Obtaining Performance from a Julia-Implementation of Trilinos Data Libraries

Neil Lindquist, Mike Heroux

Saint John's University, Minnesota

SIAM Conference on Computational Science and Engineering
February 27th 2019

Julia

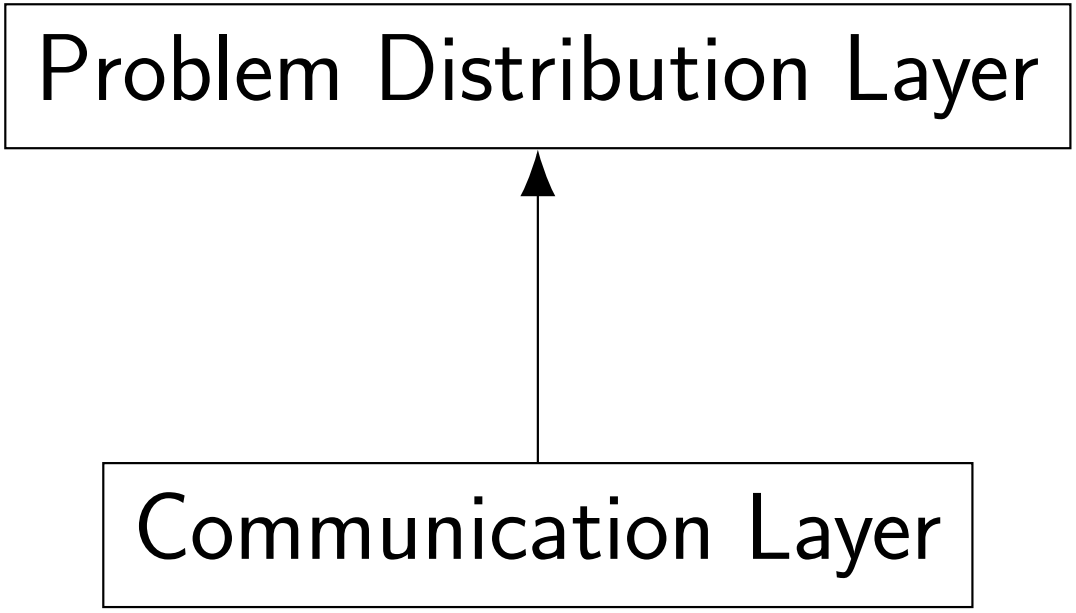
- High Level
- Compiles to efficient machine code

