SPLARF

Series and Parallel Linear Algebra Routines and Functions

Sander Burton *sanderburton@gmail.com*
Max Clark *clark.max.a@gmail.com*
James Sanford-Luevano *sanford.james.m@gmail.com*
Jaxton Winder *jaxton.winder@gmail.com*

April 18th, 2022

UtahStateUniversity

Abstract

SPLARF is a C++ library of linear algebra routines optimized for speed and efficiency, created to research parallel and serial implementations and provide an interface for commonly used routines.

- The following fields benefit from optimized computational LA:
  - Large data scientific computation
  - Computer graphics
  - Machine learning
  - Physics simulation
  - Commercial applications
  - Computer vision

Goals

- Parallelize linear algebra routines
- Optimize algorithms for parallel application
- Analyze trends
- Compare results
- Learn more about linear algebra, parallel programming, and collaborative work!

Previous Work

- Implementing linear algebra in parallel can speed up the computation time of large matrix/vector operations, and repeated matrix/vector operations (we found several studies with this objective in mind)
- Hypercube topologies for communication
- Parallel Techniques used in previous work (and our project)
  - Manager/Worker relationship
  - Equal partition approach
  - Load balancing
- Numpy (LAPACK)

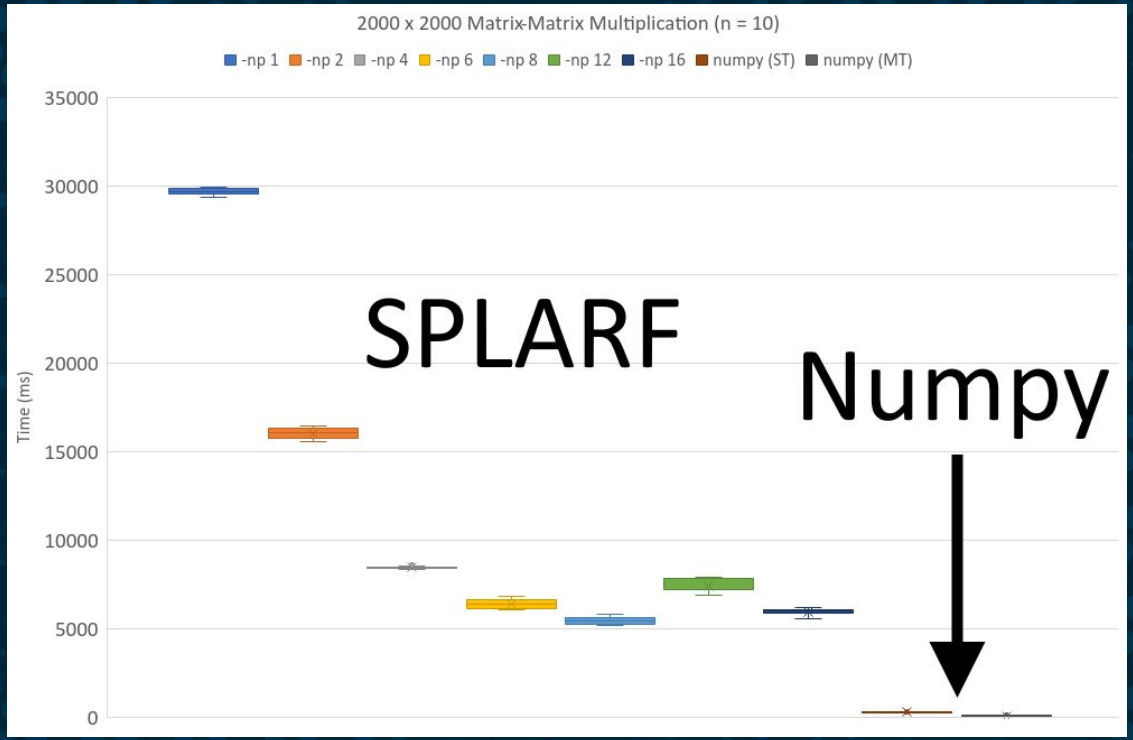
Project Structure

- Jira - Project management, dividing work, and generating charts
- Github - repository for source control
- Source code architecture - each routine or group of closely related routines has its own directory within the src/ directory, containing one or more .cpp files and header files.
- Test architecture - test/ and performance-testing/ directories. Each containing standalone .cpp files to test accuracy and speed of the library.
- To use the library, include it in your project and import the .h files containing your desired functions.

