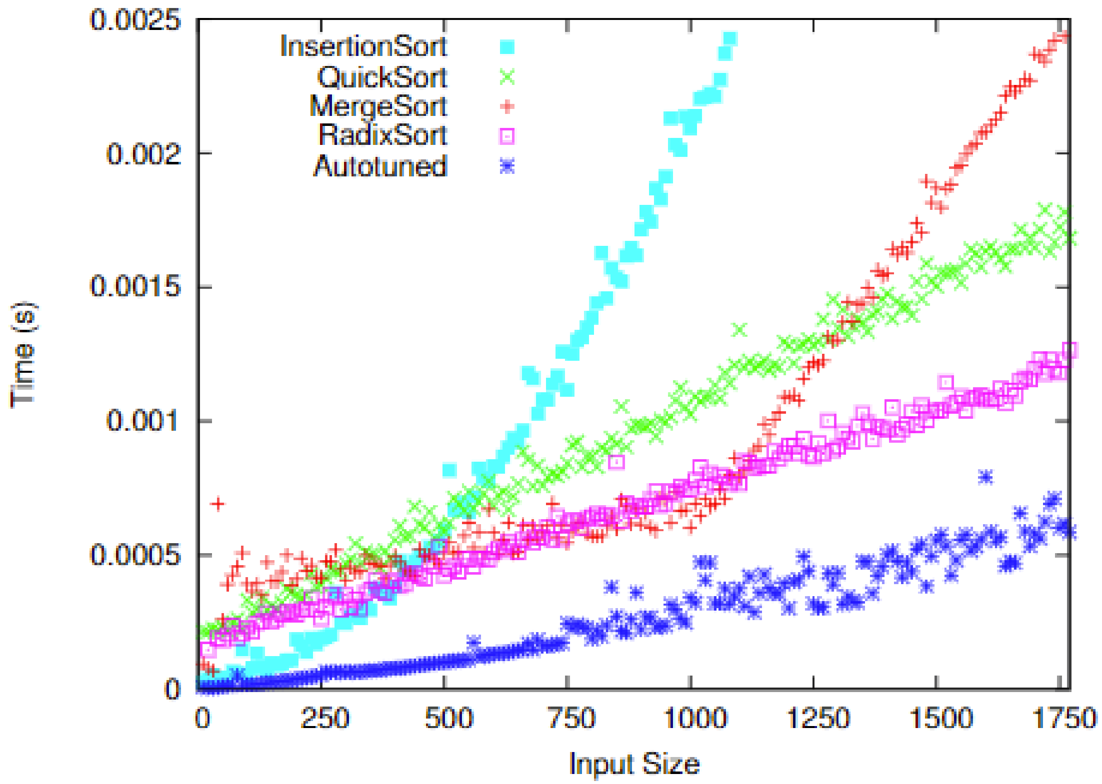


Figure 14. Performance for sort on 8 cores.