Petra Object Model

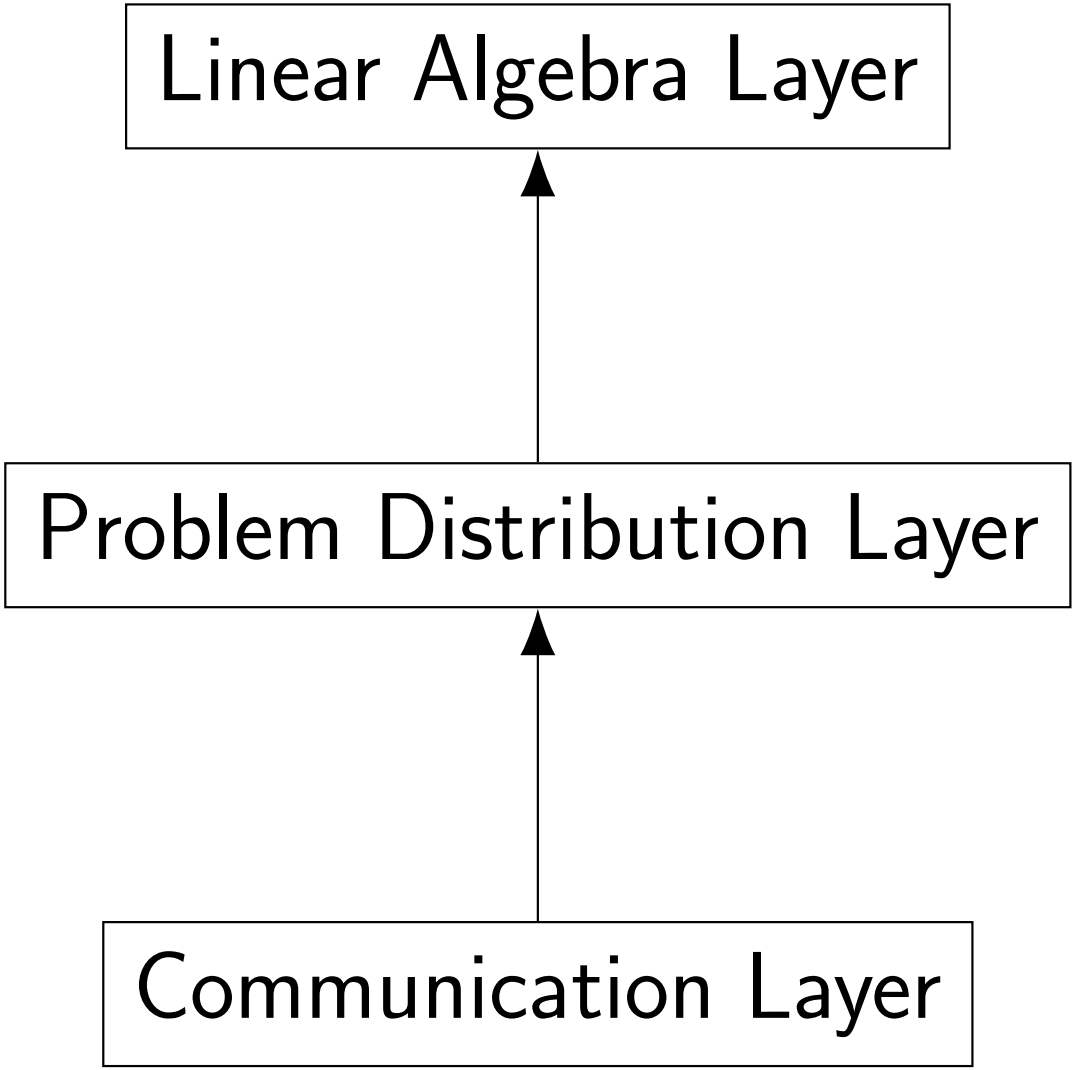
- Underlying data libraries for Trilinos
- Family of sparse linear algebra frameworks
- 3 existing implementations

