THEME ARTICLE: COMPUTATIONAL MODELING OF ICE SHEETS AND GLACIERS

A Python Multiprocessing Approach for Fast Geostatistical Simulations of Subglacial Topography

Nathan W. Schoedl, Emma J. MacKie, Michael J. Field, Eric A. Stubbs, Allan Zhang, and Matthew Hibbs, *University of Florida, Gainesville, FL, 32611, USA*

Mathieu Gravey, *Austrian Academy of Sciences, 1010, Innsbruck, Austria*

Realistically rough stochastic realizations of subglacial bed topography are crucial for improving our understanding of basal processes and quantifying uncertainty in sea level rise projections with respect to topographic uncertainty. This can be achieved with sequential Gaussian simulation (SGS), which is used to generate multiple nonunique realizations of geological phenomena that sample the uncertainty space. However, SGS is very CPU intensive, with a computational complexity of O(Nk^3), where N is the number of grid cells to simulate, and k is the number of neighboring points used for conditioning. This complexity makes SGS prohibitively time-consuming to implement at ice sheet scales or fine resolutions. To reduce the time cost, we implement and test a multiprocess version of SGS using Python's multiprocessing module. By parallelizing the calculation of the weight parameters used in SGS, we achieve a speedup of 9.5 running on 16 processors for an N of 128,097. This speedup—as well as the speedup from using multiple processors—increases with N. This speed improvement makes SGS viable for large-scale topography mapping and ensemble ice sheet modeling. Additionally, we have made our code repository and user tutorials publicly available (GitHub and Zenodo) so that others can use our multiprocess implementation of SGS on different datasets.

Subglacial topography is a key parameter in ice sheet models used to make sea level rise projections.[1] The bed topography of the Greenland and Antarctic ice sheets has been extensively surveyed using airborne ice-penetrating radar.[2] However, there remain large gaps in measurements that must be interpolated. This interpolation is often performed using kriging, spline, or mass conservation[3] approaches, which all solve for the optimal bed elevation value at each spatial coordinate. These methods are deterministic, meaning they produce a single solution or interpolation. One major drawback of the deterministic approach is that it cannot sample the parameter space, making it difficult to determine how uncertainty in basal conditions is propagated in ice sheet models. Furthermore, these methods produce bed estimates that are smoother than the observed topography, which may bias interpretations of basal sliding processes.

The aforementioned issues can be resolved with geostatistical simulation, which is used to generate multiple realizations of topography that retain the roughness observed in radar measurements. Geostatistical simulation has previously been used to quantify uncertainty in hydrological and ice sheet models[1] and to investigate basal motion.[4] The ability of geostatistical simulation to create ensembles of equiprobable topographic realizations could be particularly advantageous for running ensemble ice sheet models to quantify uncertainty.

One of the most widely used geostatistical simulation methods is sequential Gaussian simulation (SGS), which treats spatial phenomena as a Gaussian process

Digital Object Identifier 10.1109/MCSE.2023.3317773
Date of publication 22 September 2023; date of current version 25 October 2023.

governed by spatial covariances.[5] These simulations are conditional, meaning they exactly match existing measurements. SGS is the stochastic version of the kriging algorithm, which uses the weighted average of nearby measurements to estimate the value at an intermediate coordinate. These weights are determined by a covariance function, which describes the variability of measurements as a function of their separation distance. SGS is implemented by 1) initializing a random order in which each grid cell will be simulated, 2) visiting a grid cell and using kriging to compute the mean and variance, 3) randomly sampling from the distribution described by the kriging mean and variance (this becomes the simulated value at that grid cell), 4) updating the conditioning data with the newly simulated value, and 5) repeating steps 2–4 until each grid cell has been visited and simulated. Each grid cell is visited and simulated sequentially to ensure that each value accounts for previously simulated values. See MacKie et al.[6] for detailed information on kriging and SGS.