Figure taken from paper in review

PetaBricks Components Graph

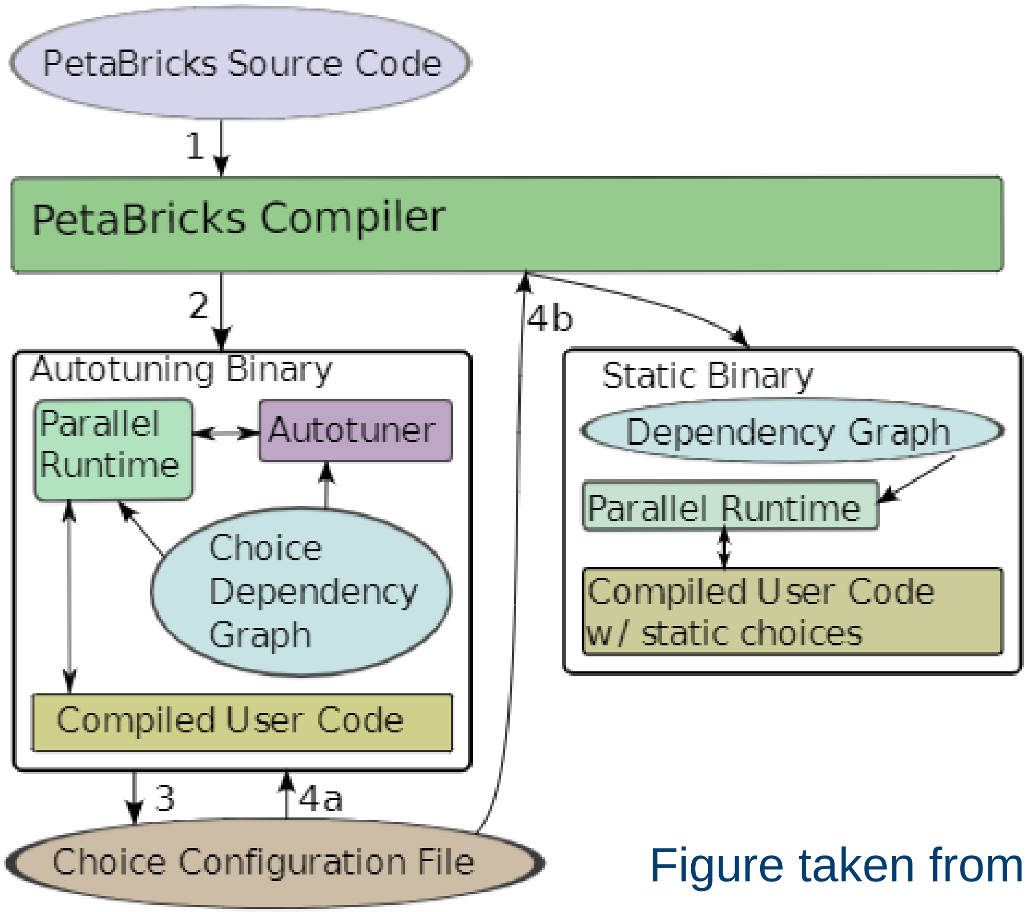


Figure taken from paper in review

PetaBricks Internals – Source2Source Compiler

- Compiles from PetaBricks to C++
- Input parsed into syntax tree
- Construct a choice grid for matrix type
- Build a choice dependency graph

Compilation – Rolling Sum example

```
transform RollingSum
from A[n]
to B[n]
{
  //rule0: sum all elements to the left
  to(B.cell(i) b) from(A.region(0, i) in) {
    b=sum(in);
  }

  //rule1: use the previously computed value
  to(B.cell(i) b) from(A.cell(i) a,
                       B.cell(i-1) leftSum) {
    b=a+leftSum;
  }
}
```

Figures taken from paper in review

Compilation – Applicable Region

```
transform RollingSum
from A[n]
to B[n]
{
  //rule0: sum all elements to the left
  to(B.cell(i) b) from(A.region(0, i) in) {
    b=sum(in);
  }

  //rule1: use the previously computed value
  to(B.cell(i) b) from(A.cell(i) a,
                       B.cell(i-1) leftSum) {
    b=a+leftSum;
  }
}
```

Figures taken from paper in review

Compilation – Applicable Region

```
transform RollingSum
from A[n]
to B[n]
{
  //rule0: sum all elements to the left
  to(B.cell(i) b) from(A.region(0, i) in) {
    b=sum(in);
  }

  //rule1: use the previously computed value
  to(B.cell(i) b) from(A.cell(i) a,
                       B.cell(i-1) leftSum) {
    b=a+leftSum;
  }
}
```

