PetaBricks and Julia

Kathleen C. Alexander

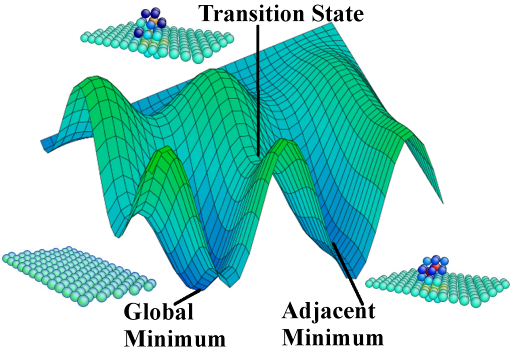
Massachusetts Institute of Technology

December 11th, 2013

Motivation

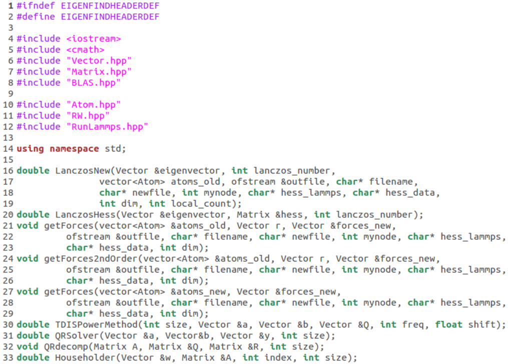
The Programmer's Dilemma

a personal example— energy landscapes



The Programmer's Dilemma

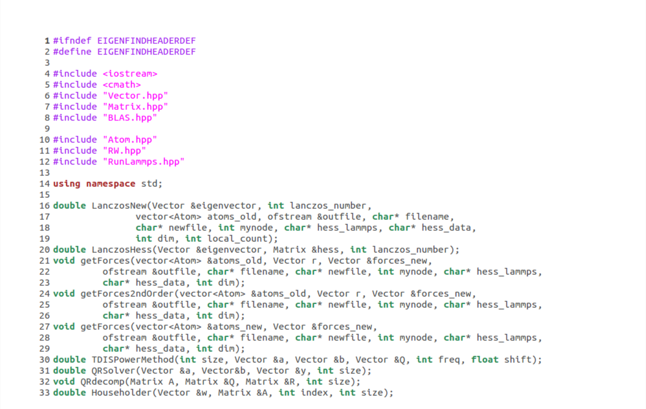
which algorithm is best?



```
#ifndef EIGENFINDHEADERDEF
#define EIGENFINDHEADERDEF

#include <iostream>
#include <cmath>
#include "Vector.hpp"
#include "Matrix.hpp"
#include "BLAS.hpp"

#include "Atom.hpp"
#include "RW.hpp"
#include "RunLammps.hpp"

using namespace std;

double LanczosNew(Vector &eigenvector, int lanczos_number,
               vector<Atom> atoms_old, ofstream &outfile, char* filename,
               char* newfile, int mynode, char* hess_lammps, char* hess_data,
               int dim, int local_count);
double LanczosHess(Vector &eigenvector, Matrix &hess, int lanczos_number);
void getForces(vector<Atom> &atoms_old, Vector r, Vector &forces_new,
         ofstream &outfile, char* filename, char* newfile, int mynode, char* hess_lammps,
         char* hess_data, int dim);
void getForces2ndOrder(vector<Atom> &atoms_old, Vector r, Vector &forces_new,
         ofstream &outfile, char* filename, char* newfile, int mynode, char* hess_lammps,
         char* hess_data, int dim);
void getForces(vector<Atom> &atoms_new, Vector &forces_new,
         ofstream &outfile, char* filename, char* newfile, int mynode, char* hess_lammps,
         char* hess_data, int dim);
double TDISPowerMethod(int size, Vector &a, Vector &b, Vector &Q, int freq, float shift);
double QRSolver(Vector &a, Vector&b, Vector &y, int size);
void QRdecomp(Matrix A, Matrix &Q, Matrix &R, int size);
double Householder(Vector &w, Matrix &A, int index, int size);
```