While SGS software packages have historically been proprietary and are predominantly used in oil and gas exploration,[7] recent developments in open source geostatistics software in Python (e.g., `SciKit-GStat`[8] and `GStatSim`[6]) have improved the availability of these methods in cryosphere research. Despite these advances in accessibility, SGS remains difficult to use for large-scale ice sheet applications because its sequential nature makes it extremely computationally expensive. Specifically, generating S topographic realizations for a grid with N grid cells while using a maximum of k neighboring points to calculate the kriging weights is an $O(SNk^3)$-type problem.[9] This increase in runtime as the grid size increases makes SGS prohibitively computationally expensive for large-scale ice sheet problems, making it difficult to realize the potential of geostatistical simulation.

To address this computational issue, previous studies[9] have used a constant simulation path approach, where the grid cells are simulated in the same order for each realization to save computational time. However, this approach can lead to the underestimation of uncertainty. Alternatively, Nunes and Almeida[10] presented a parallelization strategy for SGS where the kriging weights are calculated in parallel. However, their implementation was tested on only four cores, is only compatible with the Windows operating system, and does not account for nonstationarity or variability in spatial statistics. As such, this approach is not well suited for the simulation of nonstationary subglacial topography, and the scalability has not been fully tested.

In this article, we implemented a Python multiprocess version of SGS following the multiprocessing approach of Nunes and Almeida,[10] which is designed to accommodate nonstationarity in subglacial topography. We tested the performance on up to 16 cores. To enhance the accessibility of this method, we have made our code repository readily available on GitHub[a] and Zenodo.[b] In this repository, we provide a script and user tutorial for generating simulations of subglacial topography. This script accepts the conditioning data, resolution, coordinate bounds, and number of realizations as inputs and then outputs topographic realizations. Here, we describe the parallelization process in more detail, quantify the improvement in model performance, and discuss the features of our tool.

METHODS

SGS Implementation

We implemented a modified version of SGS using methods available in `GStatSim` v1.0.5,[6] which was specifically designed for simulating subglacial topography. The topographic roughness is quantified using a covariance function, also known as the variogram, which measures the covariance between pairs of data points as a function of their separation distance. Rather than fitting one variogram to the entire region, the data are broken into different spatial clusters based on the density of measurements (see MacKie et al.[6] for details). This allows the topographic roughness to vary within a realization, which is important for capturing complex subglacial conditions or nonstationarity in the variogram statistics. Then, a variogram model is fit to the experimental variogram for the data in each cluster. The variograms are modeled using the exponential variogram function in the `Scikit-GStat` software package.[8] We use the exponential variogram type,[8] which is the most appropriate model for subglacial topography.[6]

The most computationally expensive step in this algorithm is the nearest neighbor octant search, which finds the k neighboring points to the grid cell being simulated such that they are evenly divided among the eight octants of the coordinate plane (see MacKie et al.[6] for details). This process is designed to reduce bias in irregularly sampled data and the subsequent calculation of the kriging weights. While SGS can be used with any type of kriging (ordinary, universal, cokriging, etc.), we use simple kriging in this study. Let $u_1, u_2, ..., u_k$ be the location of neighbors, u_0 be the location of the current grid cell, and $C_{m,n}$ be the covariance

[a]https://github.com/GatorGlaciology/SGS-topography-parallelization
[b]https://doi.org/10.5281/zenodo.7627029

between u_m and u_n. The kriging weights are determined by solving the following system for λ:

$$\begin{bmatrix} C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,1} & C_{3,2} & C_{3,3} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} C_{0,1} \\ C_{0,2} \\ C_{0,3} \end{bmatrix} \tag{1}$$

assuming $k = 3$. In this study, we use a k of 50. The matrix on the left describes the covariance between the conditioning data, and the term on the right is the covariance between the conditioning data and u_0. Note that the covariance calculation depends only on the relative locations of the k neighbors, not their values. Let Z be the variable that is being estimated. The kriging mean, Z^* is determined by computing the weighted sum of the elevation values

$$Z^*(u_0) = \sum_{\alpha=1}^{k} \lambda_\alpha Z(u_\alpha) \tag{2}$$

and the kriging variance $\sigma_E^2(u_0)$ is defined as

$$\sigma_E^2(u_0) = C(0) - \sum_{\alpha=1}^{k} \lambda_\alpha C_{0,\alpha} \tag{3}$$

where *C*(0) is the variance of data in the current variogram cluster. After these calculations, the simulated bed elevation value is determined by sampling from a normal distribution, defined by the kriging mean $Z^*(u_0)$ and variance $\sigma_E^2(u_0)$. This simulated value is then added to the conditioning data. The process repeats until all grid cells are simulated.