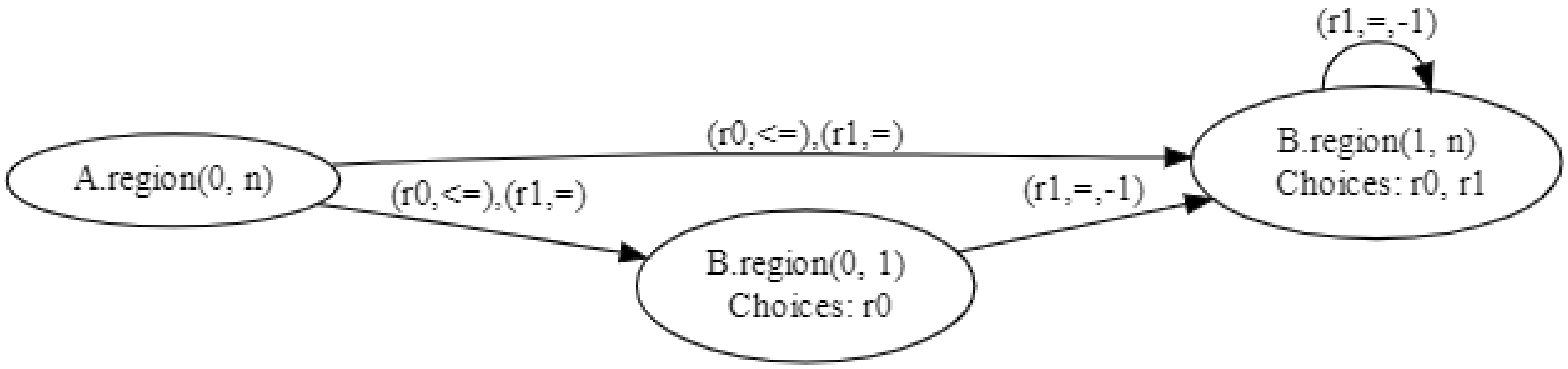
Figures taken from paper in review

Compilation - Choice Grid Analysis

- Split the data into matrix (grids)
- Map data to rules
- E.g. [0, 1) = {rule 0} only
  while [1, n) = {rule 0 or 1}
- Rule priority is applied here as well

Figures taken from paper in review

Compilation - Choice Dependency Graph



Figures taken from paper in review

Code Generation

- Two modes
  - Default – choices and autotuner information are embeded in the output code
  - Second mode – code generation with all choices eliminated based on autotuner results
  - Second mode is useful to produce an intermediate code for C++ to compile – it is more efficient when choices are eleminated

Auto Tuning System

- Tuning is done by running search on the available configurations
- The available configuration is described using the choice dependency graph
- Using bottom-up approach, works on smaller input and works it way up to large input

Runtime library

- Dynamically schedule tasks to distribute workload
- When task reach tunable cut-off point they stop calling the scheduler and execute sequentially
- Maximize locality using Cilk, task stealing protocol;
  - Thread operates on top of its dequeue
  - When it runs out of tasks
  - Select a random victim to steal work from bottom of their dequeue

PetaBricks – Other features

- Calling external libraries and other languages
- Template Transformation – Similar to C++
- Rule priorities and where clauses to manual tune edge cases
- Deadlock and race conditions prevention using the dependency graph
- Automated Consistency Checking - advantage of choices, you can run multiple versions and check their results for consistency

