Automatic performance tuning for Albany Land Ice

Max Carlson, Jerry Watkins, Irina Tezaur

PII:	S0377-0427(23)00166-8
DOI:	https://doi.org/10.1016/j.cam.2023.115222
Reference:	CAM 115222
To appear in:	*Journal of Computational and Applied Mathematics*
Received date :	31 October 2022
Revised date :	7 March 2023

Automatic performance tuning for Albany Land Ice

Max Carlson[1], Jerry Watkins[1], and Irina Tezaur[1]

[1]Sandia National Laboratories, Quantitative Modeling & Analysis Department, Livermore, CA, USA.

March 7, 2023

Abstract

Accurate simulation of the evolution of polar ice-sheets requires a massive amount of computational power. In order to take advantage of the newest generation of supercomputing clusters, the Albany Land Ice code has been modernized for performance portability across a variety of parallel architectures, with a focus on enabling end-to-end GPU capability. Albany uses a multigrid preconditioning approach for solving linear systems via performance portable smoothers from the Trilinos package Ifpack2. Since the Albany Land Ice code is constantly evolving and both Albany and Trilinos are in constant development, it is likely that the optimal choice of solver parameters will change over time. It is therefore critical to have an automatic performance tuning framework to ensure that the best possible performance is maintained. Toward this effect, we have developed an automatic performance tuning framework to determine the best fine- and coarse-grid smoothing algorithms and parameters. We treat the underlying performance model of the linear solve as a black box and use the python-based `GPTune` Bayesian optimization library to determine the optimal smoother choice and parameters. Using this approach, we have found smoothers and their corresponding parameters that result in, on average, 1.2 times faster, and up to 1.5 times faster solve-times than our manually-tuned parameters. We also show that the proposed auto-tuning approach produces reliably better parameters than naive black box optimization techniques like random search for a given function evaluation budget. By implementing our tuning framework in the Python-based workflow management tool `parsl`, we also ensure that we efficiently use available computing resources during the tuning process and avoid unnecessary long wait times in computing cluster job queues.

1 Introduction

In order to properly take advantage of the new generation of exascale computing environments, there are a number of performance challenges that scientific software must overcome. Performance portability is one of these challenges due to the ever increasing number of parallel computing architectures that come with their own specific performance needs and development frameworks. Additionally, software must exhibit a very high degree of scalability in order to actually take advantage of the available hardware needed to achieve exascale performance. For the task of Earth system modeling, these challenges are compounded due to the variety of component climate modules that are developed independently, each with their own specific performance concerns. As part of the push for scalable and portable climate simulation software, the Albany Land Ice model (ALI) [1] has been modernized to take advantage of the Kokkos performance portability framework to enable GPU runs. While improving the scalability of linear solvers on the GPU is ongoing work, in this paper we present a method for automatically choosing linear solver parameters to attain maximal performance for any given linear solver strategy.

Typical optimization techniques rely on producing or approximating derivatives of an objective function with respect to the parameters of interest. Black box optimization refers to the class of optimization problems where the inner-workings of the function being minimized is unknown or inaccessible and derivatives cannot be constructed or are too expensive to be constructed. Since we are not only interested in tuning the linear solver parameters but also wish to treat the *choice* of algorithm as a tunable parameter, we must treat the underlying performance model as a black box. It is also likely that the optimal solver parameters will change as ALI is developed or the problems of interest are modified. Therefore, it is necessary for us to have a method that does not make strong assumptions about the underlying performance model. There are many techniques for black box optimization, including naive methods like grid/exhaustive search and random search and more advanced methods like Bayesian optimization. In this paper, we focus our attention on a specific Bayesian optimization implementation known as `GPTune` [2] that has been used to tune the performance of applications including SuperLU [3] and choosing randomized sketching algorithms for computing least squares [3].

Some of the approaches used for automatic performance tuning include racing-based methods [4, 5] and population-based evolutionary methods [6]. Recent methods for performance tuning treat the problem as a surrogate-assisted bi-level optimization [7]. In the case where the underlying performance model is not treated as a blackbox, gradient-based approaches can be used to train a model that predicts optimal parameters for unseen input problems. [8] The type of method we are interested in for this paper is a sequential model based global optimization method and such methods have been benchmarked [9] on a variety of performance tuning applications including HYPRE.

Albany is a C++ finite element code developed for solving partial differential equations [10]. Over the years, the Albany code base has housed a variety of science and engineering applications, including the Aeras global atmosphere code [11], the Quantum Computer Aided Design (QCAD) simulator [12], and the Laboratory for Computational Mechanics (LCM) research code [13, 14, 15], the Arctic Coastal Erosion (ACE) model [16] and the Albany Land Ice (ALI) ice sheet model solver [17]. Attention herein is restricted to the ALI module within Albany, which discretizes and solves the so-called first-order Stokes equations [18, 19] for the ice velocities. In order to perform dynamic land ice simulations and enable coupling to the U.S. Department of Energy's Energy Exascale Earth System Model (E3SM) [20], ALI has been coupled to the Model for Prediction Across Scales (MPAS) climate modeling framework [21] to yield the MPAS-Albany Land Ice (MALI) model; for more details on MALI, the interested reader is referred to [1].