Petra Organization

Petra Organization



Petra Organization



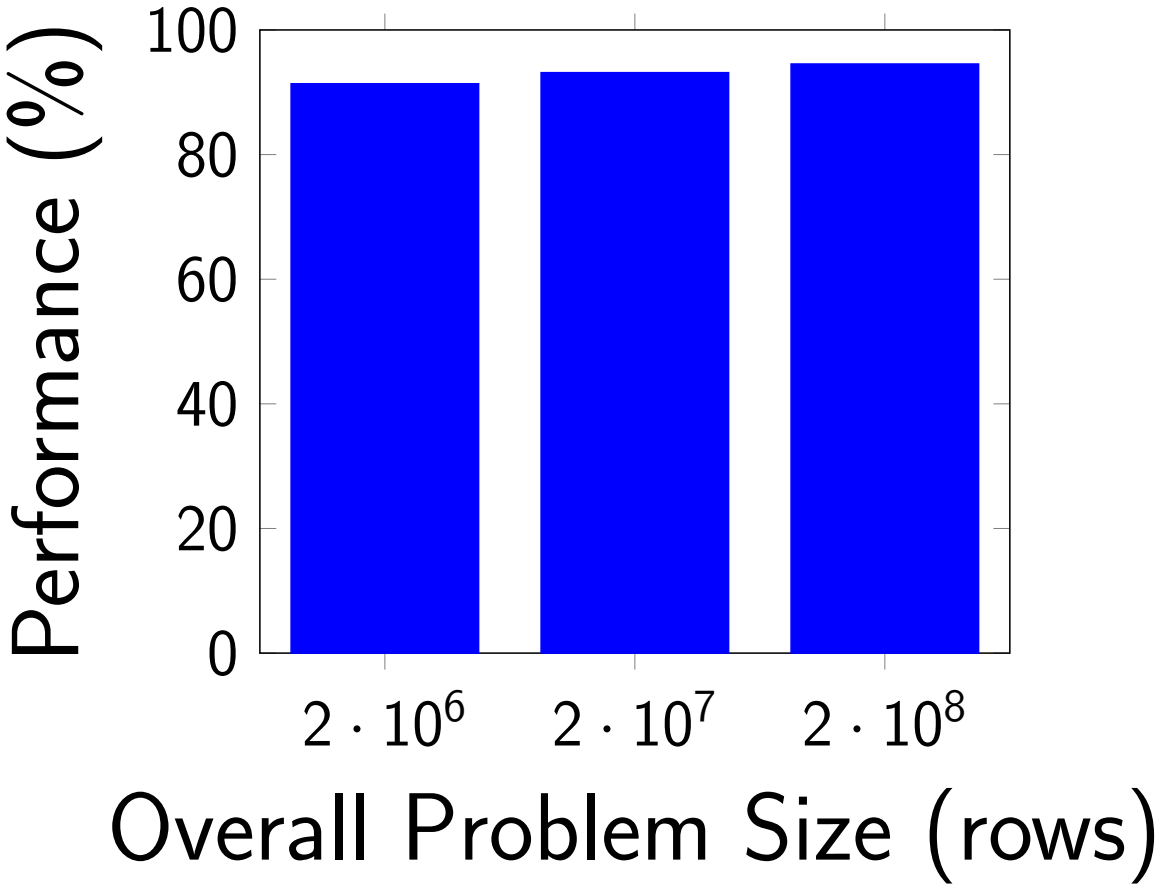
JuliaPetra Example I

```
1  function powerMethod(A::RowMatrix{Data},
2           tol::Data) where Data
3      blockmap = getRowMap(A)
4      q = DenseMultiVector{Data}(blockmap, 1)
5      z = DenseMultiVector{Data}(blockmap, 1)
6      randn!(z.data)
7      r = DenseMultiVector{Data}(blockmap, 1)
```