Evaluation - Performance

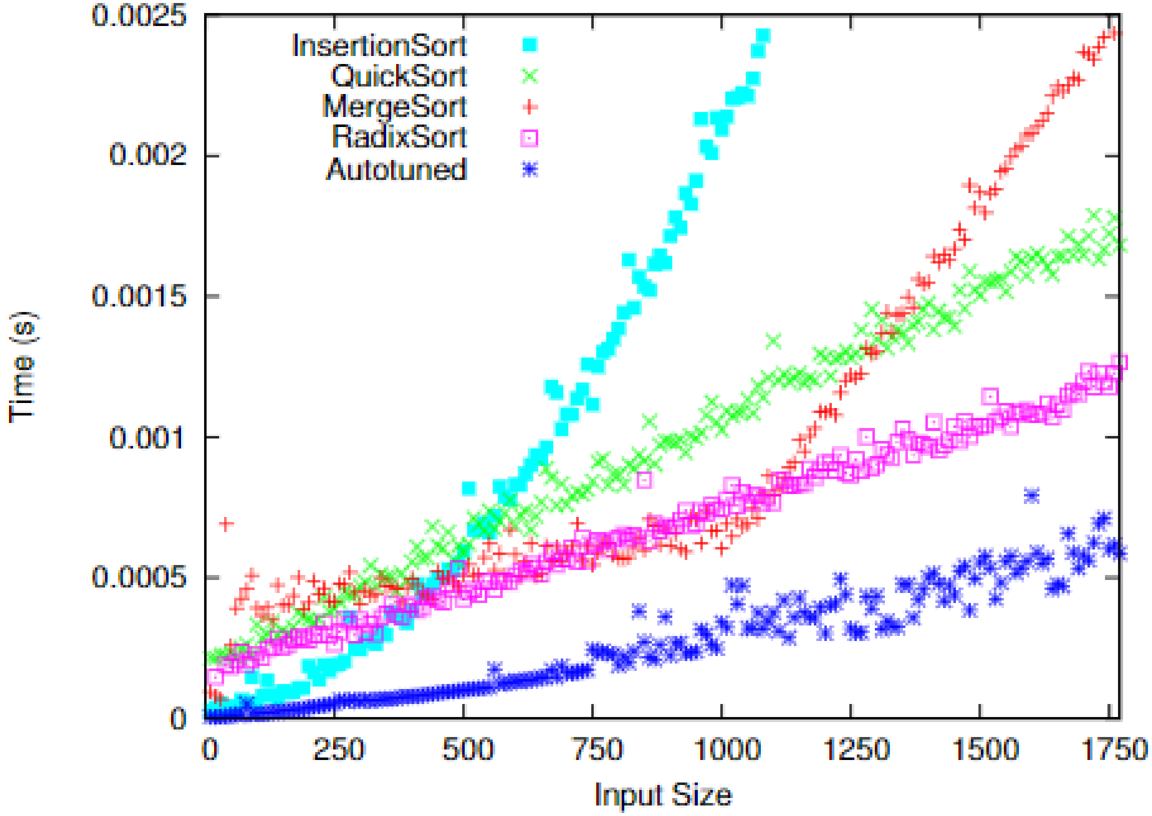


Figure 14. Performance for sort on 8 cores.

Figure taken from paper in review

Evaluation – Performance other algorithms

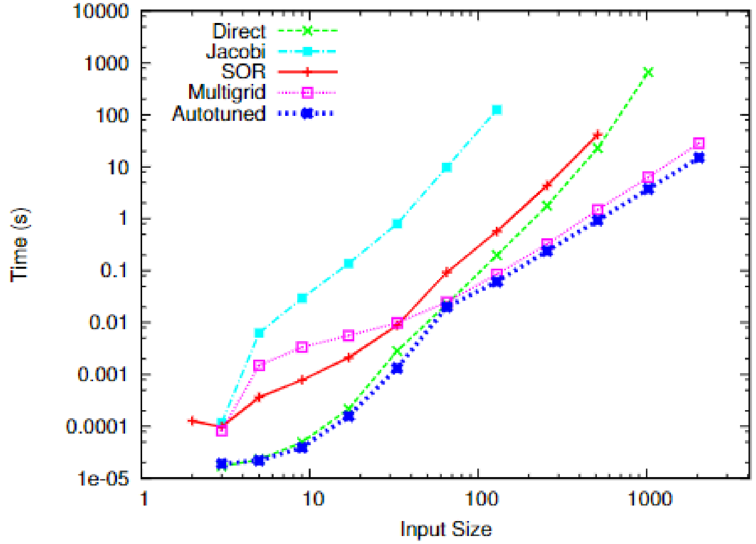


Figure 11. Performance for algorithms to solve Poisson's equation up to an accuracy of 10^9 using 8 cores. The iterated SOR algorithm uses the corresponding optimal weight ω_{opt} for each of the different input sizes

