Automated Parameter Tuning for Land Ice Simulations

Xplore Project Report – Stanford CME 291, Spring 2021, #Credits = 3

Carolyn Kao
Institute for Computational and Mathematical Engineering
Stanford University
chkao831@stanford.edu

June 3, 2021

1 Mentors

- **Jerry Watkins, Ph.D., Sandia National Laboratories**
  Jerry Watkins is a computational scientist in the Quantitative Modeling & Analysis Department at Sandia National Laboratories in Livermore, CA. His research focuses on the modeling and simulation of complex physical phenomena in fields such as computational fluid dynamics, computational solid mechanics and climate modeling. He has an interest in the development of high-order numerical methods for next generation supercomputing platforms. He received his PhD in Aeronautics & Astronautics from Stanford University under the guidance of Professor Antony Jameson.
- **Irina Tezaur, Ph.D., Sandia National Laboratories**
  Dr. Tezaur is currently a Principal Member of Technical Staff in the Quantitative Modeling & Analysis Department at Sandia National Laboratories in Livermore, California. She began her research career at Sandia in 2007. Since then, she has had the opportunity to experience research in three of the lab's centers, the Engineering Sciences Center, the Center for Computing Research and the Center for Homeland Security, spanning Sandia's two main sites (Albuquerque, NM and Livermore, CA). Dr. Tezaur holds a Ph.D. in Computational & Mathematical Engineering from Stanford University, in addition to a B.A. and M.A. in Mathematics from the University of Pennsylvania. Her research focuses broadly on developing algorithms and software to enable the modeling and simulation of complex multi-scale and multi-physics problems using high-performance computing. Her research interests include numerical solution to partial differential equations (PDEs), mixed/hybrid finite elements, reduced order modeling (ROM), multi-scale coupling methods, and climate modeling.

2 Faculty Sponsors

- **Trevor Hastie**, Professor of Statistics at Stanford University
- **Eric Darve**, Professor of Mechanical Engineering at Stanford University

3 Project Background and Problem Statement

Global mean sea-level is rising at a rate of 3.2 mm/year and this rate is increasing, with the latest studies suggesting possible increase in sea-level of 0.3-2.5 m by 2100, due to melting of the polar ice sheets (Greenland, Antarctica). With this in mind, Sandia National Laboratories have been developing the MPAS-Albany Land Ice (MALI) Model, as part of the DOE's Energy Exascale Earth System Model, to provide actionable projections of 21st century sea-level rise and support national security missions on high performance computing (HPC) systems.

Our team focused on building a framework for each type of data to facilitate the automated parameter tuning for ice sheet simulations of Earth's polar ice sheets. This is part of an ongoing effort to modernize climate software and develop more accurate and reliable models for probabilistic sea-level projections. We explored a subset of the input parameter space to tune performance, utilized grid and random search for optimization, and automated the tuning process with an efficient framework.

4 Data

Data is constructed and retrieved from Albany Land Ice[1], developed by Sandia National Laboratories. Input files are generated in yaml format. Its corresponding outputs would be extracted into json text files upon running the performance test.

Input In terms of defining the input parameter space to tune, we narrow down our focus to specific parameters on the basis of past experience and domain knowledge from researchers. Then, we proceed to focus on maneuvering the parameters of `mySmoothers` and design our program such that it provides an option to tune parameters on either a single smoother or multiple ones at a time.

```
mySmoother1:
    type:
    ParameterList:
mySmoother4: # w/similar structures; omitted due to space constraint
```

where the `ParameterList` options that we've explored for each smoother follows,

```
    type: RELAXATION
    ParameterList:
        'relaxation: type': MT Gauss-Seidel
        'relaxation: sweeps': positive integer
        'relaxation: damping factor': positive real number
    type: RELAXATION
    ParameterList:
        'relaxation: type': Two-stage Gauss-Seidel
        'relaxation: sweeps': positive integer
        'relaxation: inner damping factor': positive real number
    type: CHEBYSHEV
    ParameterList:
        'chebyshev: degree': positive integer
        'chebyshev: ratio eigenvalue': positive real number
        'chebyshev: eigenvalue max iterations': positive integer
```

Output Once a simulation is performed in Albany Land Ice, checks are made to ensure the solution is correct. For solution that passes these checks, we focus on observing `time_NOX` as the sum of `NOX Total Linear Solve` and `NOX Total Preconditioner Construction` as the output evaluation metric.

```
    "passed": true,
    "timers": {
    # some timers are omitted due to the space constraint
      "Albany Total Time:": 34.4281,
      "NOX Total Linear Solve:": 7.33092,
      "NOX Total Preconditioner Construction:": 6.2781
    }
```

4.1 Offline data

Data are generated with fixed software versions and hardware configurations. For example, we utilized the ICME GPU cluster to generate offline data of a specific version of Albany.

