Wildfire Classification using PETSc-based Support Vector Machines on Distributed-Memory GPU-based Parallel Computers

Richard Tran Mills[α], Zachary Langford[β], Jitendra Kumar[β], Forrest M. Hoffman[β]
[α]Argonne National Laboratory [β]Oak Ridge National Laboratory

1. Introduction

As high-resolution geospatiotemporal data sets from observatory networks, remote sensing platforms, and computational Earth system models increase in abundance, fidelity, and richness, machine learning approaches that can fully utilize increasingly powerful parallel computing resources are becoming essential for analysis and exploration of such data sets. We explore one such approach, applying a state-of-the-art distributed memory parallel implementation of Support Vector Machine (SVM) classification to large remote-sensing data sets in which we want to identify wildfire-affected areas. The parallel tool we employ is PermonSVM, which is built on top of the widely-used open source toolkit PETSc, the Portable, Extensible Toolkit for Scientific Computation. Recent developments in PETSc have focused on supporting cutting-edge GPU-based high-performance computing (HPC) architectures, and these can be easily leveraged in PermonSVM by using appropriate GPU-enabled matrix and vector types in PETSc. This combines the existing extreme inter-node scalability of PETSc to be combined with efficient use of the on-node, fine-grained parallelism that is representative of the design of upcoming exascale-class supercomputers

2. Wildfire Detection

In this study, wildfire detection is a binary classification problem where the model will classify a wildfire pixel or non-wildfire pixel. Knowing the location and extent of wildfire burns can help with a variety of applications (evacuation routes, ecological restoration, climate modeling, etc.). We focus on the 2004 wildfire season for Alaska and the 2020 wildfire season for California (largest wildfire seasons for both states). The study region for Alaska (Figure 1 top) consisted of interior Alaska, while we focus on the entire state for California (Figure 1 bottom). The 2004 Alaska wildfire season was the worst on record in terms of area burned by wildfire, burning more than 27,000 km^2 (6 million acres) of land. The 2020 California wildfire season was also the worst on record, burning more than 17,000 km^2 (4 million acres) of land.

2.1 Data Collection

We used 500 m MODIS 8-day surface reflectance (MOD09A1) for classifying wildfires over Alaska and California. Google Earth Engine was used to extract MODIS products for the wildfire season. MOD09A1 consists of 7 bands ranging from 620 nm to 2155 nm in wavelength. Table 1 describes the time periods for data collection and the amount of pixels for the wildfire and non-wildfire class for our study regions. We plan to look into 8-day land surface temperature (MOD11A2) in future research. California wildfire boundaries for 2020 was collected from the California Department of Forestry and Fire Protection's Fire and Resource Assessment Program (`https://frap.fire.ca.gov`). Alaska wildfire boundaries for 2020 was collected from the Monitoring Trends in Burn Severity (MTBS) project, a multi-agency program (`https://www.mtbs.gov/`).

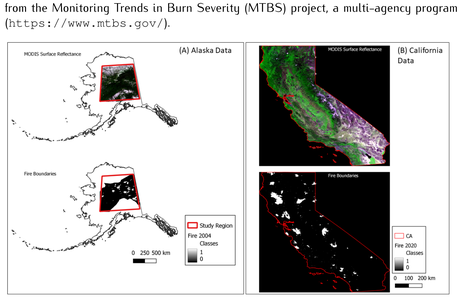

Figure 1: *Study regions showing MODIS surface reflectance and fire boundaries for Alaska (A) and California (B).*

2.2 Data Processing

After the data was collected, we processed the data as follows:

- Convert wildfire boundaries to raster pixels (e.g., -1 = no wildfire and 1 = wildfire)
- Combine all MODIS images together into a single image (i.e., `gdal_merge`)
- Convert 3D array into 2D dataframe, where each feature consists of a band (Figure 2)
- Split data in train and test sets, randomly shuffling rows, using 70% for training and 30% for testing
- Convert dataframes into PETSc binary array for PermonSVM

	label	b0	b1	b2	b3	b4	b5	b6	b7	b8	...	b123	
1471	-1	0.8876	0.7521	0.8885	0.9047	0.2988	0.0530	0.0162	0.7245	0.6314	...	0.2591	0.
1472	-1	0.8876	0.7521	0.8885	0.9047	0.2988	0.0530	0.0162	0.7245	0.6314	...	0.2591	0.
1473	-1	0.8876	0.7521	0.8885	0.9047	0.2988	0.0530	0.0162	0.7245	0.6314	...	0.2591	0.
3229	-1	0.9350	0.7877	0.9233	0.9481	0.3208	0.0525	0.0158	0.7415	0.6453	...	0.2678	0.