Functions Implemented

- LU factorization
- Diagonal And Triangle Ax=b solvers
- Matrix addition/subtraction
- Matrix element-wise product
- Matrix-matrix product
- Matrix-vector product
- Matrix transpose
- Matrix scalar arithmetic
- Vector addition/subtraction
- Vector element-wise product
- Vector dot product
- Vector scalar arithmetic
- Random matrix and vector generators
- And more!

Results - Matrix/Matrix Multiplication

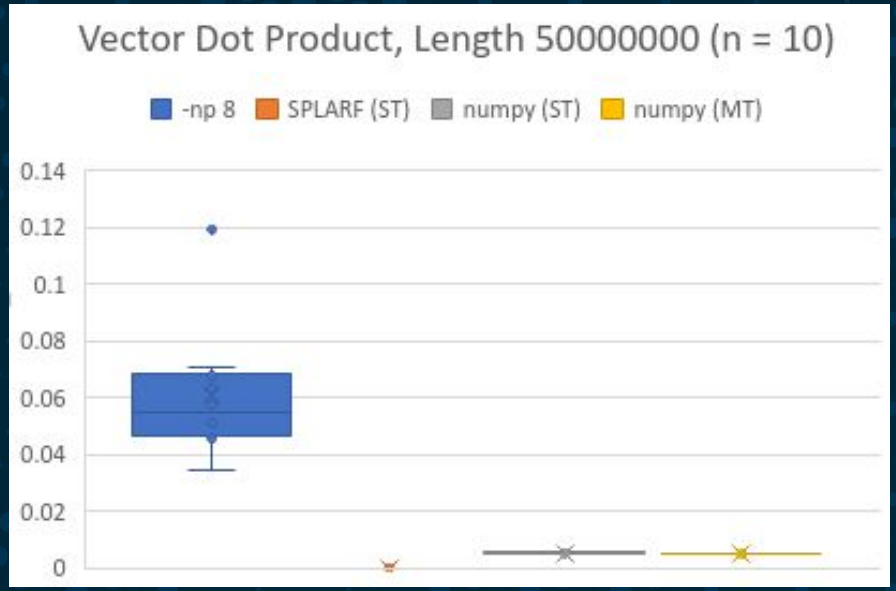


Results - Matrix/Matrix Multiplication

- Numpy's source code was reviewed
- numpy.matmul led to LAPACK
- LAPACK is an open source Linear Algebra library that has been developed and optimized since 1992
- LAPACK performs many optimizations, including checking different functions for non-square multiplication

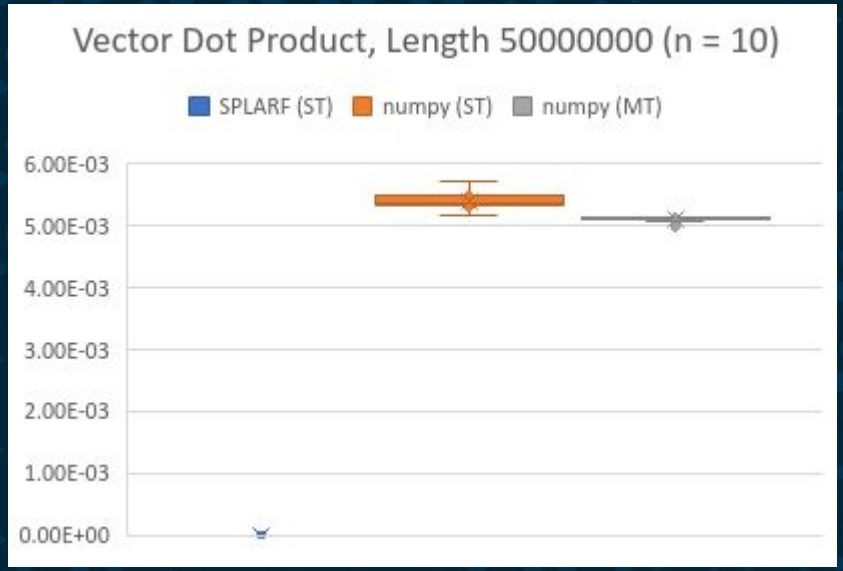
NUMPY CHEATED AND USED C!

Results - Vector Dot Product



- A dot product with two vectors sized 50000000 was tested
  - MPI: ~50ms Overhead
  - Raw C: VERY FAST!
  - Python: ~5ms Overhead

Results - Vector Dot Product Cont.



