PetaBricks and Julia

Kathleen C. Alexander

Massachusetts Institute of Technology

December 11th, 2013
The Programmer’s Dilemma

*a personal example— energy landscapes*

[Diagram showing energy landscapes with labels for Transition State, Global Minimum, and Adjacent Minimum]
The Programmer’s Dilemma

which algorithm is best?

```cpp
#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 &x, Matrix &Q, int index, int size);
```
Goal: determine the best algorithm for the application—which may be machine dependent
Parallel Programming

- 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
• 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

```plaintext
1#ifndef SORT_PBCC
2#define SORT_PBCC
3
4#define SORTSUBARRAY SortSubArray
5#include "Quicksort.pbcc"
6#include "Insertionsort.pbcc"
7#include "Radixsort.pbcc"
8#include "Parallel_Mergesort.pbcc"
9#include "Mergesort.pbcc"
10#include "Selectionsort.pbcc"
11
12transform SortSubArray
13from IN[n], Pos
14to OUT[n], TEMP[n]
15{
16   //to (OUT out, TEMP temp) from (IN in, Pos p)
17   //{}
18   // Parallel_MergesortSubArray( out, temp, ln, p);
19   //}
20
21rule MergeSort2
22to (OUT out, TEMP temp) from (IN ln, Pos p)
23{
24   MergesortSubArray<1>(out, temp, ln, p);
25 }
26
27rule MergeSort4
28to (OUT out, TEMP temp) from (IN ln, Pos p)
29{
30   MergesortSubArray<2>(out, temp, ln, p);
31 }
32
33rule MergeSort8
34to (OUT out, TEMP temp) from (IN ln, Pos p)
35{
```
PetaBricks was developed to alleviate some of the optimization responsibility from the programmer.

```
1#define SORT_PBCC
2#define SORT_PBCC
3#define SORTSUBARRAY SortSubArray
4#include "Quicksort.pbcc"
5#include "Insertionsort.pbcc"
6#include "Radixsort.pbcc"
7#include "Parallel_Mergesort.pbcc"
8#include "Mergesort.pbcc"
9#include "Selectionsort.pbcc"
10
11transform SortSubArray
12from IN[n], Pos
13to OUT[n], TEMP[n]
14{
15  //to (OUT out, TEMP temp) from (IN in, Pos p)
16  // Parallel_MergesortSubArray( out, temp, in, p);
17  //}
18  rule MergeSort2
19  to (OUT out, TEMP temp) from (IN in, Pos p)
20  {
21    MergesortSubArray<1>(out, temp, in, p);
22  }
23  rule MergeSort4
24  to (OUT out, TEMP temp) from (IN in, Pos p)
25  {
26    MergesortSubArray<2>(out, temp, in, p);
27  }
28  rule MergeSort8
29  to (OUT out, TEMP temp) from (IN in, Pos p)
30  {
31    
32  }
```
the autotuner determines the best configuration for the machine under the tuning constraints
Eigen Problem

Matrix Multiply

• Julia was developed to bridge the gap between interpreted and compiled scientific computing
• streamlining parallelization techniques has been a priority
http://forio.com/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

Matrix Multiply

Wall-Clock Time [s]

Size

K.C. Alexander (MIT)
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

<table>
<thead>
<tr>
<th>PetaBricks</th>
<th></th>
<th>PetaBricks</th>
</tr>
</thead>
<tbody>
<tr>
<td>→ functions read in ASCII files and output same</td>
<td>→ JIT for each independent executable</td>
<td></td>
</tr>
<tr>
<td>→ determines parallelization during autotuning</td>
<td>→ can addprocs(n), but may not parallelize</td>
<td></td>
</tr>
<tr>
<td>→ autotuning can take days</td>
<td>→ can be used interactively</td>
<td></td>
</tr>
</tbody>
</table>

→ 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

Matrix Multiply

- Julia-Scaled
- PB-Julia-Scaled
- PetaBricks

→ Can get PetaBricks improvement by incorporating PetaBricks executable in Julia

→ Effect of i/o not considered wrt performance
Recommendations
→ under many circumstances, Julia performs as well as PetaBricks without days of compilation.
there is room for improvement on the startup time for Julia
Matrix Multiply

Julia-Scaled
PB-Julia-Scaled
PetaBricks

→ PetaBricks performance can be achieved by using a shell command in Julia
implementing Open-Tuner (when better documentation is available) with Julia may be a reasonable long term goal for performance gains of this kind.

Running PetaBricks from Julia

Comparing PetaBricks with Julia Matrix Multiply

- Julia
- Julia-Scaled
- Julia-Scaled-8p
- PetaBricks

Wall-Clock Time [s]

Size

0 20 40 60 80

0 1000 2000 3000 4000

Results

PetaBricks – Algorithmic Choice

PetaBricks – Autotuning

Petabricks – Tuner (when better documentation is available)


K.C. Alexander (MIT)