The Programmer's Dilemma

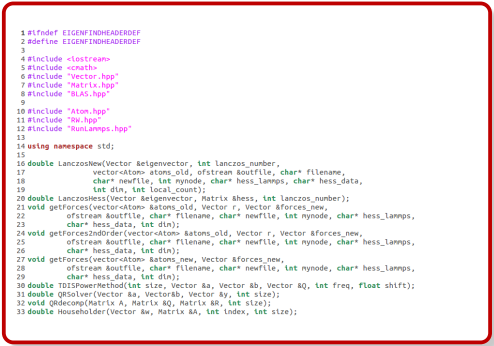
which algorithm is best?



```
#ifndef EIGENFINDHEADERDEF
#define EIGENFINDHEADERDEF

#include <iostream>
#include <cmath>
#include "Vector.hpp"
#include "Matrix.hpp"
#include "BLAS.hpp"

#include "Atom.hpp"
#include "RW.hpp"
#include "RunLammps.hpp"

using namespace std;

double LanczosNew(Vector &eigenvector, int lanczos_number,
               vector<Atom> atoms_old, ofstream &outfile, char* filename,
               char* newfile, int mynode, char* hess_lammps, char* hess_data,
               int dim, int local_count);
double LanczosHess(Vector &eigenvector, Matrix &hess, int lanczos_number);
void getForces(vector<Atom> &atoms_old, Vector r, Vector &forces_new,
         ofstream &outfile, char* filename, char* newfile, int mynode, char* hess_lammps,
         char* hess_data, int dim);
void getForces2ndOrder(vector<Atom> &atoms_old, Vector r, Vector &forces_new,
         ofstream &outfile, char* filename, char* newfile, int mynode, char* hess_lammps,
         char* hess_data, int dim);
void getForces(vector<Atom> &atoms_new, Vector &forces_new,
         ofstream &outfile, char* filename, char* newfile, int mynode, char* hess_lammps,
         char* hess_data, int dim);
double TDISPowerMethod(int size, Vector &a, Vector &b, Vector &Q, int freq, float shift);
double QRSolver(Vector &a, Vector&b, Vector &y, int size);
void QRdecomp(Matrix A, Matrix &Q, Matrix &R, int size);
double Householder(Vector &w, Matrix &A, int index, int size);
```

Goal: determine the best algorithm for the application–
which may be machine dependent

Parallel Programming

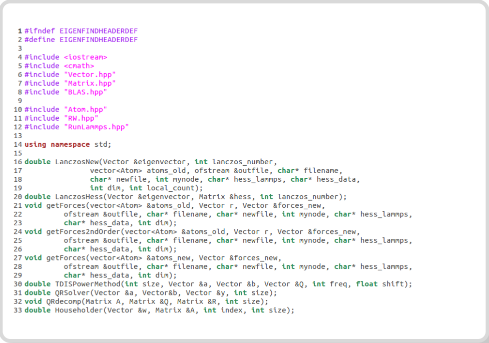


```
#ifndef EIGENFINDHEADERDEF
#define EIGENFINDHEADERDEF

#include <iostream>
#include <cmath>
#include "Vector.hpp"
#include "Matrix.hpp"
#include "BLAS.hpp"

#include "Atom.hpp"
#include "RW.hpp"
#include "RunLammps.hpp"

using namespace std;

double LanczosNew(Vector &eigenvector, int lanczos_number,
               vector<Atom> atoms_old, ofstream &outfile, char* filename,
               char* newfile, int mynode, char* hess_lammps, char* hess_data,
               int dim, int local_count);
double LanczosHess(Vector &eigenvector, Matrix &hess, int lanczos_number);
void getForces(vector<Atom> &atoms_old, Vector r, Vector &forces_new,
         ofstream &outfile, char* filename, char* newfile, int mynode, char* hess_lammps,
         char* hess_data, int dim);
void getForces2ndOrder(vector<Atom> &atoms_old, Vector r, Vector &forces_new,
         ofstream &outfile, char* filename, char* newfile, int mynode, char* hess_lammps,
         char* hess_data, int dim);
void getForces(vector<Atom> &atoms_new, Vector &forces_new,
         ofstream &outfile, char* filename, char* newfile, int mynode, char* hess_lammps,
         char* hess_data, int dim);
double TDISPowerMethod(int size, Vector &a, Vector &b, Vector &Q, int freq, float shift);
double QRSolver(Vector &a, Vector&b, Vector &y, int size);
void QRdecomp(Matrix A, Matrix &Q, Matrix &R, int size);
double Householder(Vector &w, Matrix &A, int index, int size);
```