Multiprocessing Strategy

We improve the performance of the previously described SGS methodology by parallelizing key aspects of the workflow following the approach of Nunes and Almeida,[10] described in Figure 1. First, we parallelize the process of fitting variograms to the data in each spatial cluster. This is easy to do because the variogram of each cluster can be calculated independently. Parallelizing SGS itself is more challenging because previously simulated points are added to the conditioning data, making each iteration dependent on the previous iterations. This is demonstrated in (2) when some $Z(u_\alpha)$ are elevation values simulated during a previous iteration. As such, the traditional SGS approach fails the parallelization requirement of independence of processes.

The parallelization approach from Nunes and Almeida[10] solves this problem by isolating the steps of the algorithm where the kriging weights λ are computed. Recall that the nearest neighbor and kriging weight calculations depend only on the variogram parameters and the locations of conditioning data; the values of the conditioning data themselves are not

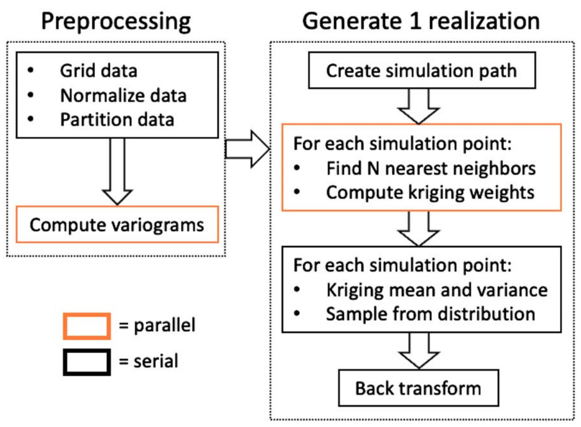
Preprocessing
Generate 1 realization
• Grid data
• Normalize data
• Partition data
Compute variograms
Create simulation path
For each simulation point:
• Find N nearest neighbors
• Compute kriging weights
For each simulation point:
• Kriging mean and variance
• Sample from distribution
Back transform
= parallel
= serial

FIGURE 1. Flowchart of the parallelized sequential Gaussian simulation (SGS) algorithm.

needed. As such, we can encapsulate the calculations for the kriging weights (1) in a function executed in parallel and prepare the parameters required for (2) and (3), which are faster to compute. To achieve this, we first define a random simulation order in which each grid cell is visited. Then, for each grid cell in the simulation path, the nearest neighbors are found, and the kriging weights are computed in parallel. These k nearest neighbors include the original conditioning data and any coordinates that will be simulated prior to u_0. The grid cell indices and weights of the neighbors are stored for later calculation of the kriging mean and variance. For each additional stochastic realization, a new random simulation path is generated, and the process repeats.

Speed Comparison

All simulations were tested on a 2021 Apple Mac Studio M1 Ultra with 20 cores. The Python `multiprocessing` module is used to distribute independent Python processes and data across a specified number of cores. We use bed elevation measurements for a 150 × 150-km^2 region in northwest Greenland from the Center for the Remote Sensing of Ice Sheets.[11] For visualization purposes, we also generate simulations for Pine Island Glacier (PIG) in West Antarctica using data from Frémand et al.[2]

We compared the performance of our parallelized algorithm with 16 processors to a single processor by running simulations with varying job sizes and recording the execution time. The number of simulation points decays approximately geometrically with resolution, so, to evenly sample job sizes, we varied the resolution geometrically from 200 m to 1200 m. Speedup for this comparison is defined as the serial algorithm's execution time divided by the parallel algorithm's execution time. Job size is defined by the number of simulated elevation values. Note that job size is a function of resolution size. During multiprocess runs, we recorded both the total execution time (including both serial and parallel components) and the time spent on calculations executed in parallel for later analysis.