The Albany code within which ALI is implemented relies on a number of packages from the Trilinos [22] software stack for tasks including automatic differentiation (Sacado), non-linear (NOX) and Krylov linear (Belos) solvers, multigrid preconditioners (MueLu) and multigrid smoothers (Ifpack2). For each non-linear iteration, a linear system is produced by Albany and solved using an iterative multigrid preconditioner designed to take advantage of extruded ice-sheet meshes [23]. The goal of this paper is to choose and tune the performance-portable Ifpack2 smoothers to ensure optimal performance of the total run-time of the linear solve phase of MALI.

To achieve this goal, we have developed an automatic performance tuning framework for Albany using the Python-based workflow management tool `parsl` [24] and the Bayesian optimization software `GPTune`. Previous work on automatically tuning Albany used random search to improve expert-determined linear solver parameters. The tuning framework we present here improves on the random search approach by using black box optimization techniques to guide parameter exploration to more efficiently and reliably find optimal solver parameters for a given problem. In Section 2, we present the design and implementation details of our automatic performance tuning framework. In Section 3, we demonstrate the usage of our tuning framework, and show that we can achieve a significant performance improvement on GPU by using Bayesian optimization over manually-tuned smoother parameters. We also show that the Bayesian optimization approach results in better optimal parameters than more naive black box optimization techniques such as random search.

2 Method

2.1 Black Box Optimization

As discussed in more detail in Sections 2.3 and 2.4, we wish to optimize a wide range of parameters, some of which take on numeric values, whereas others are strings denoting the names of different smoothing algorithms used within the ALI linear solver. To handle different parameter types from different spaces, e.g., categorical, discrete, continuous, it is helpful to treat the linear solver as a black box and use existing black box optimization techniques. For discrete parameter spaces, a naive approach is to use 'exhaustive' search which is equivalent to testing all possible parameters and returning the optimal. This approach is really only feasible when the parameter space is trivially small or function evaluations are cheap. For continuous parameter spaces, the corresponding naive approach is 'grid' search. By overlaying a grid on the continuous parameter space, search can be done exhaustively but there is no guarantee a true optimal is found. Another approach, for discrete or continuous parameter spaces, is random search where candidate parameters are selected at random from a given distribution. Random search is attractive in that all of the function evaluations can be done completely independently. The drawback is that the choice of any given set of candidate parameters from random search does not take into account any of the information exposed by evaluating other sets of parameters. Bayesian optimization is a technique that balances the idea of exploration (random search) and exploitation (search guided by previous function evaluations) [25]. In this work, we explore the method of Bayesian optimization for performance parameter tuning. Bayesian optimization fits a Gaussian model to the performance data collected by another method such as random search and then makes a prediction about what the optimal parameters should be given the recorded data. The objective of this optimization is to minimize the total time spent during the solve phase of ALI and is measured by existing timing instrumentation throughout Albany.

There are a number of existing software libraries for doing Bayesian optimization that we looked into for this framework such as 'sklearn' [26], 'OpenTuner' [27], and 'HpBandSter' [28] that each have their own pros and cons. We decided to use a Bayesian optimization library known as `GPTune`

[2] for this framework due to its multi-task optimization functionality that seems to be unique, among these tuning software libraries, to `GPTune`. The idea behind multi-task optimization is that the result of tuning one problem could be used to inform the tuning of another related problem that has a different but related optimal solution. Multi-task optimization potentially enables us to do optimization on problems with less expensive function evaluations (such as a problem over a small mesh) to give us a good idea of the optimal parameters of a problem that requires more expensive function evaluations. `GPTune` also has the ability to include a performance model for tuning cases where we have a good guess of how performance varies with tuning parameters.

2.2 Online vs Offline Tuning

Automatic performance tuning can be further categorized into offline and online methods [29]. Offline tuning consists of generating candidate parameters and then optimizing the objective function via trial executions and then whatever optimal parameters that are found are used for all practical executions. Online tuning, by comparison, evaluates candidate parameters during practical execution. The framework we present in this work is an example of offline tuning. In this section, we mention some of the tradeoffs between online and offline tuning and justify the usage of offline tuning for automatic performance tuning of ALI.

One of the benefits of online performance tuning is that online tuning is done during practical execution of the software so that the time-cost of tuning is somewhat hidden. However, this approach exposes the user to uncertainty in performance since any given execution of the software might use a sub-optimal set of parameters in order to explore the parameter space. In contrast, with offline tuning, performance guarantees are available for practical execution but the time-cost of the tuning process can not be hidden and may be prohibitively expensive. One of the goals of this paper is to determine the cost of offline tuning for Albany Land Ice. In the case of offline tuning having an acceptable cost, we can then avoid exposing the end-user from performance uncertainty in practical execution. The intended use case for this tuning framework is to supplement our nightly performance and regression testing of ALI with tuning runs. Since optimal parameters likely do not change on a nightly basis, these tuning runs will only be done when a change in performance is detected by a changepoint algorithm within Albany's performance monitoring tool.