- g++ -O3 -g3 optimization gained over 100x performance here
- C at its finest
- No, single threaded didn't actually take 0e+0 seconds, on average it took 3.2e-08 seconds, or 32 nanoseconds.
- If the single-threaded algorithm takes 32 nanoseconds, we can assume remaining time is MPI overhead

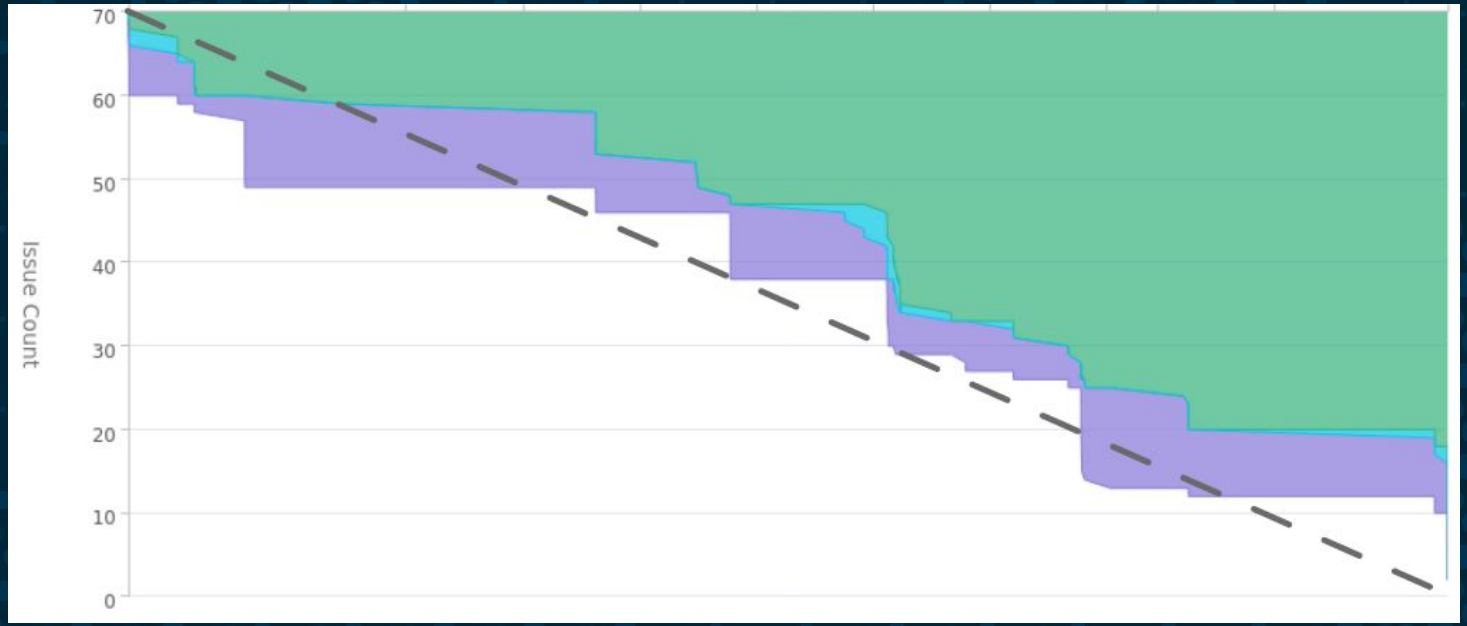
Conclusion

- Research with SPLARF was able to show relative differences between single-threaded, MPI, and Numpy processes
- There are problem sizes where single-threaded processing is faster, followed by MPI, followed by MPI networked. Finding these sizes and switching is key to efficiency gains.
  - MPI added 50ms of overhead on test computer 1
  - Python added 5ms of overhead on top of LAPACK
- Algorithm choice is extremely important
  - Matrix-matrix multiplication can be much faster
- Raw C is fast!