Figure 2: *Subset of the data converted into dataframe.*

Table 1: *Description of data, where pixels is equal to 500 m.*

State	Year	Start	End	Area Burned (pixels)	Area Other (pixels)
CA	2020	June	October	81392	1735393
AK	2004	May	September	43680	684069

2.3 Results

Table 2 shows the results on the test dataset, displaying the true positives (TP), false positives (FP), true negatives (TN), false negatives (FN), accuracy, precision, recall and F1 (harmonic mean of precision and recall). We compared PermonSVM to Python's scikit-learn SVM using the default parameters (`https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`). Figure 3 shows how precision and recall are calculated from TP, FP, and FN. Both classifiers show similar scores for accuracy and F1 but PermonSVM shows more false positives for both AK and CA. Since MTBS does not map wildfires under 1000 acres, more false positives (i.e., classifying non-wildfire pixel as wildfire pixel) would be more of interest than having more false negatives.

Table 2: *Results comparing PETSc PermonSVM to Python scikit-learn SVM.*

	CA 2020 PermonSVM	CA 2020 scikit-learn	AK 2004 PermonSVM	AK 2004 scikit-learn
TP	18199	18028	6925	8461
TN	519253	519287	135131	204784
FP	6069	1132	1681	403
FN	1515	6589	1813	4677
Accuracy	0.98	0.98	0.97	0.97
Precision	0.75	0.94	0.80	0.95
Recall	0.94	0.73	0.79	0.64
F1	0.83	0.82	0.80	0.77

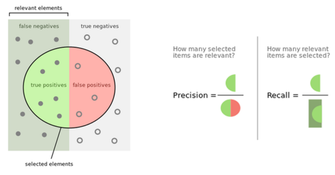

Figure 3: *Diagram explaining precision and recall (source `https://towardsdatascience.com/whats-the-deal-with-accuracy-precision-recall-and-f1-f5d8b4db1021`)*

3. Parallel Solution of SVM Problems using PermonSVM and PETSc

3.1 PermonSVM and PETSc

We build our SVM models using PermonSVM, an SVM implementation designed for distributed-memory parallel HPC resources that is part of the PERMON package (`http://permon.vsb.cz/`):

- Solves the dual formulation for soft-margin SVM problems
- Supports full and relaxed-bias (used here) formulations
- Supports grid search with k-fold and stratified k-fold cross-validation for hyperparameter tuning
- Solves the quadratic programming problems for SVM training using PermonQP, which provides an assortment of sophisticated QP solvers, or, optionally, using the TAO numerical optimization solvers that are part of PETSc
- PermonSVM and PermonQP are built on top of PETSc, which provides parallel communication constructs, parallel matrix and vector operations, Krylov solvers, and (optionally) numerical optimization solvers used by PERMON.

PETSc (the Portable, Extensible Toolkit for Scientific Computation, `https://petsc.org/`) is a software library for the scalable solution of linear, nonlinear, and ODE/DAE systems, computation of adjoints (sometimes called sensitivities) of ODE systems, and optimization. It is most often used in PDE-based simulation, but one of us (Mills) has recently been applying it to model-parallel machine learning problems.

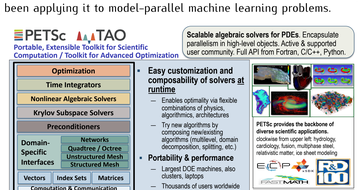

Figure 4: *Diagram illustrating the hierarchical organization of PETSc components and some key features*

3.2 Utilizing GPUs in PermonSVM via PETSc

PETSc's design for GPU support (`https://arxiv.org/abs/2011.00715`) enables existing codes that use PETSc to invoke its GPU-enabled backends with very little, or sometimes no, changes to user code:

- Every PETSc object is an instance of a class whose data structure and functionality is provided by specifying a delegated implementation type at runtime.
  - For example, a matrix in compressed sparse row representation is created as an instance of class `Mat` with type `MATAIJ`, whereas a sliced ELLPACK storage matrix has type `MATSELL`.
- Using GPUs to execute the linear algebra operations defined over `Vec` and `Mat` is accomplished by choosing the appropriate delegated type.
  - For instance, the computations will use the vendor-provided kernels from NVIDIA if `VECCUDA` and `MATAIJCUSPARSE` are specified in user code or through command line options.
  - Alternatively, Kokkos kernels will be used when `VECKOKKOS` and `MATAIJKOKKOS` are specified
- Because the higher-level classes such as timesteppers (`TS`) ultimately employ `Vec` and `Mat` operations for the bulk of their computations, this provides a means to offload most of the computation for PETSc solvers—even the most complicated and sophisticated—onto GPUs.
  - This includes both the TAO optimization solvers and the PermonQP solvers (which rely almost entirely on PETSc `Mat` and `Vec` operations).