Another tradeoff between online and offline tuning is that offline tuning produces optimal parameters for a specific instance of a problem but these optimal parameters may not be optimal in general. With online tuning, tuning is done during practical execution on a per-problem basis. There are approaches, such as multi-task optimization, for transferring the optimal solutions found in an offline setting to similar problems and is something we intend to explore in the future since the scope of this paper is limited to tuning a single problem instance.

2.3 Problem Details

Albany can be used to model various aspects of the evolution of ice-sheets including the thickness, velocity, and temperature. For these experiments, we focus our effort on tuning the velocity solve which is modeled as a first-order Stokes equation. The meshes used in ALI are unstructured in two-dimensions and then extruded in the relatively-thin vertical direction, resulting in a semi-unstructured discretization. ALI employs a hybrid multigrid approach that does structured semi-coarsening in the vertical direction until the only remaining matrix entries correspond to the two-dimensional unstructured grid which is then coarsened using algebraic multigrid.

For this problem, Albany employs two multigrid smoothers (fine and coarse grid) that can be tuned to reduce total solve time. For each smoother, there are three choices of GPU-enabled algorithms from Ifpack2: Multi-threaded Gauss-Seidel, Two-stage Gauss-Seidel, and Chebyshev. For the Gauss-Seidel smoothers, we need to choose the number of total sweeps and the damping factor (or inner damping factor for the two-stage variant). The Chebyshev smoother has three parameters. The first is the degree of the Chebyshev polynomial, second is the ratio of max and min eigenvalues of the linear operator, and the third is the maximum number of iterations for the smoother. The search space for a given smoother is then a combination of categorical, discrete and continuous spaces:

- Smoother Type: {Multi-threaded (MT) Gauss-Seidel, Two-stage Gauss-Seidel, Chebyshev}
- Number of sweeps: {1, 2} [MT Gauss-Seidel, Two-stage Gauss-Seidel]
- Damping factor: [0.8, 1.2] [MT Gauss-Seidel]
- Inner damping factor: [0.8, 1.2] [Two-stage Gauss-Seidel]
- Chebyshev degree: {1, 2, 3, 4, 5, 6} [Chebyshev]
- Eigenvalue ratio: [10.0, 50.0] [Chebyshev]
- Maximum chebyshev iterations: {5, 6, ..., 100} [Chebyshev]

Our goal for performance tuning is to minimize the total runtime of a first-order Stokes solve in ALI. The objective function we are minimizing takes smoother parameters as inputs and has a single scalar output of total runtime. We refer to a single run of the first-order Stokes solver generically as a "function evaluation". For efficiency, we would like to produce an optimal solution using as few function evaluations as possible and refer to the total number of function evaluations for a given tuning run as the "total function evaluation budget".

2.4 Design and implementation

Since we would like to control the parallelism for any given function evaluation, we use the reverse communication interface (RCI) mode in `GPTune`. This means that each call to the `GPTune` tuning script will update a tuning database with candidate parameters and then exit. We then run Albany using each of the proposed candidate parameters and then write the total solve time back into the database and run the `GPTune` script again. Since we are running Albany many times in parallel from a high-level python tuning script, we utilize the PyAlbany [30] python interface to simplify the workflow. This process continues until we have exhausted the total function evaluation budget and then the tuning script will report the optimal parameters according to the model that has been generated.

For a given tuning run, the total function evaluation budget is split evenly between the exploration phase and the exploitation phase. In notation, we refer to the total budget as 2ϵ, therefore each phase does ϵ function evaluations. In the exploration phase, candidate parameters are chosen randomly and evaluated in parallel. `GPTune` has multiple sampling approaches for choosing candidate parameters. The first is Monte Carlo sampling (MCS) which samples each search dimension uniformly and is equivalent to our previous random search. The second is Latin hypercube sampling (LHS) which ensures samples are evenly distributed through the search space. All of the candidate parameters for the sampling phase are returned in a single call to the RCI `GPTune` script and can be evaluated in parallel. During the exploitation phase, `GPTune` chooses candidate parameters by updating the underlying Gaussian model and making a prediction and is therefore sequential in terms of function evaluations. An example diagram of the full tuning process can be seen in Figure 1.

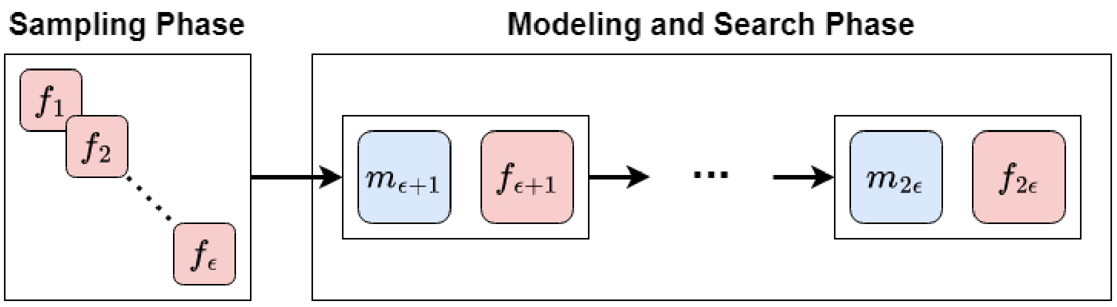

Figure 1: Example of `GPTune` reverse communication interface workflow. The sampling (exploration) phase can be done in parallel, whereas the modeling and search phase (exploitation) is sequential. Modeling step m_i involves updating the underlying assumed (Gaussian) performance model. The total number of function evaluations that will happen in a given tuning run is notated in this figure as 2ϵ.