Performance Analysis

We ran additional simulations specifically to perform scalability testing. Scalability is an algorithm's ability to increase in speedup with the number of processors. To perform this test, we ran two trials with the Greenland dataset at resolutions of 300 m and 1000 m and sequentially increased the number of processors from one to 16. For these trials, speedup (S) is defined as

$$S(p) = \frac{T(1)}{T(p)} \qquad (4)$$

where $T(1)$ is the execution time of our parallel algorithm with one processor, and $T(p)$ is the execution time of the parallel algorithm with p processors.

To interpret our results, the ideal speedup line (displayed in Figure 2) was used as a baseline for assessing scalability. However, ideal speedup is often unattainable in practice.[12] Ideal speedup is defined under the assumption that the execution time for a parallel algorithm with p processors should follow

$$T_{\text{ideal}}(p) = \frac{T(1)}{p} \qquad (5)$$

and, therefore, attain and a speedup of

$$S_{\text{ideal}}(p) = \frac{T(1)}{T_{\text{ideal}}(p)} = p. \qquad (6)$$

This metric does not account for the extra time required for parallel algorithms to perform process

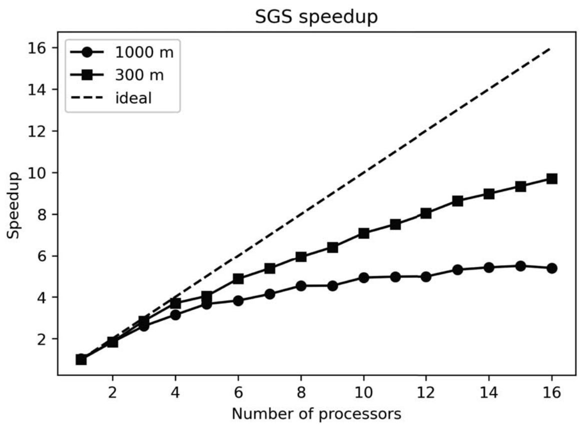

FIGURE 2. Speedup, defined by (4), with respect to the number of processors.

synchronization, known as parallel overhead. Taking the time for parallel overhead T_o into consideration, the execution time for algorithms that are strictly parallel can be defined by

$$T(p) = \frac{T(1) + T_o}{p}. \qquad (7)$$

We also used Amdahl's law to analyze the performance of our multiprocess implementation. Amdahl's law states that the performance of a parallel algorithm is limited by the percentage of time spent running a parallel process.[13] As such, the maximum theoretical speedup S_T of a parallel algorithm with p processors is

$$S_T(p, f) = \frac{1}{f + \frac{1-f}{p}} \qquad (8)$$

where f is the proportion of time spent performing serial calculations.

Additionally, we compared the empirical speedup to the idealized speedup line. According to Amdahl's law, the difference between the ideal speedup and the maximum empirical speedup D is given by

$$D = \frac{pf - f}{f + \frac{1-f}{p}}. \qquad (9)$$

Notice that, as the number of processors p increases, D is expected to increase. This is because, as p increases, the maximum theoretical speedup described by Amdahl's law becomes increasingly limited by the proportion of time performing serial calculations [f in (8)].

Visualization of Topographic Realizations

We visualized multiple realizations produced by our parallelized SGS algorithm using the `PyVista` library in Python.[14] The conditioning data were plotted to display the spatial distribution of the ice-penetrating radar measurements from the northwest Greenland and PIG data sets. From these data, we generated two SGS realizations for both locations. The Greenland simulation has a resolution of 400 m, and the PIG simulation has a resolution of 500 m. For comparison, we plotted the topography of BedMachine[3] for both locations at the same resolution size. BedMachine was derived using deterministic interpolation methods, including mass conservation and spline interpolation.