Future Work

- Wrap MPI around LAPACK functions
  - ScaLAPACK does this, is there more we could add?
- Create an algorithm to find the problem sizes to switch from single-threaded to MPI to MPI networked
- Networking performance with SPLARF
- Ax=b Solver In Serial Using LU=A factorization
- Parallel LU=A factorization in O(n^3/p) time
  - Implementing this leads to a parallel LUx=Ax=b solver

Project Burndown Chart



Tasks

Summary	Issue key	Assignee
FInal Presentation	PLAR-70	Jaxton Winder
FInal Presentation	PLAR-69	Sander Joel Burton
Final Presentation	PLAR-68	Max Clark
FInal Presentation	PLAR-67	James Sanford
Final Paper	PLAR-66	Sander Joel Burton
Final Paper	PLAR-65	Jaxton Winder
Final Paper	PLAR-64	James Sanford
Final Paper	PLAR-63	Max Clark
Configure auto-generated latex docs for report	PLAR-62	Sander Joel Burton
Write Documentation Section of report	PLAR-61	Sander Joel Burton
Re-write arithmetic files	PLAR-60	Max Clark
Add citation for code	PLAR-59	James Sanford
Benchmark against other libraries	PLAR-58	Sander Joel Burton
Look into and implement Auto Docs	PLAR-57	Sander Joel Burton
Start Docs	PLAR-56	Max Clark
Linear Regression	PLAR-55	Max Clark
Multi-threaded Norms	PLAR-54	Jaxton Winder
Matrix-Matrix solver, 2D	PLAR-53	Max Clark
Upper Triangular Solver Parallel	PLAR-52	Jaxton Winder
Lower Triangular Solver Parallel	PLAR-51	James Sanford
LU Decomp Parallel Solver Algorithm	PLAR-50	Jaxton Winder
LU Decomp Parallel Algorithm	PLAR-49	James Sanford
Look into arithmetic test	PLAR-48	James Sanford
Create makefile for automated testing	PLAR-47	Max Clark
Create test code for Dot Product	PLAR-46	Sander Joel Burton
Implement Parallel Dot Product	PLAR-45	Sander Joel Burton
Work on implementation	PLAR-44	James Sanford
Merge Max's PR	PLAR-42	Jaxton Winder
Create vector error computation algorithms	PLAR-41	Jaxton Winder
Review Design & Development and Preliminary Results sections of Midpoint Report	PLAR-40	Max Clark
Create various matrix and vector generators for testing purposes	PLAR-39	Jaxton Winder
Determine how to build our library implementation and preliminarily plan for a buildsystem	PLAR-38	Max Clark
Study additional useful linear algebra algorithms	PLAR-37	Jaxton Winder
Review Max's work	PLAR-36	Jaxton Winder

Summary	Issue key	Assignee
Add serial norm algorithms	PLAR-35	Jaxton Winder
Fix bug in vectorProd	PLAR-34	Max Clark
Start on midpoint report	PLAR-33	Sander Joel Burton
Start on midpoint report	PLAR-32	Jaxton Winder
Start on midpoint report	PLAR-31	James Sanford
Start on midpoint report	PLAR-30	Max Clark
Choose algorithm to implement	PLAR-29	Sander Joel Burton
Choose algorithm to implement	PLAR-28	Jaxton Winder
Choose algorithm to implement	PLAR-27	James Sanford
Choose algorithm to implement	PLAR-26	Max Clark
Read previous papers	PLAR-25	Sander Joel Burton
Read previous docs	PLAR-24	Max Clark
Jaxton Presentation	PLAR-23	Jaxton Winder
Write up Git/Jira workflow	PLAR-22	Sander Joel Burton
Research Literature for Algorithms to Implement	PLAR-21	Jaxton Winder
Research Literature for Algorithms to Implement	PLAR-20	Sander Joel Burton
Research Literature for Algorithms to Implement	PLAR-19	James Sanford
Research Literature for Algorithms to Implement	PLAR-18	Max Clark
Proposal Revision	PLAR-17	Sander Joel Burton
Proposal Revision	PLAR-16	Jaxton Winder
Proposal Revision	PLAR-15	James Sanford
Proposal Revision	PLAR-14	Max Clark
Methodology Section of proposal - rough draft	PLAR-12	Sander Joel Burton
Proposal - Tasks	PLAR-11	Max Clark
Create Github Repo	PLAR-10	Jaxton Winder
Create branch in Repo	PLAR-9	Jaxton Winder
Create branch in Repo	PLAR-8	Sander Joel Burton
Create branch in Repo	PLAR-7	James Sanford
Create branch in Repo	PLAR-6	Max Clark
Talk with Dr. Watson about our project proposal	PLAR-5	Jaxton Winder
Verify authors names are correct on GitHub	PLAR-4	James Sanford
Create Github Repo	PLAR-3	Jaxton Winder
Set up Jira	PLAR-2	Sander Joel Burton
Project Proposal - Project Management	PLAR-1	Max Clark

Thank You!

Questions?

UtahStateUniversity