JuliaPetra Example II

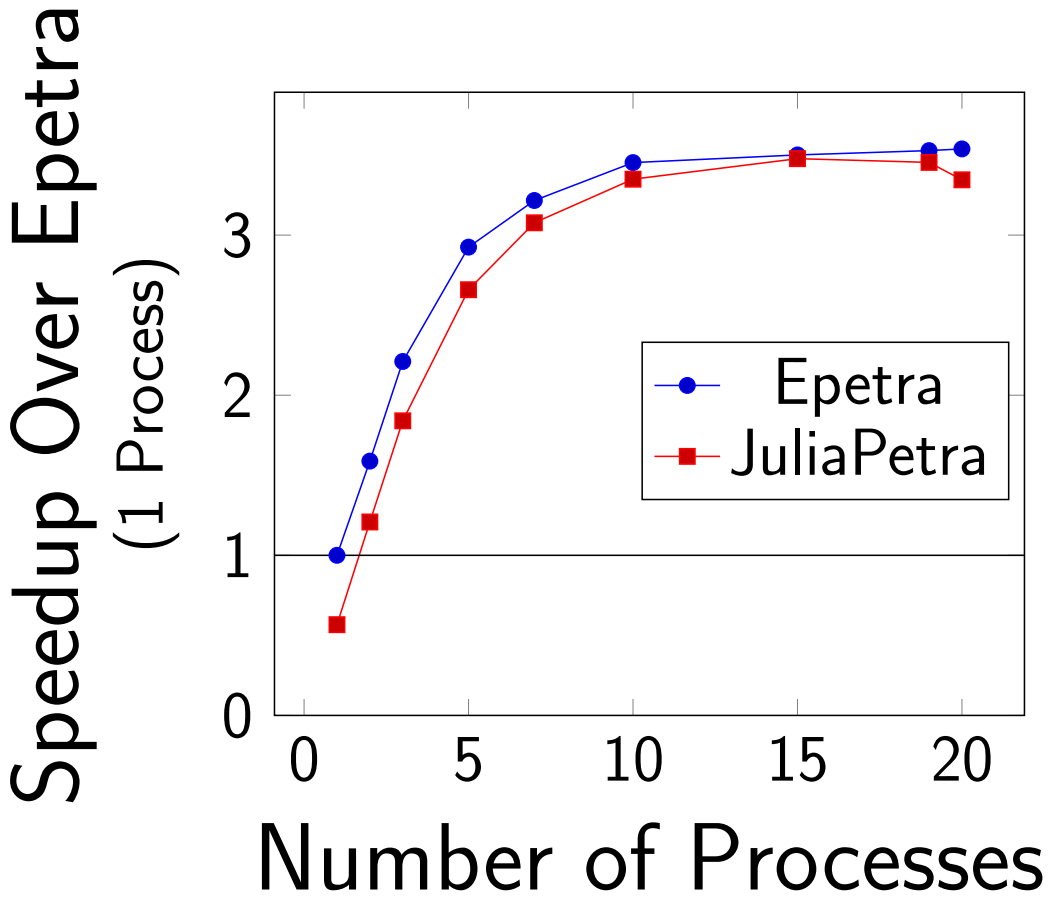
```
 8       while true
 9           normz = norm(z, 2)
10           @. q = z/normz
11           apply!(z, A, q)
12           λ = q⋅z # = dot(q, z)
13           @. r = z - λ*q
14           if norm(r, 2)[1] < tol
15               return λ
16           end
17       end
18   end
```

Performance Comparison



20 CPU processes on a 20 core node.

Performance Comparison



10^7 rows on a 20 core node.

Optimizing JuliaPetra

- Requires disabling high level features
  - Dynamic Typing
  - Garbage Collection
  - Bounds Checks

Type Stability

Optimizing JuliaPetra

- type annotations are optional
- types inferred $\Longrightarrow$ "Type Stable"
- allows inlining

Type Stability Tools

Optimizing JuliaPetra

- `code_warntype` - view inferred types
- TypeStability.jl - Automated checking

Reducing Garbage Collection

Optimizing JuliaPetra

- Garbage collection is automatic
- `Ptr` type doesn't need garbage collection

Bounds Checks

Optimizing JuliaPetra

- Julia checks bounds automatically
- `@inbounds` macro skips bounds checks

Conclusion

- promising, high level language
- clean APIs
- high performance

More Information

Github.com/Collegeville/JuliaPetra.jl

References I

[1] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, *Julia: A Fresh Approach to Numerical Computing*, SIAM Review, 59 (2017), pp. 65–98.

[2] M. Gu, *Single- and Multiple-Vector Iterations (Section 4.3)*, in Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide, Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

References II

[3] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, *An Overview of the Trilinos Project*, ACM Trans. Math. Softw., 31 (2005), pp. 397–423.

References III

[4] N. Lindquist, *TypeStability.jl*, , Retrieved 2018-08-21 from https://github.com/collegeville/TypeStability.jl.

[5] A. Noack, J. Bolewski, K. Fischer, T. Besard, S. Verweij, T. Mohapatra, V. Churavy, and V. B. Shah, *DistributedArrays.jl*, , Retrieved 2018-02-23 from https://github.com/JuliaParallel/DistributedArrays.jl.