- many parts of these algorithms can be written in parallel
- often they can be parallelized in many different ways
- optimizing these options is a challenge

Determine the best way to parallelize the program–
which will be machine dependent

Parallel Programming



```
#ifndef EIGENFINDHEADERDEF
#define EIGENFINDHEADERDEF

#include <iostream>
#include <cmath>
#include "Vector.hpp"
#include "Matrix.hpp"
#include "BLAS.hpp"

#include "Atom.hpp"
#include "RW.hpp"
#include "RunLammps.hpp"

using namespace std;

double LanczosNew(Vector &eigenvector, int lanczos_number,
               vector<Atom> atoms_old, ofstream &outfile, char* filename,
               char* newfile, int mynode, char* hess_lammps, char* hess_data,
               int dim, int local_count);
double LanczosHess(Vector &eigenvector, Matrix &hess, int lanczos_number);
void getForces(vector<Atom> &atoms_old, Vector r, Vector &forces_new,
         ofstream &outfile, char* filename, char* newfile, int mynode, char* hess_lammps,
         char* hess_data, int dim);
void getForces2ndOrder(vector<Atom> &atoms_old, Vector r, Vector &forces_new,
         ofstream &outfile, char* filename, char* newfile, int mynode, char* hess_lammps,
         char* hess_data, int dim);
void getForces(vector<Atom> &atoms_new, Vector &forces_new,
         ofstream &outfile, char* filename, char* newfile, int mynode, char* hess_lammps,
         char* hess_data, int dim);
double TDISPowerMethod(int size, Vector &a, Vector &b, Vector &Q, int freq, float shift);
double QRSolver(Vector &a, Vector&b, Vector &y, int size);
void QRdecomp(Matrix A, Matrix &Q, Matrix &R, int size);
double Householder(Vector &w, Matrix &A, int index, int size);
```

- many parts of these algorithms can be written in parallel
- often they can be parallelized in many different ways
- optimizing these options is a challenge

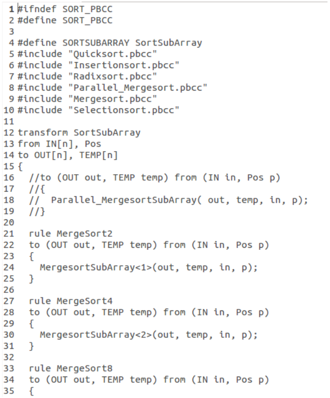
Determine the best way to parallelize the program–
which will be machine dependent

Background

Petabricks – Algorithmic Choice

PetaBricks was developed to alleviate some of the optimization responsibility from the programmer

the transform

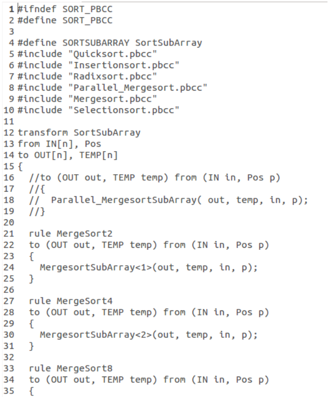


```
#ifndef SORT_PBCC
#define SORT_PBCC

#define SORTSUBARRAY SortSubArray
#include "Quicksort.pbcc"
#include "Insertionsort.pbcc"
#include "Radixsort.pbcc"
#include "Parallel_Mergesort.pbcc"
#include "Mergesort.pbcc"
#include "Selectionsort.pbcc"

transform SortSubArray
from IN[n], Pos
to OUT[n], TEMP[n]
{
  //to (OUT out, TEMP temp) from (IN in, Pos p)
  //{
  //  Parallel_MergesortSubArray( out, temp, in, p);
  //}

  rule MergeSort2
  to (OUT out, TEMP temp) from (IN in, Pos p)
  {
    MergesortSubArray<1>(out, temp, in, p);
  }

  rule MergeSort4
  to (OUT out, TEMP temp) from (IN in, Pos p)
  {
    MergesortSubArray<2>(out, temp, in, p);
  }

  rule MergeSort8
  to (OUT out, TEMP temp) from (IN in, Pos p)
  {
```