RESULTS

The runtimes for different simulation sizes using one versus 12 processors are shown in Figure 3(a). The resulting speedup is shown in Figure 3(b). The smallest speedup of 3.2 occurs at a 1,200-m resolution, which

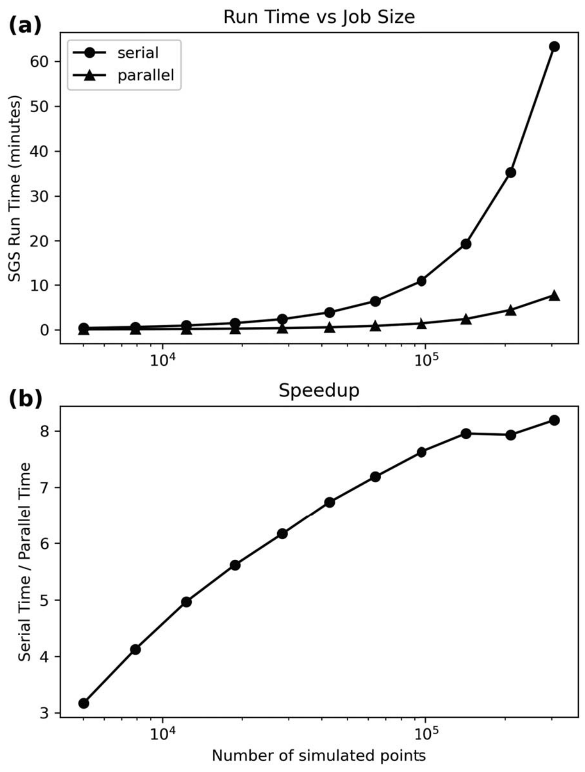

FIGURE 3. (a) Total execution time of the parallel and serial SGS simulations. (b) Speedup defined by (4). The x-axis uses a log scale and 12 processors were used to obtain these data.

produces the fewest simulation points, at 5022. This job took approximately 24 s on one processor and 7.6 s on 12 processors. The greatest speedup of 8.2 occurred at a 200-m resolution, yielding the largest number of simulation points at 306,811. This simulation took 1 hour 3 min on one processor and 7 min 43 s on 12 processors. For the 12 processors, we find that, as the job size increased from 5022 to 306,811 simulated points, the proportion of time performing serial calculations monotonically decreased from 24.9% to 5.2%.

Figure 2 displays the results from our scalability testing with speedup defined by (4). The 300-m runs simulated 128,097 missing points, and the 1000-m runs simulated 7899 missing points. The speedup for the 300-m-resolution runs is greater than that for the 1000-m runs. The 300-m-resolution simulation has a speedup of 9.5 when using 16 processors. Figure 4 provides a visualization for the topographic realizations generated by our multiprocess SGS algorithm, along with the BedMachine elevation model.[3]

We provide our parallelized SGS algorithm as an open source tool for users to run accelerated simulations on their own data sets. Our Python script and user tutorial are hosted on GitHub and Zenodo. The Python script provides a command line interface for users to enter a comma-separated value file (in polar stereographic coordinates) and specify the bounding coordinates, grid resolution, and number of simulations to run. For each realization, we return a color map visualization of the modeled topography and the associated simulation data. The user tutorial is a Jupyter notebook that also enables the user to provide their own data; however, its intent is to provide a more detailed understanding of our multiprocess implementation. The notebook also provides an interactive visualization of the simulated data using `PyVista`.[14]

DISCUSSION

Our results show that multiprocessing performance increases with job size. This phenomenon can be explained by Amdahl's law because job size has a direct effect on the proportion of time performing serial calculations [f in (8)]. With the number of processors held constant, we found that, as the job size increases, f decreases monotonically. Note that in (8), S_T and f are inversely proportional; therefore, a decreasing f as job size increases results in an increase in speedup.

Our multiprocess algorithm significantly reduces the time cost barrier to generating multiple high-resolution simulations. These results point toward a feasible path for simulating entire ice sheets. While Figure 2 shows that there are diminishing returns when increasing the number of processors, this effect can be explained by the increasing difference between the theoretical maximum speedup and the ideal speedup as the number of processors increases [see (9)]. Furthermore, this effect is lessened for larger simulations, as demonstrated by Figure 3. This suggests that, for ice-sheet-scale simulations, which would involve simulating tens of millions of grid cells, a larger number of processors may be effective. The speed of this method could be further improved by performing the parallel component of this algorithm on a GPU. Future work is needed to test our algorithm's scalability using high-performance computing clusters and GPU acceleration.