2.5 Workflow Management

A naive approach to tuning via `GPTune`'s RCI mode would be to have a script that queries `GPTune` for candidate parameters and then launches a job for each independent function evaluation as they are received. In practice, this approach is untenable on large computing clusters since it would require for each new requested function evaluation to put a new job into the back of the queue. The majority of the wall-clock time for this approach would likely be spent waiting for a job to reach the front of the cluster's queue. An alternative approach would be to launch a single job that runs the entire tuning process. However, this is not ideal either since, for the sampling phase, we want a large number of machines to be doing function evaluations in parallel but that would mean most machines are idle during the modeling phase. The other extreme would be to launch a job on a single machine for the modeling phase; the downside of this approach is that the parallelism of the sampling phase could not be taken advantage of.

The solution to this problem is to use a workflow management tool to dynamically assign function evaluations to available helper jobs. For our purposes, we use the Python workflow management tool known as `parsl` but there are other tools with attractive functionality such as `fireworks` [31].

```
# 1) get number of function evaluations
requested_num_evals = get_num_function_evals(...)
num_func_evals = requested_num_evals.result()

while num_func_evals > 0:

    # 2) create function evaluation configuration files
    requested_eval_configs = get_function_eval_configs(...)

    # 3) do function evaluations
    function_evaluated = []
    for k in range(num_func_evals):
        function_evaluated.append(evaluate_function(...))

    # 4) get function evaluations
    evaluation_results = []
    for k in range(num_func_evals):
        evaluation_results.append(get_evaluation_result(...))

    # 5) write function evaluations to database
    database_updated = update_database(...)

    # 6) get remaining number of function evaluations
    requested_num_evals = get_num_function_evals(...)
    num_func_evals = requested_num_evals.result()
```

Listing 1: "Pseudocode example of the core tuning algorithm in `parsl`. Steps 3 and 4 are done in parallel and synchronization occurs at step 5. Each function call corresponds to a `parsl` app that returns a Python futures object to handle synchronization and file I/O dependencies."

3 Tuning Framework Evaluation

3.1 Experiment Setup

In order to evaluate our automatic tuning framework, we need to answer the following questions. First, what is the optimal budget for the number of function evaluations for a tuning run? Ultimately, we would like to find the optimal solver parameters in as few function evaluations as possible. Second, does using Bayesian optimization improve the results we would have gotten from just a simple random search? With random search, there is no sequential modeling phase and all function evaluations can be done in parallel. Therefore, if random search is just as good as "smarter" approaches, then random search would be the efficient choice. Third, how frequently does each method produce a parameter set that results in a failed run of Albany? A failed run of Albany could take considerably longer than a successful one. The total time elapsed of the tuning process could be excessive if the majority of evaluations are timing out and failing; hence, a method that produces quality candidate parameters is important. It is also possible that a failed run finishes significantly faster due to invalid parameters causing the solve to fail right away. However, for this problem, we have chosen the search space such that no invalid parameters would be selected as candidates. Our framework detects and reports the type of failure by checking internal solver flags that are exposed by the PyAlbany interface. The two types of failures that show up for this problem are when maximum wall-time has elapsed or the number of solver iterations has exceeded the maximum without convergence.

To answer these questions, we use our tuning framework to tune ALI's first-order Stokes solver on an extruded unstructured tetrahedral mesh of the Greenland ice sheet with variable resolution ranging from 20km to 3km. The results of these experiments are compared against the first-order Stokes solver with "default" solver parameters that were originally chosen by multigrid experts. The default fine-grid smoother uses a two-stage Gauss-Seidel method with an inner damping factor of 0.25 and only 1 relaxation sweep. For the coarse-grid, the default smoother is two-stage Gauss-Seidel with inner damping factor of 1.0 and 4 relaxation sweeps. We ran all of these experiments on the Perlmutter computing cluster using a single A100 GPU per function evaluation. Using these default smoother parameters, the total runtime was about 30.2 seconds and is used as a baseline for comparing the tuning results. It should be noted that the optimal parameters may vary depending