[1] `https://github.com/SNLComputation/Albany`

4.2 Real-time data

Data are generated nightly with potentially new software versions and hardware configurations. For example, real-time data for Sandia's Blake (CPU) and Waterman (GPU) test beds are posted on Github.

5 Methodology

To automate the parameter tuning process, we spend a significant amount of time exploring the parameter space under various linear solvers using optimization algorithms including Grid Search and Random Search. In Grid Search, every combination of a preset list of values within a certain range following a grid-like pattern are run and evaluated. However, in Random Search, the random combinations of parameters within a range of values are generated.

As the search space grows exponentially with the number of parameters tuned, we initially begin with a limited choice of input parameters and some discrete values that are separated with relatively large steps; then, we gradually narrow down the ranges as well as the step sizes to further conduct the search exhaustively over selected preconditioners with multiple grids of feasible values.

On the other hand, Random Search selects random combinations over the parameter space beyond the discrete setting, using a random disturbance modeled by a Gaussian probability distribution over the feasible ranges.

6 Application

6.1 An Offline Autotuning Framework

6.1.1 Implementation

We initiate an offline automated framework that allows users to automatically run multiple simulations over the feasible parameter space. As the offline data is fixed to a specific point in time (hardware/software), the offline framework runs simulations on the fly. Given an input file, we apply the optimization methods across the user-defined ranges with a single command. In particular, the parameter generators for grid search and random search are respectively implemented using `ParameterGrid` and `ParameterSampler` of scikit-learn[2].

Based on the updated input parameters by algorithm, the output ctest files, in json format, yield various timers which serve as the evaluation metrics. Lastly, to demonstrate the tuning results, the program outputs a pandas dataframe to console and simultaneously exports a csv file to showcase the numerical results in ascending order by time (specifically, by `time_NOX` if a performance test passes).

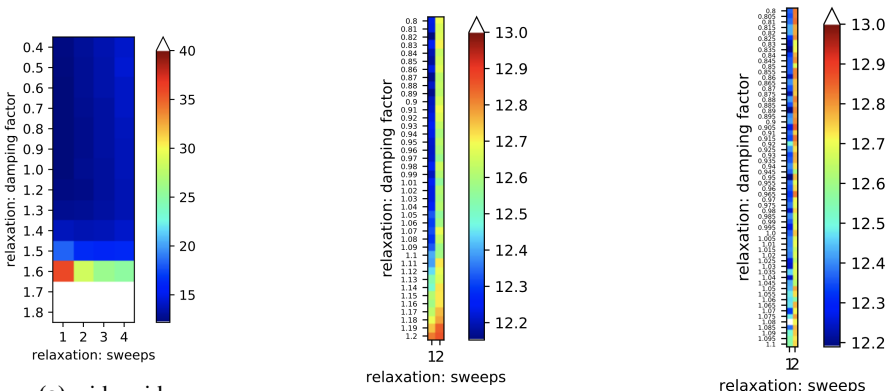
6.1.2 Experiments

We run our offline experiments on both k80 and V100 primarily using ICME-GPU clusters[3], depending on the scope of cases and number of parameters to tune at each time. A sample set of full tuning ranges is showcased in Appendix A.

At the beginning of the exploration, we set up a wider range of the selected parameters as listed above. Then, we zoom into finer grain grids once we're able to narrow down the scope to some subset of the grids using grid search and random search. Fig. 1 shows a visualization of such a sequential process while we tune two types of parameters at a time.

[2] `https://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection`

[3] `https://icme.stanford.edu/get-involved/resources/hpc-resources`



(a) wide grid **(b)** narrower grid **(c)** narrowest grid

Figure 1: Grid Search on mySmoother1 [relaxation: type = MT Gauss-Seidel]

Note that in Fig. 1, we have the `relaxation:type` fixed while changing `sweeps` and `damping factor` which are respectively illustrated on x and y axes. The colorbar on the right is used to indicate the output timers (in seconds) as the evaluation metric. For example, darker shades of blue correspond to better parameter combinations.