Our code repository with our multiprocessing Python script and user tutorials are publicly available on GitHub and Zenodo. This open source software will make it easy to incorporate geostatistical simulation into various ice sheet investigations. For example, this tool could be used to generate ensemble realizations of subglacial hydrological models.[15] These can be used to quantify uncertainty in the locations of subglacial flow paths.

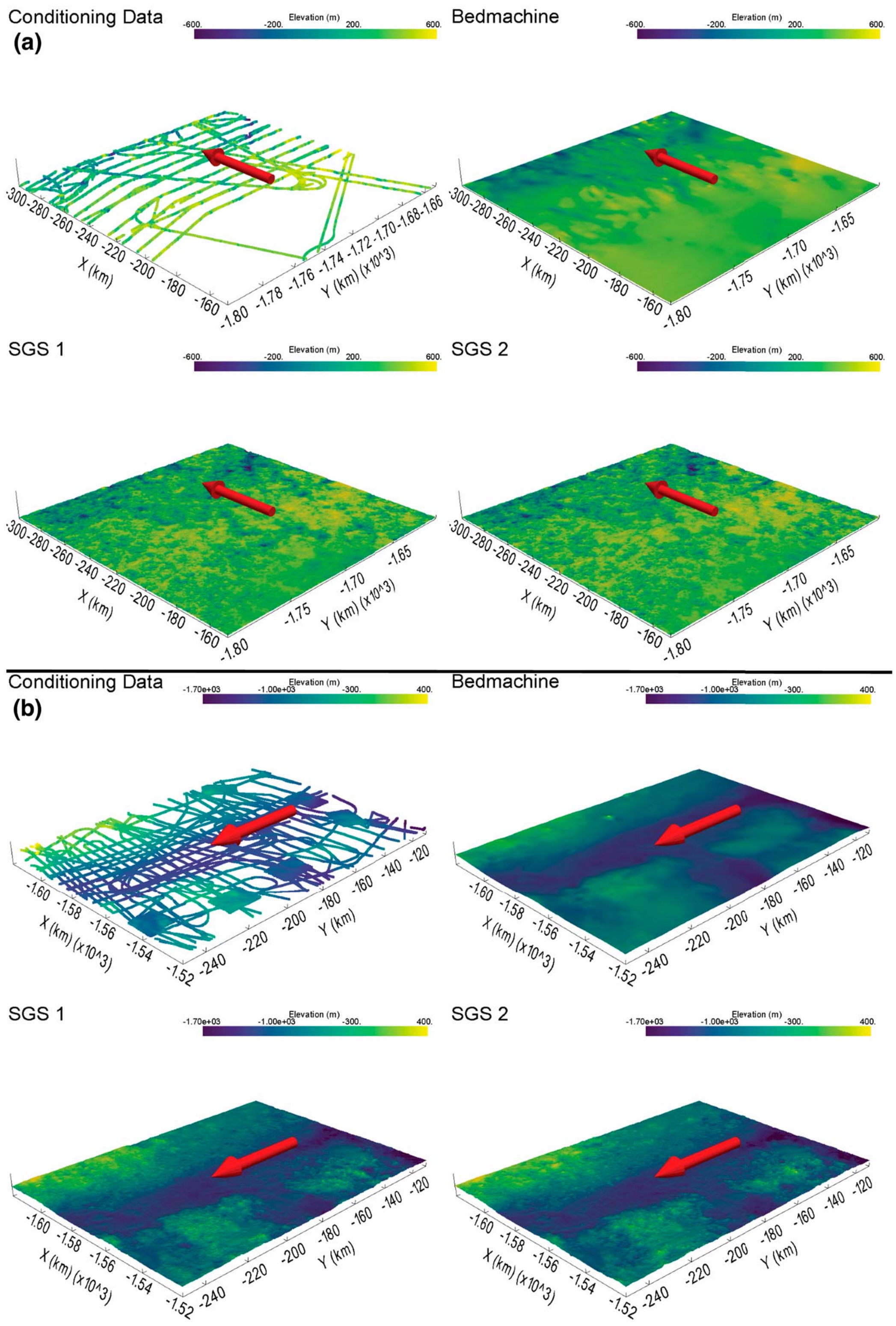

FIGURE 4. Ice-penetrating radar measurements, deterministic interpolation using BedMachine, and parallelized SGS interpolations for (a) northwest Greenland and (b) Pine Island Glacier with arrows indicating the direction of ice flow.