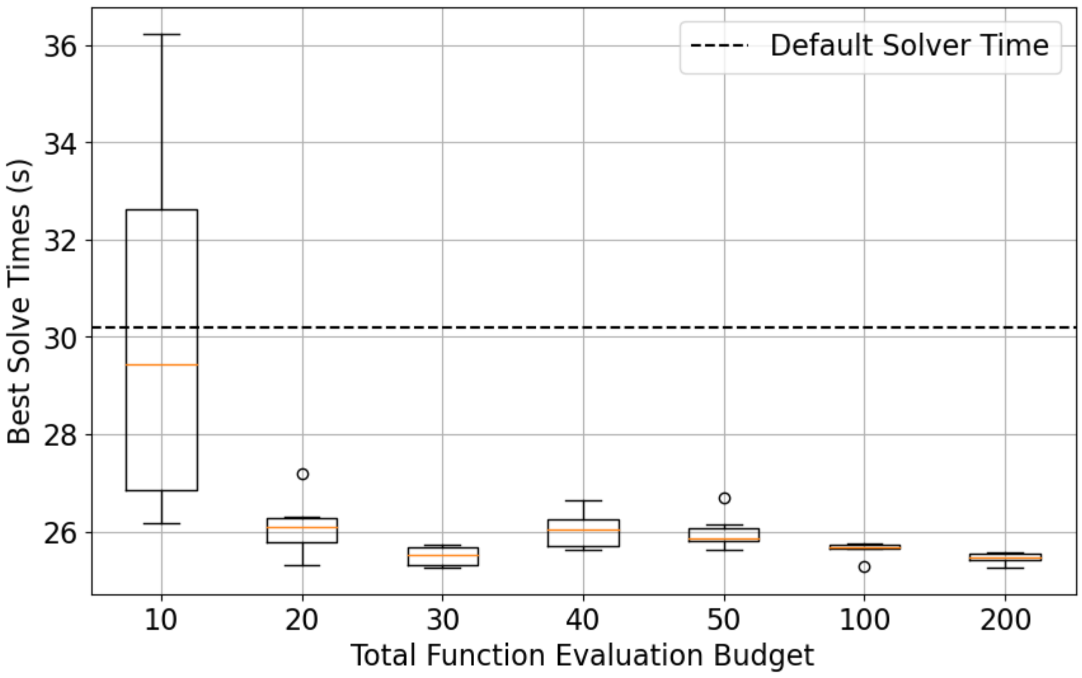

Figure 2: Best runtimes for a range of function evaluation budgets. For each budget, best runtimes are reported for 6 sample runs of the tuning process.

on architecture so it is helpful to have an automatic tuning framework for deploying ALI on other machines.

3.2 Results

In order to determine the optimal function evaluation budget, we ran the tuning framework with a range of budgets, six times for each choice of budget, and then recorded the best times and solver parameters. The results of this experiment can be seen in Figure 2. The results seem to match the expectation that as you increase the number of function evaluations, the likelihood of finding a good time increases. However, the best times produced using a function evaluation budget of 200 is not dramatically better than simply using a budget of 30. It is also interesting to note that a budget of 30 seems to reliably produce better results than a budget of 40 or 50 and comparable results to large budgets like 100 or 200. These results seem to indicate that 30 is a good choice of evaluation budget for this particular problem. While a budget of 30 is a good choice, a large budget could potentially be more likely to find outlier solve times that are significantly better.

Additionally, we are interested in how frequently a candidate parameter set results in a failed run of Albany. To evaluate this, we looked at all of the function evaluations produced over all of the runs in Figure 2 separated into samples generated during the sampling phase (latin hypercube random search) and the modeling phase. For the sampling phase, about 64% of runs resulted in failure whereas in the modeling phase, about 18% of runs resulted in failure. Combining these function evaluations, we see that on a given run of the tuning framework, about 41% of parameters will result in a failed run. Since the sampling phase is equivalent to random search, using Bayesian optimization brings the failure rate down from about 64% to 41% and could be brought down further by spending more of the evaluation budget in the modeling phase. In Figure 3, the distribution of

