Automating Sparse Linear Solver Selection with Lighthouse

Kanika Sood[1], Pate Motter[2], Elizabeth Jessup[2], Boyana Norris[1]
University of Oregon[1], University of Colorado Boulder[2]

Abstract

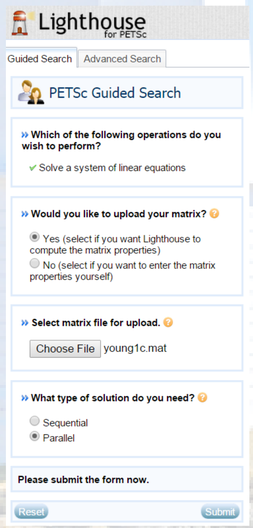
Solving large, sparse linear systems efficiently is a challenging problem in scientific computing. Accessible, comprehensive, and usable interfaces or tools for high quality code production for this computation are not available. Lighthouse is the first framework that offers an organized taxonomy of software components for linear algebra that enables functionality- and performance-based search and generates code templates and optimized low-level kernels. We present the integration of PETSc [2] and Trilinos [3] iterative solvers for sparse linear systems into the Lighthouse framework.

Motivation

- Current HPC linear algebra software is based on years of research.
- The power of libraries is increasing, but they are becoming harder to use. Selecting an appropriate library and using it effectively is a non-trivial task.
- PETSc and Trilinos together offer more than 500 solver-preconditioner combinations.

Our Solution: Lighthouse

- Is a taxonomy that guides users in finding the appropriate numerical method
- Generates customized code templates from user-specified input and output requirements
- Makes it easier to use advanced libraries
- Uses modern web-based application techniques for a user-friendly experience



Approach

Our approach to selecting sparse linear solvers in Lighthouse proceeds as follows:

1. Create a dataset of preconditioner-solver pair timings and other characteristics using PETSc and Trilinos on a collection of sparse matrices [5].
2. Collect "features" for each matrix such as its structure, variability across rows and columns, and spectral properties.
3. Reduce the feature set to remove features that do not contribute significantly to the accuracy of the classification.
4. Use a subset of the data as a training set to build classifiers using machine learning methods. Select the one that most accurately and efficiently classifies solver performance.
5. Verify the accuracy of classifiers using the remaining data as the testing data set.
6. Use the best classifier to predict well-performing preconditioner-solver pairs for new matrices arising in applications.

Results

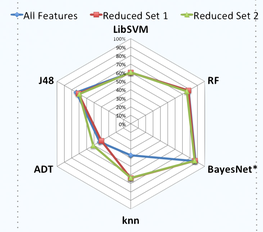
- **PETSc:** BayesNet performs best using Anamod-computed features.
- **Trilinos:** Voting Feature Intervals (VFI) performs best using Trilinos-computed features.

Reduced Feature Sets

▸ **PETSc (with 68 Anamod features and two reduced subsets)**

Feature name :

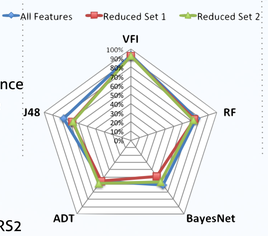
- avg-diag-dist
- nnz *
- norm1
- col-variability
- min-nnzeros-per-row
- row-variability
- n-nonzero-diags *
- kappa



▸ **Trilinos (with 38 Trilinos features and two reduced subsets)**

Feature name :

- Dummy rows
- Trace
- Column diagonal dominance
- Row diagonal dominance
- Diagonal nonzeros *
- row-variability
- Diagonal mean
- Total DNE %



* Feature in RS1 but not in RS2

Accuracy of Classifiers

Timing

Method	T_{All}	T_{RS1}	T_{RS2}
LibSVM	6.88	1.94	1.79
RF	2.67	2.40	1.43
BayesNet	0.10	0.02	0.01
knn	0.001	0.001	0.001
ADTree	0.74	0.18	0.09
J48	0.24	0.19	0.06

Method	T_{All}	T_{RS1}	T_{RS2}
VFI	0.09	0.02	0.02
RF	14.33	6.83	6.09
BayesNet	0.30	0.08	0.07
ADTree	5.83	1.41	0.77
J48	1.05	0.26	0.18

Top 10 solvers labeled as "good"

Occurrence %	PETSc	Occurrence %	Trilinos
3.8%	LSQR, Jacobi	17.2%	Hybrid Block GMRES, Chebyshev
3.6%	BiCG, ASM(0)	9.4%	Hybrid Block GMRES, Diagonal
3.4%	BCGS, ILU(0)	8.8%	Hybrid Block GMRES, Jacobi
3.2%	GMRES, ASM(2)	3.9%	TFQMR, Chebyshev
3.0%	BCGS, ASM(0)	3.7%	Hybrid Block GMRES, ILUT
2.8%	BICG, BJacobi	2.8%	Hybrid Block GMRES, RILUK
2.3%	Chebyshev, Jacobi	2.8%	Block CG, Jacobi
2.2%	BCGS, ASM(3)	2.7%	Block CG, Diagonal
2.1%	iBCGS, ASM(2)	2.4%	Pseudoblock CG, Jacobi
2.0%	CG, ASM(0)	2.4%	TFQMR, Jacobi