Additionally, simulated topography can be used in ice sheet models to investigate basal sliding processes[4] or quantify uncertainty in ice sheet models with respect to uncertainty in bed topography.[1] The case studies by Law et al.[4] and Wernecke et al.[1] used a small number of SGS topographic realizations with relatively small spatial domains. Our SGS speed improvements, coupled with advances in ice-sheet-modeling methods, could help facilitate the implementation of large ensembles of ice sheet models at regional or ice sheet scales.

CONCLUSION

The geostatistical simulation of realistically rough bed topography is imperative for accurately characterizing basal processes and quantifying uncertainty in ice sheet models. However, the inherently sequential nature of SGS makes this algorithm difficult to use at large scales or for large ensembles. Our multiprocessing implementation significantly reduces the time cost associated with generating simulations. Furthermore, by making this tool open source with well-documented user tutorials, we have reduced the barrier to spinning up SGS models for use with ice sheet workflows. This will enable SGS to be integrated with a variety of ice sheet analyses, including uncertainty quantification in ice sheet models.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation Geosciences Open Science Ecosystem Award 2324092.

REFERENCES

1. A. Wernecke, T. L. Edwards, P. B. Holden, N. R. Edwards, and S. L. Cornford, "Quantifying the impact of bedrock topography uncertainty in Pine Island Glacier projections for this century," *Geophys. Res. Lett.*, vol. 49, no. 6, Mar. 2022, Art. no. e2021GL096589, doi: 10.1029/2021GL096589.
2. A. C. Frémand et al., "Antarctic Bedmap data: Findable, accessible, interoperable, and reusable (FAIR) sharing of 60 years of ice bed, surface, and thickness data," *Earth Syst. Sci. Data*, vol. 15, no. 7, pp. 2695–2710, Jul. 2023, doi: 10.5194/essd-15-2695-2023.
3. M. Morlighem et al., "BedMachine v3: Complete bed topography and ocean bathymetry mapping of Greenland from multi-beam echo sounding combined with mass conservation," *Geophys. Res. Lett.*, vol. 44, no. 21, pp. 11,051–11,061, Sep. 2017, doi: 10.1002/2017GL074954.
4. R. Law, P. Christoffersen, E. MacKie, S. Cook, M. Haseloff, and O. Gagliardini, "Complex motion of Greenland Ice Sheet outlet glaciers with basal temperate ice," *Sci. Adv.*, vol. 9, no. 6, May 2022, Art. no. eabq5180, doi: 10.31223/X5MM1K.
5. C. V. Deutsch and A. G. Journel, *Geostatistical Software Library and User's Guide*, vol. 8. New York, NY, USA: Oxford Univ. Press, 1992.
6. E. MacKie et al., "GStatSim V1.0: A Python package for geostatistical interpolation and conditional simulation," *Geoscientific Model Develop.*, vol. 16, no. 13, pp. 3765–3783, 2023, doi: 10.5194/gmd-16-3765-2023.
7. M. J. Pyrcz and C. V. Deutsch, *Geostatistical Reservoir Modeling*. New York, NY, USA: Oxford Univ. Press, 2014.
8. M. Mälicke, "SciKit-GStat 1.0: A SciPy-flavored geostatistical variogram estimation toolbox written in Python," *Geoscientific Model Develop.*, vol. 15, no. 6, pp. 2505–2532, Mar. 2022, doi: 10.5194/gmd-15-2505-2022.
9. R. Nussbaumer, G. Mariethoz, M. Gravey, E. Gloaguen, and K. Holliger, "Accelerating Sequential Gaussian Simulation with a constant path," *Comput. Geosci.*, vol. 112, pp. 121–132, Mar. 2018, doi: 10.1016/j.cageo.2017.12.006.
10. R. Nunes and J. A. Almeida, "Parallelization of sequential Gaussian, indicator and direct simulation algorithms," *Comput. Geosci.*, vol. 36, no. 8, pp. 1042–1052, Aug. 2010, doi: 10.1016/j.cageo.2010.03.005.
11. CReSIS, *Radar Depth Sounder Data*. Lawrence, KS, USA: Digital Media, 2021. [Online]. Available: http://data.cresis.ku.edu/
12. A. Y. Grama, A. Gupta, and V. Kumar, "Isoefficiency: Measuring the scalability of parallel algorithms and architectures," *IEEE Parallel Distrib. Technol., Syst. Appl.*, vol. 1, no. 3, pp. 12–21, Aug. 1993, doi: 10.1109/88.242438.
13. G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities, reprinted from the AFIPS conference proceedings, vol. 30 (Atlantic City, N.J., Apr. 18–20), AFIPS Press, Reston, Va., 1967, pp. 483–485, when Dr. Amdahl was at International Business Machines Corporation, Sunnyvale, California," *IEEE Solid-State Circuits Soc. Newslett.*, vol. 12, no. 3, pp. 19–20, Summer 2007, doi: 10.1109/N-SSC.2007.4785615.
14. B. Sullivan and A. Kaszynski, "PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK)," *J. Open Source Softw.*, vol. 4, no. 37, May 2019, Art. no. 1450, doi: 10.21105/joss.01450.
15. E. J. MacKie, D. M. Schroeder, C. Zuo, Z. Yin, and J. Caers, "Stochastic modeling of subglacial topography exposes uncertainty in water routing at Jakobshavn Glacier," *J. Glaciol.*, vol. 67, no. 261, pp. 75–83, 2021, doi: 10.1017/jog.2020.84.