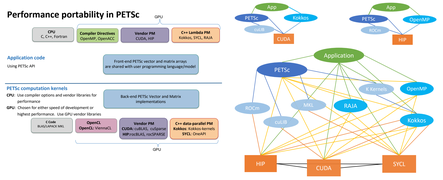

Figure 5: *PETSc application developers are able to use a variety of programming models for GPUs independently of PETSc's internal programming model. At left is a conceptual diagram illustrating the separation between supported GPU programming models for user code and the PETSc backends. At right are examples of PETSc usage with Kokkos-cuLIB-CUDA, OpenMP-ROCm-HIP, and all combinations. Here cuLIB indicates the cuBLAS and cuSPARSE libraries from NVIDIA.*

- We have made a small set of modifications to PermonSVM and PermonQP (adding 84 lines of code) to allow the types of all `Mat` and `Vec` objects to be set via PETSc's runtime database.
- PETSc's GPU back-ends can thus be used by simply specifying appropriate command-line options when invoking the PermonSVM driver.
- Testing with the PermonQP MPGP solver and the TAO BQNLS and BLMVM solvers indicates that approximately 100% of the floating-point operations in the solve step are executed on the GPU when CUDA `Mat` and `Vec` types are used on machines equipped with NVIDIA GPUs

4. Computational Experiments

We have experimented with the Alaska (2004) and California data sets (using a larger data set of the 2016–2020 data, as well as a smaller one from 2020 only) on Summit, the IBM AC922 system at Oak Ridge National Laboratory (ORNL). Our SVM model performance has already been described in section 2. Here, we focus on a discussion of our selection of the underlying optimization solvers and on observed computational performance on Summit. Because our interest is in working with large data sets by scaling across HPC resources, we work with the largest data set we had available (California for years 2016–2020), although this is actually a relatively small dataset for Summit nodes (we aim to eventually use much larger data sets).

4.1 Hardware platform: Summit IBM AC922 system at ORNL

One of the reasons that we have chosen Summit is that its GPU-centric design reflects a trend we expect to continue in upcoming supercomputers, both large and small: NERSC Perlmutter, OLCF Frontier, and ALCF Aurora all feature a similar node design with several extremely powerful GPUs. And very powerful GPUs have also become a mainstay of workstations used for data science.

Summit System totals

- ~ 200 PFlop/s theoretical peak
  143 PFlop/s LINPACK—#2 in TOP500
- 4,608 compute nodes

Node configuration

- Compute:
  - Two IBM Power9 CPUs, each 22 with cores, 0.5 DP TFlop/s
  - Six NVIDIA Volta V100 GPUs, each with 80 SMs–32 FP64 cores/SM, 7.8 DP TFlop/s
- Memory:
  - 512 GB DDR4 memory
  - 96 (6 × 16) GB high-bandwidth GPU memory
  - 1.6 TB nonvolatile RAM (I/O burst buffer)

Almost all compute power is in GPUs!

4.2 SVM Training performance on Summit

4.2.1 Solver selection

- We explored three optimization solvers, MPGP from PermonQP and two quasi-Newton methods, BQNLS and BLMVM, from PETSc/TAO:
  - MPGP is the Modified Proportioning with (reduced) Gradient Projections algorithm
  - BQNLS is the Bounded Quasi-Newton Line Search method for nonlinear minimization with bound constraints; it approximates the action of the inverse-Hessian with a limited memory quasi-Newton formula.
  - BLMVM is the Bounded Limited Memory Variable Metric; it solves the Newton step $H_k d_k = -g_k$ using an approximation B_k in place of H_k, where B_k is composed using the BFGS update formula
- MPGP's convergence for our data sets was too slow to be practical, requiring more wall-clock time for convergence than the queue limits on Summit allow.
- BQNLS and BLMVM are both able to use the GPU back-ends in PETSc for all of their FLOPs, as their limited-memory formulations consist of many vector operations that can be offloaded to GPU.
- BQNLS convergence was fragile for our problems, often determining that it had satisfied the convergence criteria in very few iterations, but then producing very poor quality SVM model performance scores; we therefore use BLMVM for the results presented here.

4.2.2 Selection of MPI rank count per GPU

For several technical and design reasons, PETSc does not directly use multithreading in its CPU code; CPU parallelism is leveraged using an MPI-only model. Using more MPI ranks may enable greater parallelism in code that executes on CPUs, but having many MPI ranks sharing a GPU can incur significant overhead; conversely, using too few CPU cores may make it difficult to feed the powerful GPUs enough work to keep them busy. We investigated the number of MPI ranks to use per Summit node and determined that 24 ranks (4 per GPU) was optimal for our workloads.