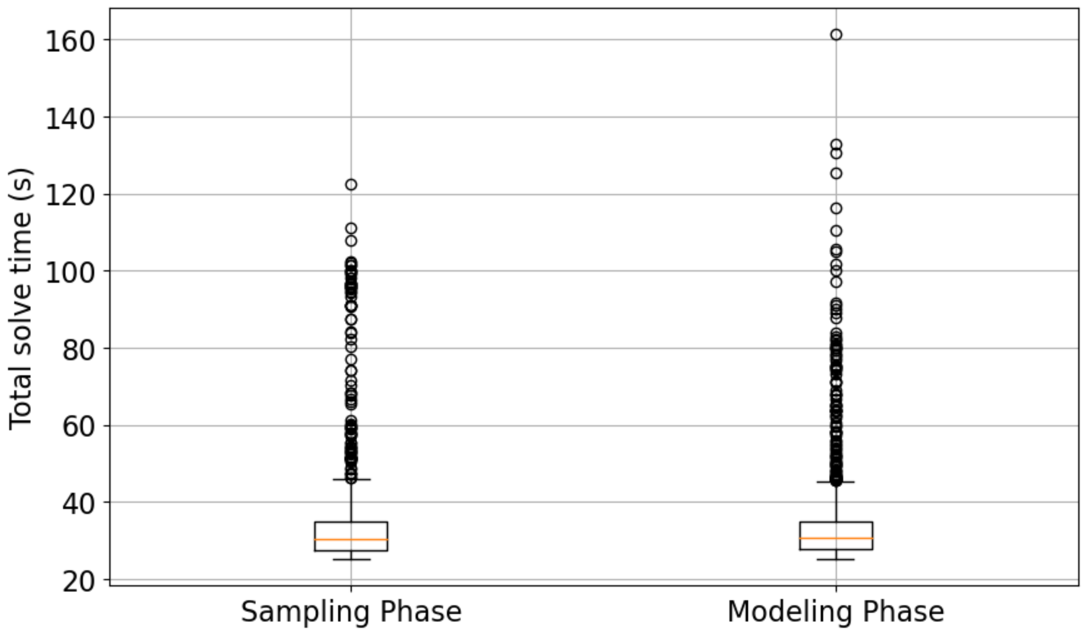

Figure 3: Distribution of runtimes for a single run of the tuning process separated by tuning phase. Failure rates for each phase are significantly different, but of the successful evaluations, the overall distribution is largely the same for both phases. The failure rate for sampling is about 64% and the failure rate for modeling is about 18%.

runtimes can be seen for the modeling and sampling phase. While the failure rates of each phase are significantly different, the overall quality of successful runs does not seem to change between the phases.

The final experiment of this section is set up to determine if using Bayesian optimization results in good enough results to offset the loss of parallelism that can be found in random search. For this experiment, we fixed the total evaluation budget to 30 and then ran multiple instances of three separate methods for parameter tuning. The first is the `GPTune` Bayesian optimization method described in this paper with 15 evaluations done in parallel for the sampling phase and then 15 sequential evaluations for the modeling/search phase. The second case is using the full function evaluation budget to explore candidate parameters produced by LHS. Finally, the third case is the same as the second case but using MCS to produce candidate parameters. This third case is equivalent to our previous work using random search to optimize [32]. Each case was run 20 times to produce the best times seen in Figure 4.

The results from the experiment seen in Figure 4 show that random search using MCS reliably produces slower runtimes than the other two methods. However, the difference between random search via LHS and Bayesian optimization is not quite as dramatic even though `GPTune` is producing reliably better runtimes. If the goal is to minimize wall-clock time, it might make sense to use LHS random search due to the available parallelism. However, from a node-hours perspective, the Bayesian optimization approach is the better choice since less time is being wasted on evaluating candidate parameters that will result in failed runs.

Finally, during the exploration of our framework, we found that, on very rare occasions, parameters that resulted in significantly better runtimes than seen in these figures were attained. While the vast majority of optimal runtimes were between 25 and 27 seconds, we found a handful

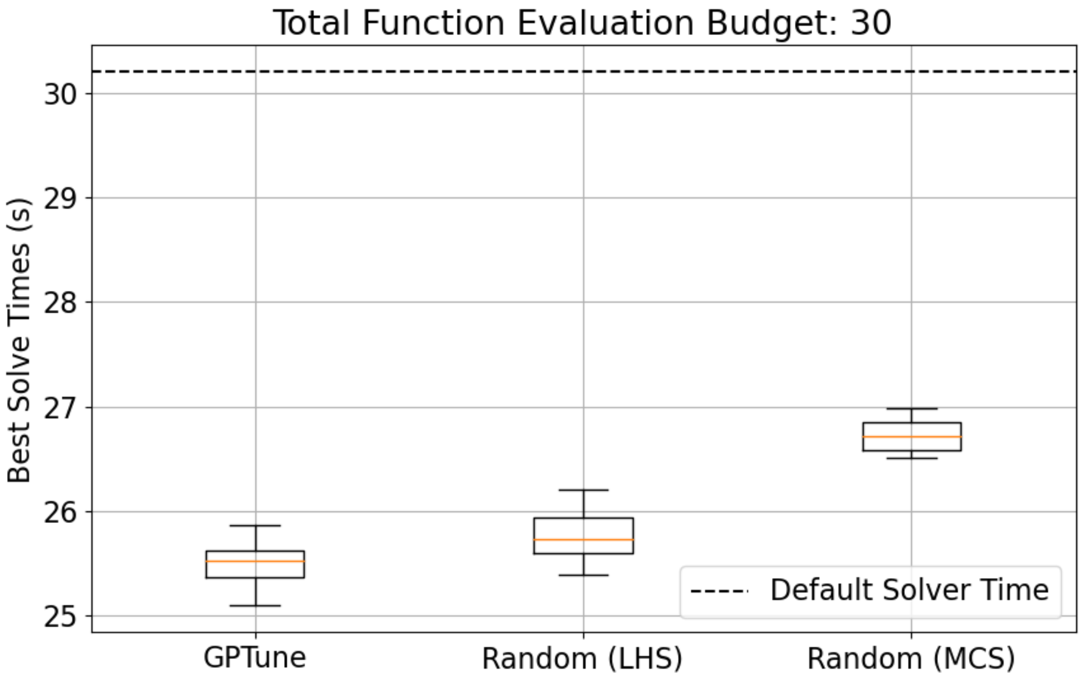

Figure 4: Comparison of best runtimes for `GPTune` Bayesian optimization against random search via Latin Hypercube Sampling and random search via Monte Carlo Sampling. Total function evaluation budget is fixed at 30 and 20 sample runs were conducted and reported for each method.

of parameters that result in runtimes of about 21 seconds (about a 1.5× speedup over our default settings). This time can be achieved by using a Chebyshev smoother for the fine grid with degree of 3, an eigenvalue ratio of about 26.85, and a maximum iteration count of 85 and then a two-stage Gauss-Seidel smoother for the coarse grids with 2 sweeps and an inner damping factor of about 0.8. One of the next steps of this work would be to extend these experiments to determine how likely each method is to find these extreme outliers.