From specific numerical and visual results, we also observe that there exists some noise upon running the tests. Inputs with similar values sometimes produce significantly different outputs. Hence, we further refine our offline script such that the user could specify more than one round of simulations of grid search (from which the median would be extracted)[4]; for random search, one may specify the random seed at the beginning of each round for the robustness check and control.

6.2 A Real-time Autotuning Framework

6.2.1 Implementation

Currently, a sets of scripts are running on a nightly basis to post-process the land ice simulation at Sandia. The real-time autotuning framework runs the random search algorithm, automatically updating the specified input file and recording history such that the next nightly test will run a new iteration based on the updated input file. Similarly to the offline version, the corresponding outputs (timers, etc.) would be automatically extracted and recorded to the history file the next time calling this script, along with new input information for the next nightly test.

To help one keep track of the day-to-day tuning history, this program exports historical csv files by chronological order and by performance. Furthermore, at every iteration, we check to see if the today's iteration is better than all past iterations. If true, we update `[inputfile]_Best.yaml` with the inputs from the current iteration.

6.2.2 Experiments

The real-time framework is currently adopted for simulations using blake (CPU)[5] and weaver (GPU)[6] of Sandia.

7 Analyses

7.1 Grid Search vs. Random Search

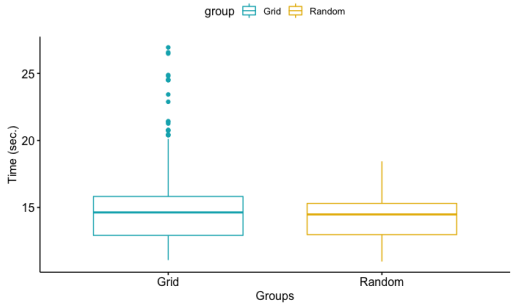
Using the offline framework in Section 6.1, we observe that the Random Search algorithm typically outperforms the Grid Search one, given the same sample size and similar ranges of inputs. An

[4]Due to the space constraint, a sample visualization to showcase the noise reduction is moved to Appendix B for reference.

[5]`https://github.com/ikalash/ikalash.github.io/tree/master/ali/blake_nightly_data`

[6]`https://github.com/ikalash/ikalash.github.io/tree/master/ali/weaver_nightly_data`

example is illustrated below. Using Random Search, with sample size of 352 for each method, we are able to achieve the best performance of the two (with time = 10.96464 secs) and a slightly lower mean of time (14.20902 secs) without as many outliers as in Grid Search[7].



Time	Grid	Random
#sample	352	352
time_lowest	11.0720	10.96464
time_mean	14.84668	14.20902
time_sd	2.735190	1.518329

Figure 2 & Table 1: Grid Search vs. Random Search on multiple smoothers

7.2 Manual Tuning vs. Autotuning

This experiment uses a fixed blake/weaver build to process the data using the nightly framework. Admittedly, we don't practically run the script in the real daily sense due to the time constraint. Within the 10-week quarter, it is infeasible for us to get enough data points for analyses. Hence, we choose to utilize the real-time framework with the random search method to run the ctests in a loop for 100 iterations.

To evaluate the efficiency of our autotuning framework specified in Section 6.2, we run two nightly cases (`vel` and `ent`) that utilize the same solver to solve dissimilar sets of equations on the same mesh, using both CPU (blake) and GPU (weaver). This results in having four sets of results to compare, namely `blake_vel, blake_ent, weaver_vel` and `weaver_ent`.

Upon running 100 iterations, we have

Cases	Manual Tuning (sec.)	Autotuning (sec.)	Speedup
blake_vel	3.533972	2.658731	1.33x
blake_ent	3.07725	2.036044	1.51x
weaver_vel	19.13084	16.30672	1.17x
weaver_ent	19.76345	15.00014	1.32x

Table 2: Autotuning Speedup

The "Manual Tuning" column showcases the results that were previously obtained after tuning from a domain expert at a fixed point in time on a specific software and hardware. Compared to the hand-tuned results, the autotuner is able to find inputs that were anywhere between 1.17x-1.51x faster depending on the case.

8 Challenges and Future Work

One thing that we encounter using the brute-force searching algorithm is that given a multiple-level nested input yaml file, tuning just a few parameters over two levels (smoothers) is already very costly. Such a high complexity would potentially hinder us from fully exploring an input space with larger feasible ranges or investigating more parameters beyond what we have until this point. In addition, even though a significant speedup is seen with our autotuning framework from Table 2, the percentage of the failure cases is concerning, too.