Performance of the best classifiers for predicting good solvers

	PETSc with Anamod features				Trilinos solvers with Anamod features				Trilinos solvers with Trilinos features			
Data points	4648				14348				39388			
Good points	1613				1765				2986			
Threshold value b	0.35				0.35				0.15			
	No. of feat.	Best TPR (%) (10-fold)	Best TPR (%) (66% data split)	Classifier	No. of feat.	Best TPR (%) (10-fold)	Best TPR (%) (66% data split)	Classifier	No. of feat.	Best TPR (%) (10-fold)	Best TPR (%) (66% data split)	Classifier
With all features	68	87.6	89.2	Bayes Net	68	71.8	72.2	J48	38	93.0	93.0	VFI
RS1	8	86.9	86.9	Bayes Net	7	75.0	73.5	RF	7	92.0	90.3	VFI
RS2	6	86.4	86.6	Bayes Net	6	73.1	70.0	RF	6	92.5	90.3	VFI

Definitions

- **TPR = TP/ P,** where **TPR** is the True Positive Rate / Sensitivity, **TP** is the number of "good" instances correctly identified as "good", and **P** is the actual number of good instances.
- **Classifiers:** Algorithms that automatically learn how to make accurate predictions based on past observations.
- **Good solvers**: Solvers whose time is within the **threshold value** of the best.
- **Full Feature Set**: A set of properties of a linear system computed using the libraries Anamod and Trilinos.
- **Reduced Set 1 (RS1)**: A subset of the full feature set. It consists of the top-most ranked features.
- **Reduced Set 2 (RS2)**: A subset of RS1 obtained by removing the size-dependent features.

Discussion

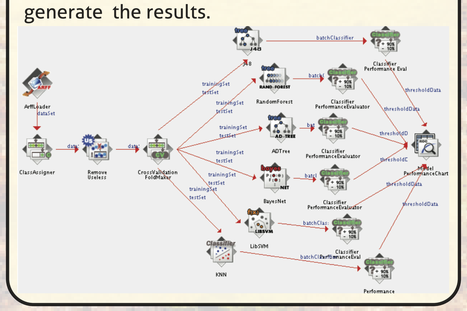
The choice of solver depends on the characteristics of the input problem. We employed several machine learning techniques to make this choice, classifying solvers by computing relatively few inexpensive features of the input system. We performed experiments for full feature sets as well as two reduced feature sets for both PETSc and Trilinos. The accuracy of various machine-learning classifiers (shown as RADAR charts) and the times taken to build these classifiers are shown above (top left) for the full and two reduced feature sets. Next, we measured the performance of various solver-preconditioner combinations for training and testing sets on Argonne's BlueGene/Q and CU's Janus supercomputers. The performance of the best classifiers for predicting good solvers was verified using 10-fold cross validation with a 66%-34% data split for training and testing sets respectively. The table on the left shows the solver configurations that performed best among all configurations as a fraction of all tested combinations. For PETSc, the table shows that there is no clear winner among the various combinations available. The confusion matrices on the right illustrate the overall accuracies for the good and bad solvers identified by the classifiers for Trilinos and PETSc.

Overall classifier accuracy

<-- True label	PETSc with Anamod features					
	All features		RS1		RS2	
	<-- Classified label -->					
	good	bad	good	bad	good	bad
good	485	59	473	71	471	73
bad	390	646	314	722	331	705
<-- True label	Trilinos with Trilinos features					
	All features		RS1		RS2	
	<-- Classified label -->					
	good	bad	good	bad	good	bad
good	963	73	935	100	936	100
bad	2180	10176	1935	10421	1935	10421

Workflow

WEKA [4] knowledge flow components used to generate the results.



Conclusions and Future Work

Results to date indicate that machine learning-based classification can produce up to 93% accurate predictions of well-performing sparse linear system solution methods. We also observed that, for PETSc, no single solver configuration was a consistent good performer. As we expand the set of input problems and solution methods, we expect the accuracy to improve. In future work, we plan to do the following:

- Expand the Lighthouse taxonomy with more HPC routines and libraries.
- Carry out additional tuning of machine learning algorithm parameters.
- Investigate hierarchical machine learning approaches to provide scalable support for all possible solvers and levels of parallelism
- Incorporate scalability and hardware resource information into predictions.

References

[1] The Lighthouse Project. http://lighthousehpc.github.io/lighthouse/, 2015.
[2] PETSc Web page. http://www.mcs.anl.gov/petsc, 2015.
[3] Trilinos. http://software.sandia.gov/trilinos, 2015.
[4] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. SIGKDD Explorations, 11, 2009.
[5] The University of Florida Sparse Matrix Collection. http: //www.cise.ufl.edu/research/sparse/matrices/, 2015.

This work is supported by
CCF-1219089
CCF 155063
CCF 1550202