4 Conclusions

With our automatic performance tuning framework, we have shown that we can reliably produce better overall runtimes for ALI than our previous random search method with speedups of up to 1.5 times greater than manually tuned smoothers. In addition to producing better runtimes, the tuning process requires fewer node-hours than our previous random search implementation since we are more likely to evaluate parameters that do not result in a failed Albany run. Finally, since the framework was developed with the `parsl` workflow management tool, we achieve a high degree of flexibility when it comes to mapping tuning tasks to available computing hardware. This flexibility allows easy integration of the tuning process into existing nightly testing for ALI and serves as a proof-of-concept for future workflow management of more complicated workflows such as running climate ensembles.

The results in this paper show that Bayesian optimization can result in better automatic performance tuning than random search but there are still a few potential avenues of exploration to further improve the tuning process. One thing to note is that in the many tuning runs we did to produce the results in this paper, the best total solve time (about 21 seconds) was only found on rare occasion and the vast majority of tuning runs resulted in a best solve time of about 25 seconds. One possible next step for this work is to quantify the rate at which outlier runtimes are found for each method which could potentially guide us to a method that more reliably finds these outlier runtimes. It would also be interesting to compare multiple tuning runs with a small function evaluation budget with a single tuning run with a budget equal to the sum of the single-run budgets.

In this paper, we have presented our exploration of the tunability of some of the linear solver parameters within the ALI model with respect to a single mesh and GPU architecture. Ultimately, we would like to use `GPTune`'s multi-task optimization capability to produce a performance model on small meshes that can be extended to predict optimal parameters for much larger meshes such as Antartica. Using a multi-task optimization approach, the two main questions we would seek to answer are: 1) are the optimal parameters for different size/shape meshes significantly different?, and 2) can the performance model for a small problem be used to reduce the total number of evaluations on a large mesh to produce optimal results? We also plan to leverage performance data collected nightly for Albany, which can help us define a prior performance model to potentially reduce the necessary number of function evaluations for performance tuning and enable automatic performance tuning in the presence of an ever changing model. This framework is also non-intrusive and can be used to tune much larger models including MALI and potentially E3SM.

Acknowledgments

Support for this work was provided through the SciDAC projects FASTMath and ProSPect, funded by the U.S. Department of Energy (DOE) Office of Science, Advanced Scientific Computing Research and Biological and Environmental Research programs. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231.

Disclaimer

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

References

[1] M. J. Hoffman, M. Perego, S. F. Price, W. H. Lipscomb, T. Zhang, D. Jacobsen, I. Tezaur, A. G. Salinger, R. Tuminaro, and L. Bertagna. Mpas-albany land ice (mali): a variable-resolution ice sheet model for earth system modeling using voronoi grids. *Geoscientific Model Development*, 11(9):3747–3780, 2018.

[2] Yang Liu, Wissam M. Sid-Lakhdar, Osni Marques, Xinran Zhu, Chang Meng, James W. Demmel, and Xiaoye S. Li. Gptune: Multitask learning for autotuning exascale applications. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '21, page 234–246, New York, NY, USA, 2021. Association for Computing Machinery.

[3] Xiaoye S Li, Paul Lin, Yang Liu, and Piyush Sao. Newly released capabilities in distributed-memory superlu sparse direct solver. 2022.

[4] Mauro Birattari. *F-Race for Tuning Metaheuristics*, pages 85–115. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[5] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[6] S.K. Smit and A.E. Eiben. Comparing parameter tuning methods for evolutionary algorithms. In *2009 IEEE Congress on Evolutionary Computation*, pages 399–406, 2009.

[7] Jesús-Adolfo Mejía-de Dios, Efrén Mezura-Montes, and Marcela Quiroz-Castellanos. Automated parameter tuning as a bilevel optimization problem solved by a surrogate-assisted population-based approach. *Applied Intelligence*, 51(8):5978–6000, aug 2021.

[8] Jae-Seung Yeom, Jayaraman J. Thiagarajan, Abhinav Bhatele, Greg Bronevetsky, and Tzanio Kolev. Data-driven performance modeling of linear solvers for sparse matrices. In *2016 7th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 32–42, 2016.

[9] Harshitha Menon, Abhinav Bhatele, and Todd Gamblin. Auto-tuning parameter choices in hpc applications using bayesian optimization. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 831–840, 2020.