[7]Even though Random Search typically outperforms Grid Search, further statistical tests indicate that we fail to make an assertion that the average timer outputs are significantly different due to the inhomogeneity in variances by F-test.

Cases	#Passed Runs	#Failed Runs	%Failure
blake_vel	70	30	30%
blake_ent	37	63	63%
weaver_vel	71	29	29%
weaver_ent	26	74	74%

Table 3: Failing Percentage

Admittedly, the searching techniques such as Grid Search and Random Search roam the full space of available parameter values in an isolated way without paying attention to past results. Tuning by means of these techniques can become a time-consuming challenge especially with large parameters spaces as we have, since the search space grows exponentially with the number of parameters tuned. At this point, there's no direct way of bringing in prior learning into our algorithms without direct human interference. As a result, a potential future work could be incorporating sequential search strategies such as Bayesian Optimization, which distinguishes itself from Grid Search and Random Search as it takes into account past evaluations in choosing the parameter sets to evaluate next.

9 Conclusion

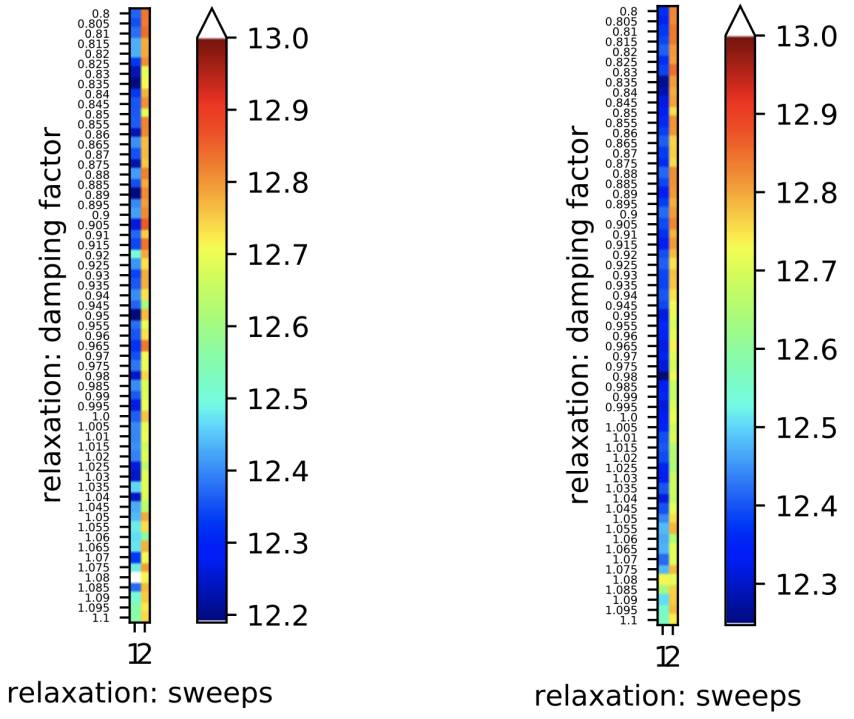
In the past, the parameter search over an input parameter space is normally conducted based on past experiences, one parameter at a time. In spite of such inefficiency and tediousness, these parameters could easily become outdated, not to mention that some optimal parameters may not extend beyond an HPC platform given the rapid changes in terms of hardware and software. Hence, implementing an efficient automated tuning framework with high extensibility to identify and optimize over the values for optimal parameters is especially critical for high-resolution simulations on high performance computing systems.

A A Sample Tuning Ranges

The brackets `[lower_bound, upper_bound]` specify the lower and upper bound (inclusive on both ends) for each parameter under the ParameterList.

```
type: RELAXATION
ParameterList:
    'relaxation: type': MT Gauss-Seidel
    'relaxation: sweeps': [1, 3] # positive  integer
    'relaxation: damping factor': [0.5, 1.5] # positive real number
type: RELAXATION
ParameterList:
    'relaxation: type': Two-stage Gauss-Seidel
    'relaxation: sweeps': [1, 3] # positive integer
    'relaxation: inner damping factor': [0.3, 1.8] # positive real
       number
type: CHEBYSHEV
ParameterList:
    'chebyshev: degree': [1, 6] # positive  integer
    'chebyshev: ratio eigenvalue': [10., 50.] # positive  real number
    'chebyshev: eigenvalue max iterations': [5, 100] # positive
       integer
```

B Noise Reduction for Grid Search



(a) With a single round

(b) With 3 rounds

Figure 3: This showcases the noise reduction by choosing the median value from multiple rounds of simulations by grid search.