Petabricks – Algorithmic Choice

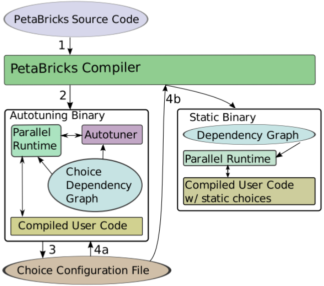
PetaBricks was developed to alleviate some of the optimization responsibility from the programmer

the transform



```
#ifndef SORT_PBCC
#define SORT_PBCC

#define SORTSUBARRAY SortSubArray
#include "Quicksort.pbcc"
#include "Insertionsort.pbcc"
#include "Radixsort.pbcc"
#include "Parallel_Mergesort.pbcc"
#include "Mergesort.pbcc"
#include "Selectionsort.pbcc"

transform SortSubArray
from IN[n], Pos
to OUT[n], TEMP[n]
{
  //to (OUT out, TEMP temp) from (IN in, Pos p)
  //{
  //  Parallel_MergesortSubArray( out, temp, in, p);
  //}

  rule MergeSort2
  to (OUT out, TEMP temp) from (IN in, Pos p)
  {
    MergesortSubArray<1>(out, temp, in, p);
  }

  rule MergeSort4
  to (OUT out, TEMP temp) from (IN in, Pos p)
  {
    MergesortSubArray<2>(out, temp, in, p);
  }

  rule MergeSort8
  to (OUT out, TEMP temp) from (IN in, Pos p)
  {
```

compiling framework

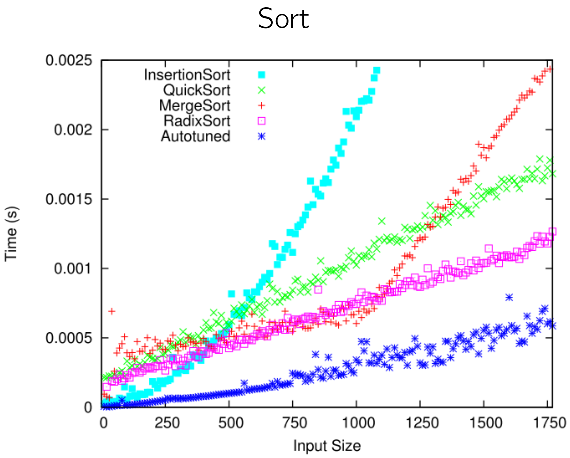


Ansel, et al. ACM SIGPLAN Conference (2009).

Petabricks – Autotuning

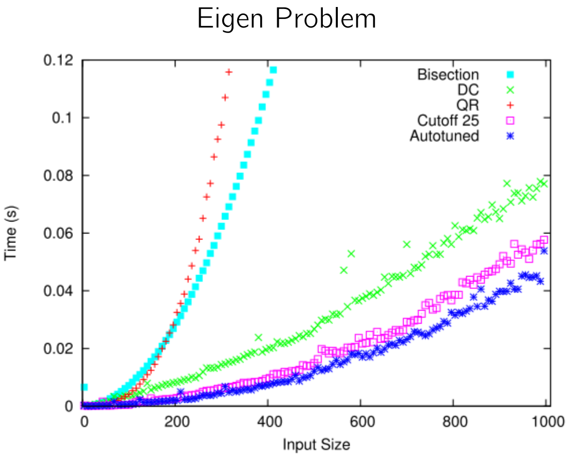
the autotuner determines the best configuration for the machine under the tuning constraints

Petabricks – Autotuning



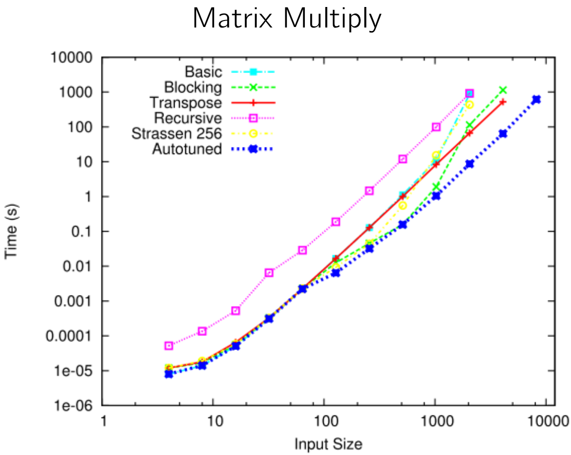
Ansel, et al. ACM SIGPLAN Conference (2009).