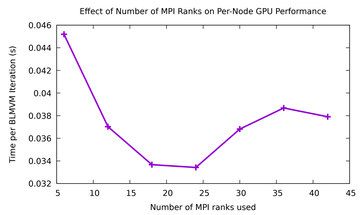

Figure 6: *A study to determine the optimal number of MPI ranks per Summit node (or MPI ranks per GPU—there are 6 GPUs per node).*

4.2.3 Strong Scalability on Summit

Figure 7 displays our preliminary performance observations with the 2016–2020 California data set (5.6 GB in size).

- Using the GPUs results in approximately 2.35X speedup (vs. CPU only) when there is enough work per node to keep the GPUs busy.
- As the number of nodes increases, eventually the problem size per node becomes too small to fully utilize the GPUs. (This is expected behavior in strong-scaling scenarios, as GPUs require enough work to hide high kernel cost latencies and to keep the many processing elements busy).
- We believe that the 2.35X GPU speedup we observe in these preliminary results has considerable room for improvement.
  - Even though PETSc supports it, we ended up not utilizing GPU-aware MPI, so communication between GPUs here requires copying data from GPU to CPU on send, and from GPU to GPU on receive
  - Our experiments using GPU-aware MPI resulted in pathologically poor performance; we believe this is related to some known performance bugs in IBM's Spectrum MPI. Using a different MPI implementation may solve this issue.
  - Although almost 100% of the BLMVM solver FLOPs are executed on GPU, the PETSc performance logs show more communicationn between CPU and GPU than might be necessary.
  - There are avenues for improving BLMVM performance on GPUs (see section 5).

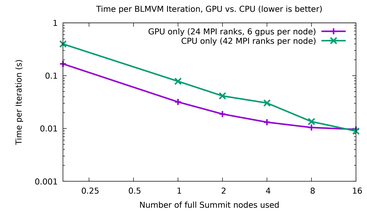

Figure 7: *A strong-scalability study (i.e., input data size is fixed while number of parallel resources is increased) using the 2016–2020 California data set (5.6 GB) on Summit, comparing CPU-only with GPU-enabled runs. Note the use of log-log axes. The leftmost points correspond to 1/6 of a node, i.e., 7 CPU cores and one GPU, and performance is studied out to 16 nodes, or 672 CPU cores and 96 GPUs.*

5. Summary and Future Directions

We consider this a preliminary study, with much work to be done, but so far

- SVM models constructed with PermonSVM show good performance for wildfire classification with MODIS data.
- Scalability across nodes is good; it appears that PermonSVM can scale to very large training datasets, ones impractical to work with on single workstations.
- The Bounded Limited Memory Variable Metric (BLMVM) quasi-Newton solver in PETSc/TAO displays good convergence properties for these problems.
- With only small code changes, we have been able to leverage GPU back-ends in PETSc to offload nearly all FLOPS in SVM training to GPUs
- Initial GPU speedup is modest but encouraging; we have good reason to believe that this can be improved.

Future work will focus on two areas: Improving GPU utilization in SVM training, and improving the training data sets for wildfire classification.

5.1 Improvements to GPU Utilization

- The BLMVM quasi-Newton solver displays good convergence and has proved able to make moderate use of GPU resources on our problems.
- PETSc/TAO developer Alp Dener has done some initial work (`https://alp.dener.me/files/slides/siam-pp20.pdf`) on a new compact dense reformulation that can better leverage fast matrix-vector products on GPUs at the cost of some additional storage, while also reducing required FLOP counts. We will continue this line of development.
- Communication efficiency should be improved if we can use an implementation of GPU-aware MPI (which PETSc already supports)

5.2 Improvements to Wildfire Classification

One focus of future work on data preparation for model development will be on creating an approach that takes into account regions of the state that vary by climate or ecoregion. Figure 8 illustrates a quantitative division of the state of Alaska into 10 ecoregions based on clustering of climate data. We believe that model performance may be significantly improved by constructing separate SVM models for each ecoregion, rather than for the entire state, and a next step is to explore this approach.

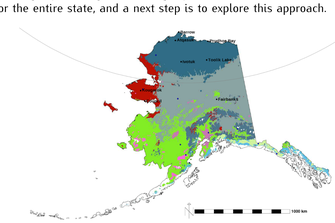

Figure 8: *Alaska ecoregions based on Hoffman et al. (2013), showing each ecoregion a different random color.*

American Geophysical Union Fall Meeting, New Orleans, LA and Online Virtual, December 13–17, 2021.

Parts of this research were supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. Some work was supported by Laboratory Directed Research and Development (LDRD) funding from Argonne National Laboratory, provided by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-06CH11357.