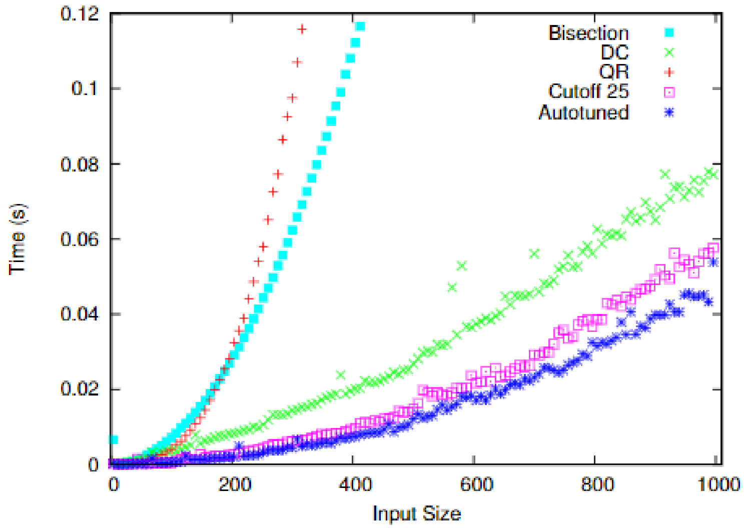


Figure 12. Performance for Eigenproblem on 8 cores. "Cutoff 25" corresponds to the hard-coded hybrid algorithm found in LAPACK.

Figures taken from paper in review

Evaluation – Parallelism

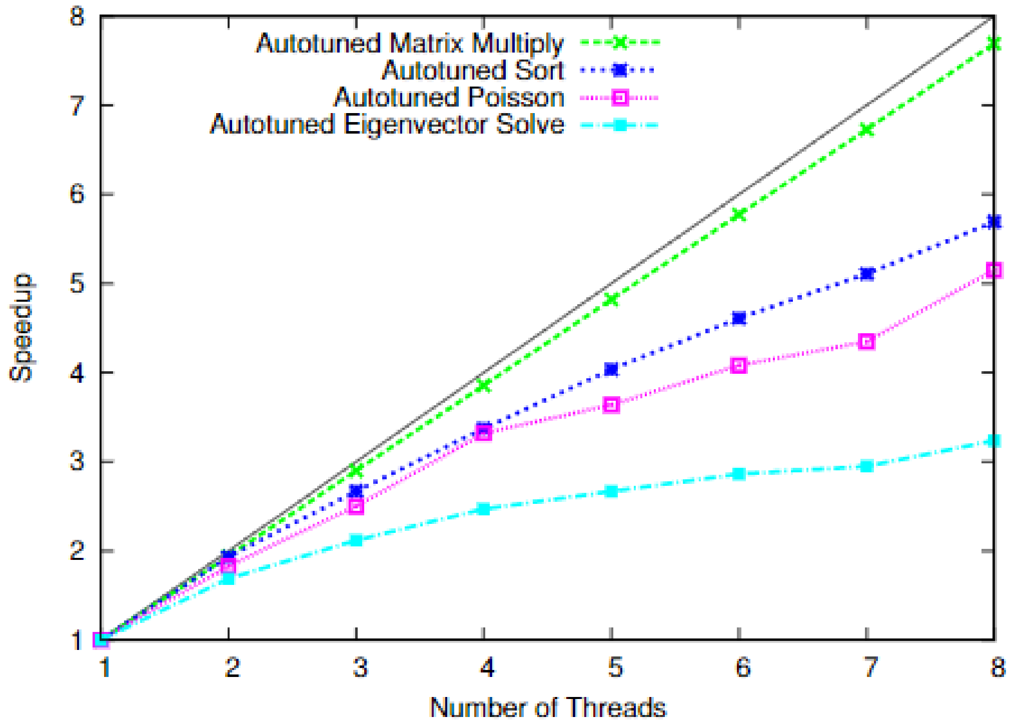


Figure 16. Parallel scalability. Speedup as more worker threads are added. Run on an 8-way (2 processor × 4 core) x86_64 Intel Xeon System.