Petabricks – Autotuning



Ansel, et al. ACM SIGPLAN Conference (2009).

Petabricks – Autotuning

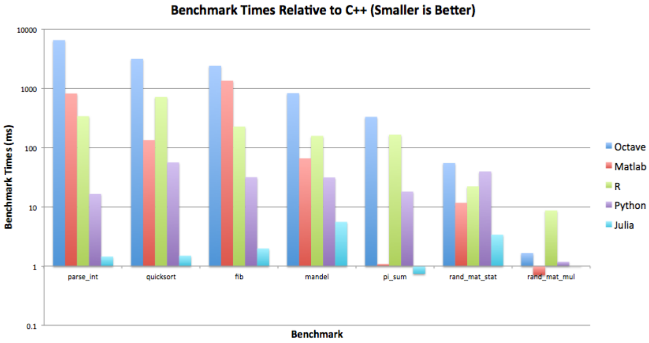


Ansel, et al. ACM SIGPLAN Conference (2009).

Julia

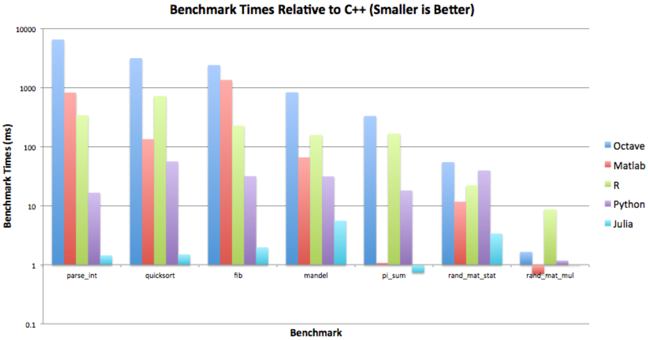
- Julia was developed to bridge the gap between interpreted and compiled scientific computing
- streamlining parallelization techniques has been a priority

Julia



http://forio.com/julia/julia

Julia



http://forio.com/julia/julia

Question: is there room for overlap between the PetaBricks and Julia approaches?

Approach

Options for Implementation

Julia in PetaBricks

- can utilize PetaBricks autotuner and compiler
- PetaBricks compiler needs to interpret Julia

Options for Implementation

Julia in PetaBricks

- can utilize PetaBricks autotuner and compiler
- PetaBricks compiler needs to interpret Julia

PetaBricks in Julia

- can run PetaBricks binaries inside Julia
- no PetaBricks shared object files, functions require disk i/o
- doesn't take advantage of JuliaLang

Options for Implementation

Julia in PetaBricks

- can utilize PetaBricks autotuner and compiler
- PetaBricks compiler needs to interpret Julia

PetaBricks in Julia

- can run PetaBricks binaries inside Julia
- no PetaBricks shared object files, functions require disk i/o
- doesn't take advantage of JuliaLang

Julia + OpenTuner

- apply PetaBricks framework to Julia
- utilize OpenTuner to optimize Julia

Approach Used Here

PetaBricks in Julia

- can run PetaBricks binaries inside Julia
- no PetaBricks shared object files, functions require disk i/o
- doesn't take advantage of JuliaLang

Approach Used Here

PetaBricks in Julia

- can run PetaBricks binaries inside Julia
- no PetaBricks shared object files, functions require disk i/o
- doesn't take advantage of JuliaLang

⇒ most naive approach possible:

→ compile PetaBricks executable, exe

→ `julia > run(‘$exe $in $out‘)`

Approach Used Here

PetaBricks in Julia

- can run PetaBricks binaries inside Julia
- no PetaBricks shared object files, functions require disk i/o
- doesn't take advantage of JuliaLang

⇒ most naive approach possible:

→ compile PetaBricks executable, exe

→ `julia > run(‘$exe $in $out‘)`

⇒ compare with PetaBricks and Julia alone

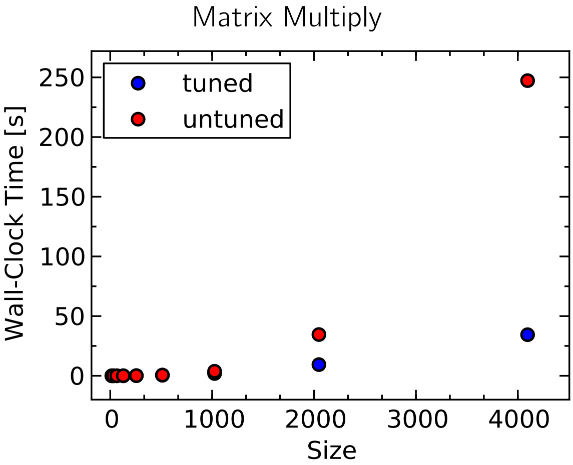
→ lower bound of performance improvement

→ is there proof of benefit?

Results

PetaBricks- Tuning Improvements

performance improvement— tuned and untuned PetaBricks



Comparing PetaBricks with Julia - Apples to Apples

PetaBricks

→ functions read in ASCII files and output same

→ determines parallelization during autotuning

→ autotuning can take days

Julia

→ JIT for each independent execution

→ can addprocs(n), but may not parallelize

→ can be used interactively

Comparing PetaBricks with Julia - Apples to Apples

PetaBricks

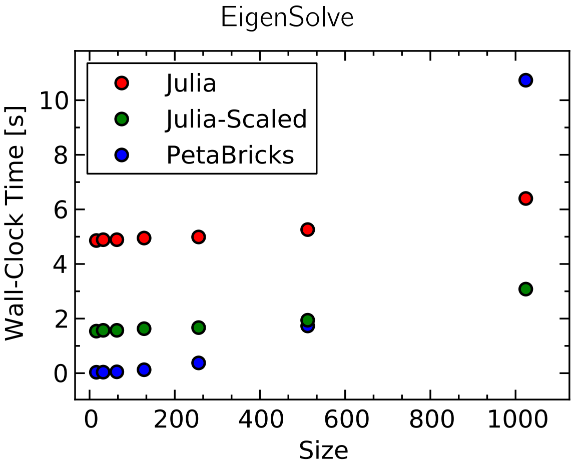
- → functions read in ASCII files and output same
- → determines parallelization during autotuning
- → autotuning can take days

PetaBricks

- → JIT for each independent executable
- → can addprocs(n), but may not parallelize
- → can be used interactively

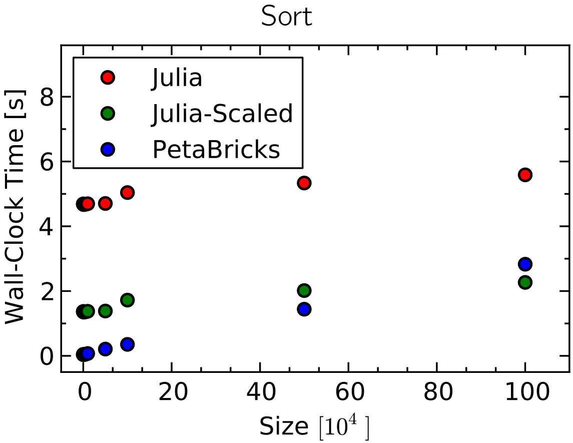
- → make both programs do i/o
- → run both programs from shell
- → try addprocs(n) in Julia, with no other instructions
- → subtract 'hello world' start-up time from Julia wall-clock

