Plan

• Audience and Problem
• Context and Relevance
• Proposed Solution
  Parallel Programming Features
• Support for Conclusions
• Future Research
• Focus Questions for CS 620
Audience and Problem

• scientific programming community
• trade-off between productivity and performance
Context and Relevance

<table>
<thead>
<tr>
<th>Language</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td>1957</td>
</tr>
<tr>
<td>C</td>
<td>1973</td>
</tr>
<tr>
<td>Matlab</td>
<td>Late 1970s</td>
</tr>
<tr>
<td>Python</td>
<td>1989</td>
</tr>
<tr>
<td>R</td>
<td>1993</td>
</tr>
</tbody>
</table>
## Context and Relevance

<table>
<thead>
<tr>
<th></th>
<th>Fortran</th>
<th>Python</th>
<th>R</th>
<th>Matlab</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc 5.1.1</td>
<td>0.70</td>
<td>77.76</td>
<td>533.52</td>
<td>26.89</td>
</tr>
<tr>
<td>parse_int</td>
<td>5.05</td>
<td>17.02</td>
<td>45.73</td>
<td>802.52</td>
</tr>
<tr>
<td>quicksort</td>
<td>1.31</td>
<td>32.89</td>
<td>264.54</td>
<td>4.92</td>
</tr>
<tr>
<td>mandel</td>
<td>0.81</td>
<td>15.32</td>
<td>53.16</td>
<td>7.58</td>
</tr>
<tr>
<td>pi_sum</td>
<td>1.00</td>
<td>21.99</td>
<td>9.56</td>
<td>1.00</td>
</tr>
<tr>
<td>rand_mat_stat</td>
<td>1.45</td>
<td>17.93</td>
<td>14.56</td>
<td>14.52</td>
</tr>
<tr>
<td>rand_mat_mul</td>
<td>3.48</td>
<td>1.14</td>
<td>1.57</td>
<td>1.12</td>
</tr>
</tbody>
</table>

**Figure:** benchmark times relative to C

http://www.julialang.org/
Proposed Solution

- JIT compilation (AST → LLVM IR)
- run-time type inference
- multiple dispatch
- “building blocks” for parallel programming
Parallel Programming Features

- message passing, remote calls
- “lazy” parallelism: @spawn and @parallel
- fork and join
- dynamic scheduling
- shared memory
- distributed processing
Support for Conclusions

- Ease-of-use
- Performance
- Multiple options for parallel programming
Ease-of-use

```plaintext
function randmatstat(t)
    n = 5
    v = zeros(t)
    w = zeros(t)
    for i = 1:t
        a = randn(n,n)
        b = randn(n,n)
        c = randn(n,n)
        d = randn(n,n)
        P = [a b c d]
        Q = [a b; c d]
        v[i] = trace((P.'*P)^4)
        w[i] = trace((Q.'*Q)^4)
    end
    std(v)/mean(v), std(w)/mean(w)
end
```

http://www.julialang.org/
Performance

<table>
<thead>
<tr>
<th></th>
<th>Fortran gcc 5.1.1</th>
<th>Julia 0.4.0</th>
<th>Python 3.4.3</th>
<th>R 3.2.2</th>
<th>Matlab R2015b</th>
</tr>
</thead>
<tbody>
<tr>
<td>fib</td>
<td>0.70</td>
<td>2.11</td>
<td>77.76</td>
<td>533.52</td>
<td>26.89</td>
</tr>
<tr>
<td>parse_int</td>
<td>5.05</td>
<td>1.45</td>
<td>17.02</td>
<td>45.73</td>
<td>802.52</td>
</tr>
<tr>
<td>quicksort</td>
<td>1.31</td>
<td>1.15</td>
<td>32.89</td>
<td>264.54</td>
<td>4.92</td>
</tr>
<tr>
<td>mandel</td>
<td>0.81</td>
<td>0.79</td>
<td>15.32</td>
<td>53.16</td>
<td>7.58</td>
</tr>
<tr>
<td>pi_sum</td>
<td>1.00</td>
<td>1.00</td>
<td>21.99</td>
<td>9.56</td>
<td>1.00</td>
</tr>
<tr>
<td>rand_mat_stat</td>
<td>1.45</td>
<td>1.66</td>
<td>17.93</td>
<td>14.56</td>
<td>14.52</td>
</tr>
<tr>
<td>rand_mat_mul</td>
<td>3.48</td>
<td>1.02</td>
<td>1.14</td>
<td>1.57</td>
<td>1.12</td>
</tr>
</tbody>
</table>

**Figure**: benchmark times relative to C

http://www.julialang.org/
Parallel Options

- `remotecall()` vs `@spawn`
- `multiple @spawn calls` vs `@parallel`
- `SharedArray` vs `@parallel`
Parallel Options: Advection
Parallel Options: Advection
Parallel Options: Advection
Parallel Options: Advection
Parallel Options: Advection
Parallel Options: Advection
Parallel Options: SharedArray

<table>
<thead>
<tr>
<th>process 2</th>
<th>process 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Grid" /></td>
<td><img src="image2" alt="Grid" /></td>
</tr>
<tr>
<td><img src="image3" alt="Grid" /></td>
<td><img src="image4" alt="Grid" /></td>
</tr>
</tbody>
</table>
Parallel Options: @parallel for

process 2
process 3
Parallel Options: @parallel for

What kinds of overhead?
Future Research

- parallel programming performance
- realistic (application-based) benchmarks
- productivity metrics
Focus Questions

- key features of a parallel programming model?
- impact of programming constructs on design?
- evaluation of parallel programming models?
Key Features

- remote calls directed from a single process
- flexible data movement / sharing
- intuitive syntax for simple parallel actions
- dynamic scheduling
- customizable distributed programming
- variety of options
Constructs Influence Design

- intuitive syntax $\rightarrow$ algorithm transparency
- multiple dispatch $\rightarrow$ code closer to ```math-ese```
- several parallel options $\rightarrow$ match model to situation
Evaluation of Models

- productivity
- performance
- flexibility
References

*The Julia Language*

http://www.julialang.org/

Retrieved Sunday, February 21 at 11:16pm
Your Thoughts

• Key Features of a Parallel Programming Model
• Impact of Programming Constructs on Design
• Evaluation of Parallel Programming Models
• Other thoughts