[10] A. Salinger, R. Bartlett, A. Bradley, Q. Chen, I. Demeshko, X. Gao, G. Hansen, A. Mota, R. Muller, E. Nielsen, J. Ostien, R. Pawlowski, M. Perego, E. Phipps, W. Sun, and I. Tezaur. Albany: Using agile components to develop a flexible, generic multiphysics analysis code. *Int. J. Multiscale Comput. Engng.*, 14(4):415–438, 2016.

[11] William F. Spotz, Thomas M. Smith, Irina P. Demeshko, and Jeffrey A. Fike. Aeras: A next generation global atmosphere model. *Procedia Computer Science*, 51:2097–2106, 2015. International Conference On Computational Science, ICCS 2015.

[12] X. Gao, E. Nielsen, R. P. Muller, R. W. Young, A. G. Salinger, N. C. Bishop, M. P. Lilly, and M. S. Carroll. Quantum computer aided design simulation and optimization of semiconductor quantum dots. *Journal of Applied Physics*, 114(16):164302, 2013.

[13] WaiChing Sun, Jakob T. Ostien, and Andrew G. Salinger. A stabilized assumed deformation gradient finite element formulation for strongly coupled poromechanical simulations at finite strain. *International Journal for Numerical and Analytical Methods in Geomechanics*, 37(16):2755–2788, 2013.

[14] Alejandro Mota, Irina Tezaur, and Coleman Alleman. The schwarz alternating method in solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 319:19–51, 2017.

[15] Alejandro Mota, Irina Tezaur, and Gregory Phlipot. The schwarz alternating method for transient solid dynamics. *International Journal for Numerical Methods in Engineering*, 123(21):5036–5071, 2022.

[16] Jennifer Frederick, Alejandro Mota, Irina Tezaur, and Diana Bull. A thermo-mechanical terrestrial model of arctic coastal erosion. *Journal of Computational and Applied Mathematics*, 397:113533, 2021.

[17] I. K. Tezaur, M. Perego, A. G. Salinger, R. S. Tuminaro, and S. F. Price. Albany/felix: a parallel, scalable and robust, finite element, first-order stokes approximation ice sheet solver built for advanced analysis. *Geoscientific Model Development*, 8(4):1197–1220, 2015.

[18] John K. Dukowicz, Stephen F. Price, and William H. Lipscomb. Consistent approximations and boundary conditions for ice-sheet dynamics from a principle of least action. *Journal of Glaciology*, 56(197):480–496, 2010.

[19] Christian Schoof and Richard C. A. Hindmarsh. Thin-Film Flows with Wall Slip: An Asymptotic Analysis of Higher Order Glacier Flow Models. *The Quarterly Journal of Mechanics and Applied Mathematics*, 63(1):73–114, 01 2010.

[20] Jean-Christophe Golaz, Luke P. Van Roekel, Xue Zheng, Andrew Roberts, Jonathan D Wolfe, Wuyin Lin, Andrew Bradley, Qi Tang, Mathew E Maltrud, Ryan M Forsyth, and et al. The doe e3sm model version 2: Overview of the physical model. *Earth and Space Science Open Archive*, page 61, 2022.

[21] MPAS development team. MPAS Developers' Guide. 2013.

[22] The Trilinos Project Team. *The Trilinos Project Website*, 2020 (acccessed May 22, 2020).

[23] R. Tuminaro, M. Perego, I. Tezaur, A. Salinger, and S. Price. A matrix dependent/algebraic multigrid approach for extruded meshes with applications to ice sheet modeling. *SIAM Journal on Scientific Computing*, 38(5):C504–C532, 2016.

[24] Yadu Babuji, Anna Woodard, Zhuozhao Li, Daniel S. Katz, Ben Clifford, Rohan Kumar, Lukasz Lacinski, Ryan Chard, Justin M. Wozniak, Ian Foster, Michael Wilde, and Kyle Chard. Parsl: Pervasive parallel programming in python. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '19, page 25–36, New York, NY, USA, 2019. Association for Computing Machinery.

[25] Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, Dec 1998.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[27] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. Opentuner: An extensible framework for program autotuning. In *2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pages 303–315, 2014.

[28] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1437–1446. PMLR, 10–15 Jul 2018.

[29] Reiji Suda. *A Bayesian Method of Online Automatic Tuning*, pages 275–293. Springer New York, New York, NY, 2010.

[30] Kim Liegeois, Mauro Perego, and Tucker Hartland. Pyalbany: A python interface to the c++ multiphysics solver albany. *Journal of Computational and Applied Mathematics*, page 115037, 2022.

[31] Anubhav Jain, Shyue Ping Ong, Wei Chen, Bharat Medasani, Xiaohui Qu, Michael Kocher, Miriam Brafman, Guido Petretto, Gian-Marco Rignanese, Geoffroy Hautier, Daniel Gunter, and Kristin A. Persson. Fireworks: a dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience*, 27(17):5037–5059, 2015.

[32] Jerry Watkins, Max Carlson, Kyle Shan, Irina Tezaur, Mauro Perego, Luca Bertagna, Carolyn Kao, Matthew Hoffman, and Stephen Price. Performance portable ice-sheet modeling with mali, 04 2022.