Comparing PetaBricks to Julia - EigenSolve



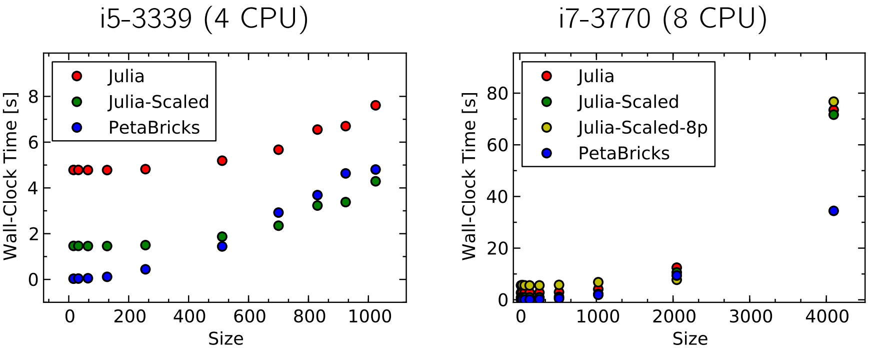
- → Julia seems to do the best for large matrices
- → however, the results were not comparable
- → this test was not a good apples-to-apples performance test

Comparing PetaBricks with Julia - Sort



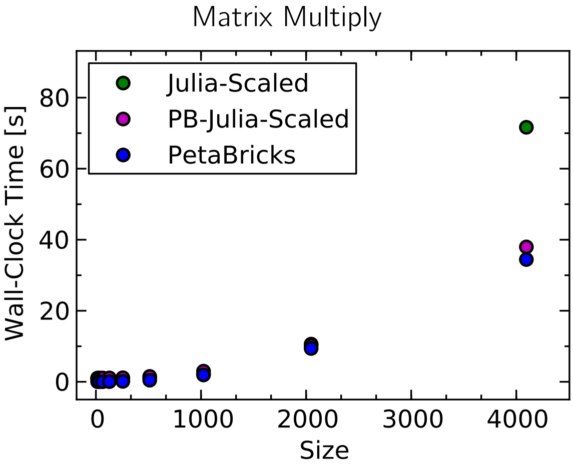
- → Julia and PetBricks converge for large vectors
- → PetaBricks is better with shorter vectors
- → effect of i/o not considered wrt performance

Comparing PetaBricks with Julia Matrix Multiply



- → Julia and PetBricks converge moderate matrix sizes on fewer cores
- → PetaBricks is better with smaller lists and larger matrices
- → using addprocs(n) with no other instruction does not utilize parallel functionality in Julia

Running PetaBricks from Julia

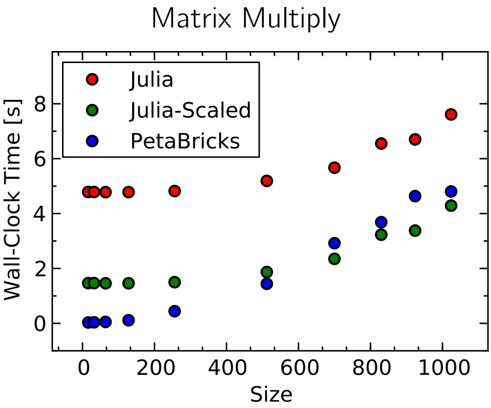


→ Can get PetaBricks improvement by incorporating PetaBricks executable in Julia

→ effect of i/o not considered wrt performance

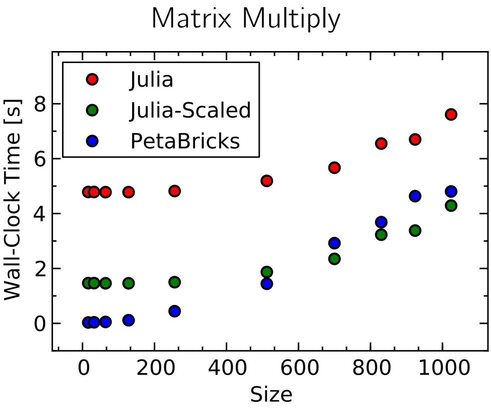
Recommendations

Recommendations



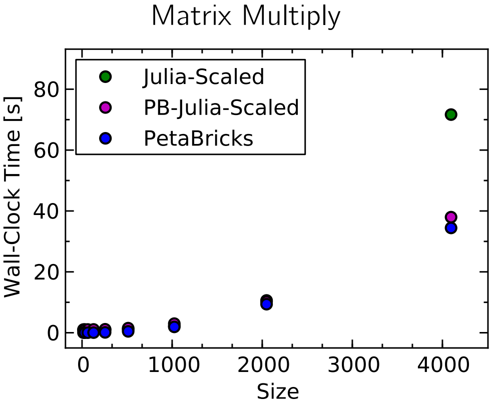
→ under many circumstances, Julia performs as well as PetaBricks without days of compilation