Figure taken from paper in review

Evaluation – Accuracy for performance

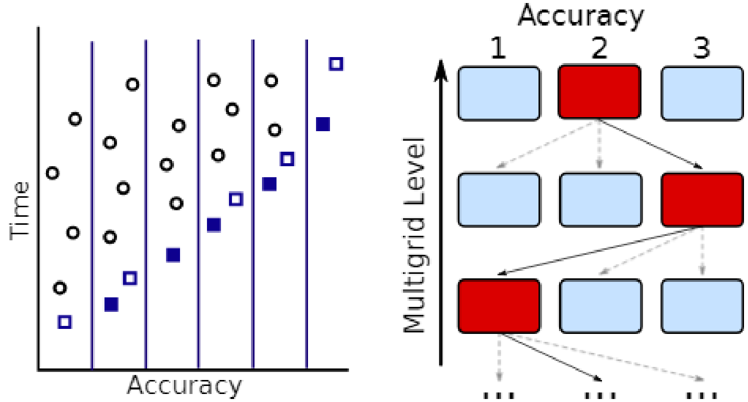
$\mathtt{POISSON}_i(x, b)$

either
  Solve directly
  Iterate using $\mathrm{SOR}_{\omega_{\mathrm{opt}}}$ until accuracy p_i is achieved
  For some j, iterate with $\mathtt{MULTIGRID}_j$ until accuracy p_i is achieved
end either

$\mathtt{MULTIGRID}_i(x, b)$

if $N = 3$ **then**
  Solve directly
else
  Compute one iteration of $\mathrm{SOR}_{1.15}$
  Compute the residual and restrict to half resolution
  On the coarser grid, call $\mathtt{POISSON}_i$
  Interpolate result and add correction term to current solution
  Compute one iteration of $\mathrm{SOR}_{1.15}$
end if

Figure 10. Pseudo code for family of functions $\mathrm{POISSON}_i$ and $\mathrm{MULTIGRID}_i$ where i is the required accuracy, as used in the benchmark.



(a) (b)

PetaBricks Today

- 363 citations according to Google Scholar
- Experiments and benchmarked continued for 4 years looking into tuning variables in algorithms, portability, and study the trade-offs between accuracy and performance.
- Main author(during his PhD), Jason Ansel, Director of engineering at GoDaddy since 2013
- GitHub repo, abandoned 6 years ago
- Ideas we can still use, with help of ML
  - Auto tuning between accuracy and performance
  - Auto tuning variables based on architecture

Critique

The Good	The Bad
First language that explores algorithmic choice with impressive results	Slow compiling time – impractical for software engineering tasks (target success metric)
Ease of selecting the trade-off between accuracy and performance	Complex code structure – harder to adhoc debug
Portability across architectures and future proof	Principles of auto tuning using simple parameter search might be too slow

Comment – Working in a heterogenous environment

SWE commits changes → Changes are built on a remote machine identical to production in a container, auto tuner enabled → Produced binaries makes it way through the testing pipeline until production

- Optimisation could be running a different algorithm choice on a different machine, storing the metadata of binaries in a key-value store, and binaries in a replicated store
- Not every line of code has to be written in PetaBrick, just the lines that require high performance – e.g. a simple microservice that has no complex logic wouldn't benefit from PetaBrick optimisation

Ideas for Future Work – GPUs?

- Extract the idea of choice and analysis per hardware architecture.
- Analyse the benefit of running algorithms on GPUs
- GPUs have different memory constraint than what traditional algorithms were designed for, auto tuning helps!
- Other work in this area already exist using ML for choosing GPU/CPU, but no (afaik) work exist to choose algorithm and tune it

Questions?

UNIVERSITY OF CAMBRIDGE