NATHAN W. SCHOEDL is an undergraduate student at the University of Florida, Gainesville, FL, USA, pursuing a triple bachelor of science degree in computer science, statistics, and mathematics. His research interests include machine learning, geostatistics, and high-performance computing. Contact him at nathanschoedl@ufl.edu.

EMMA J. MACKIE is an assistant professor in the Department of Geological Sciences, University of Florida, Gainesville, FL, 32611, USA. Her research interests include geophysical glaciology, stochastic modeling, and glacial geology. Mackie received her Ph.D. degree in geophysics from Stanford University. She is a Member of IEEE. Contact her at emackie@ufl.edu.

MICHAEL J. FIELD is a Ph.D. student in the Department of Geological Sciences, University of Florida, Gainesville, FL, 32611, USA. His research interests include the applications of geostatistical modeling and Bayesian inversion to cryospheric problems. Field received his B.S. degree in geophysical engineering from Colorado School of Mines. Contact him at michael.field@ufl.edu.

ERIC A. STUBBS is a research facilitator and software engineer at Information Technology Research Computing, University of Florida, Gainesville, FL, 32611, USA. His research interest is high-performance computing. Stubbs received his Ph.D. degree in agricultural education from the University of Florida. Contact him at ericeric@ufl.com.

ALLAN ZHANG is an undergraduate student at the University of Florida, Gainesville, FL, 32611, USA, pursuing a double bachelor of science degree in data science and mathematics. His research interest is geostatistical modeling. Contact him at zhangallan@ufl.edu.

MATTHEW HIBBS is an undergraduate student in the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL, 32611, USA, where he is studying computer science and statistics. His research interest is high-performance computing. Contact him at hibbs.matthew@ufl.edu.

MATHIEU GRAVEY is a researcher in the Institute for Interdisciplinary Mountain Research, Austrian Academy of Sciences, 1010, Innsbruck, Austria. His research interests include geostatistics, high-performance computing, remote sensing, and machine learning. Gravey received his Ph.D. degree in geostatistics from the Institute of Earth Surface Dynamics at the University of Lausanne. Contact him at research@mgravey.com.