Recommendations



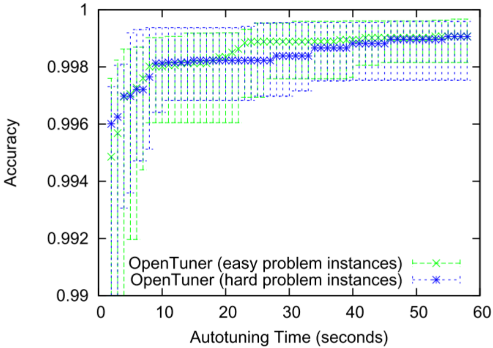
→ there is room for improvement on the start-up time for Julia

Recommendations



→ PetaBricks performance can be achieved by using a shell command in Julia

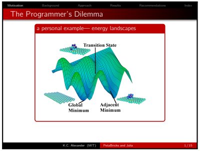
Recommendations



Ansel, et. al. MIT CSAIL Technical Report MIT-CSAIL-TR-2013-026 (2013).

→ implementing OpenTuner (when better documentation is available) with Julia may be a reasonable long term goal for performance gains of this kind

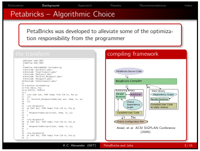
Index I



The Programmer's Dilemma

a personal example— energy landscapes

1



Petabricks – Algorithmic Choice

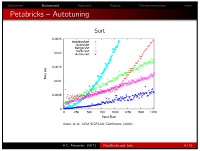
PetaBricks was developed to alleviate some of the optimization responsibility from the programmer

the transform

compiling framework

Ansel, et al. ACM SIGPLAN Conference (2009).

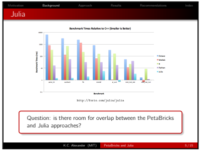
2



Petabricks – Autotuning

Ansel, et al. ACM SIGPLAN Conference (2009).

3

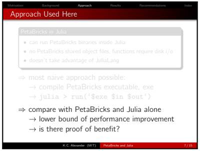


Julia

http://forio.com/julia/julia

Question: is there room for overlap between the PetaBricks and Julia approaches?

4



Approach Used Here

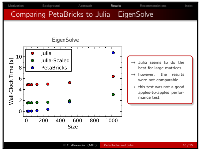
PetaBricks in Julia

- can run PetaBricks binaries inside Julia
- no PetaBricks shared object files, functions require disk i/o
- doesn't take advantage of JuliaLang

⇒ most naive approach possible:
→ compile PetaBricks executable, exe
→ `julia > run('$exe $in $out')`

⇒ compare with PetaBricks and Julia alone
→ lower bound of performance improvement
→ is there proof of benefit?

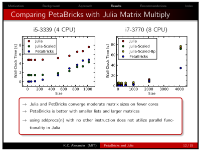
5



Comparing PetaBricks to Julia - EigenSolve

→ Julia seems to do the best for large matrices
→ however, the results were not comparable
→ this test was not a good apples-to-apples performance test

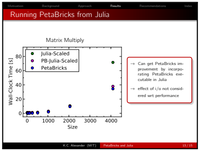
6



Comparing PetaBricks with Julia Matrix Multiply

→ Julia and PetBricks converge moderate matrix sizes on fewer cores
→ PetaBricks is better with smaller lists and larger matrices
→ using addprocs(n) with no other instruction does not utilize parallel functionality in Julia

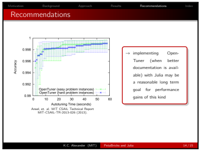
7



Running PetaBricks from Julia

→ Can get PetaBricks improvement by incorporating PetaBricks executable in Julia
→ effect of i/o not considered wrt performance

8



Recommendations

Accuracy
OpenTuner (easy problem instances)
OpenTuner (hard problem instances)
Autotuning Time (seconds)

Ansel, et. al. MIT CSAIL Technical Report MIT-CSAIL-TR-2013-026 (2013).

→ implementing OpenTuner (when better documentation is available) with Julia may be a reasonable long term goal for performance gains of this kind

9