ParCFD'2020
32^{nd} International Conference on Parallel Computational Fluid Dynamics
May-11-13 2020, Nice FRANCE

DASK PARALLELIZED GEOPHYSICAL INVERSIONS USING SIMPEG

JOSEPH R. CAPRIOTTI*, LINDSEY J. HEAGY†, AND DOMINIQUE FOURNIER^

*Geophysical Inversion Facility, University of British Columbia
Department of Earth, Ocean and Atmospheric Sciences
Vancouver, BC Canada V6T 1Z4
e-mail: josephrcapriotti@gmail.com

†UC Berkeley, Department of Statistics
367 Evans Hall, University of California
Berkeley, CA 94720-3860
email: lindseyheagy@gmail.com

^Mira Geoscience Ltd.
Vancouver, BC Canada V6C 1T2
email: dominiquef@mirageoscience.com

Key words: Parallel inversion, potential fields, resitivitiy, electromagnetics

Abstract. The goal of the SimPEG package is to provide modeling and inversion tools for geophysical datasets. As our users apply larger and larger data sets, we've identified areas and pathways for scaling SimPEG to larger clusters. These primarily exploit the natural parallelism from solving partial differential equations with multiple independent source terms. We further parallelize the problem with the use of a localized meshing approach. Dask allows us to develop these algorithms independent of the computing environment and monitor them as they are running.

1 INTRODUCTION

SimPEG was designed to be a tested open-source resource for geophysicists looking to tackle common parameter estimation problems [1]. This Python package was developed with the inverse modeling in mind, mostly focussing on simulating and inverting potential fields, electrical, and electromagnetic data. It also supports straight ray tomography and fluid-flow modeling; extending support to other methods is a welcome area for contribution.

SimPEG is used by many different groups with a range of programming skills and knowledge. Undergraduate students use it to explore the fundamentals of various geophysical surveys, e.g. visualizing electromagnetic fields in conductive media.. Researchers use it to develop new inversion methodologies and extend SimPEG functionality. Industry members test the scalability of the code to large airborne surveys. The package enables an open-source community to work together to tackle some of the higher level problems

without having to completely understand the intricacies of implementing derivatives. It is designed to be modular and follow a framework that allows the users to plug in different aspects of a problem to suit their needs.

Most SimPEG simulation capability is designed around solving the elliptical or parabolic partial differential equations (PDE) describing the geophysical systems, and minimizing functions involving them. In general, we solve the PDEs to obtain predicted data, in a manner that is generalizable to the discretization scheme. There are also a subset of the geophysical methods that have analytic or integral implementations, including potential field methods. Our inversion mechanics are primarily focused around Gauss-Newton like methods, which require operators for Jacobian-vector products. This enables us to define derivatives and approximate Hessians operators necessary to minimize non-linear least squares functionals.

2 PARALLELIZATION STRATEGIES

SimPEG is typically used to solve underdetermined geophysical inversion problems,, where the number of data is less than the number of unknown parameters. This leads to a regularized minimization function:

$$\phi(m) = ||W_d \left(d_{obs} - \mathcal{F}[m]\right))||^2 + \beta||W_m(m - m_{ref})||^2. \quad (1)$$

We approach the minimization operation using the Gauss-Newton strategy [2], where we solve a system of equations to obtain a step direction at each iteration, k,

$$(J^T W_d^T W_d J + \beta W_m^T W_m)\delta m = J^T W_d^T W_d(d_{obs} - \mathcal{F}[m_k]) - \beta W_m^T W_m(m_k - m_{ref}), \quad (2)$$

where J is the Jacobian of the forward operation, $\mathcal{F}$. We solve equation 2 by conjugate gradient (CG) which requires us to define operators for J and J^T times a vector. A valid SimPEG invertible simulation knows how to predict its data, how to multiply the Jacobian times a vector, and the adjoint operation of a Jacobian transpose times a vector, which are the operations we focus on parallelizing.

For the numerical PDE based simulations in SimPEG, (i.e. direct current resistivity, or time/frequency domain electromagnetics), there are often many sources with different locations and frequencies. The PDE forward solutions and Jacobian-vector products of these problems require repeated sparse matrix solutions for many different right-hand side source terms. This is the clearest embarrassingly parallel portion of SimPEG, and indeed most sparse matrix solvers will perform this in parallel when given multiple right hand sides. We can break each repeated solve operation up amongst the sources in parallel.

These operations assume that we can solve the sparse system of equations on a single machine, but with large models this becomes infeasible. For each observation point (or a small collection of them), we can create a mesh that is accurate locally, but coarsens away from that point [3]. This allows us to accurately compute solutions for each receiver, while reducing the computational requirements of sparse matrix solutions, or reduce time spent calculating analytic functions (Figure 1). This approach is independant for each observation point and has been used to parallelize large scale inversions [4].

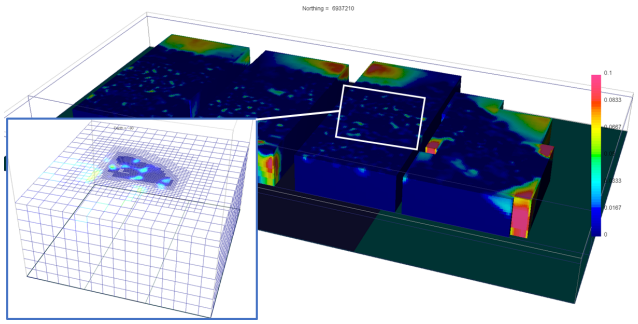

Figure 1: A local OcTree mesh for a set of magnetic observations. The inversion is solved on the entire domain at the finest level, but sensitivities and forward solutions are computed on the many local meshes in parallel.

Another area where inversion can be parallelized is the explicit formation of the Jacobian matrix. If there is enough storage space to hold the matrix either in memory or on disk, there are potential time savings. Forming the Jacobian matrix requires a matrix solution for each observed data point, which is also embarrassingly parallel. Even though this may seem like more operations compared to the implicit operation, it scales well on clusters and can be advantageous when performing increased CG iterations to solve equation 2. We also have access to improved preconditioners for CG, making the iterations more effective.

3 Dask

Having identified these areas of potential scalability for SimPEG, we chose to implement the parallel portions using Dask [5]. Dask consists of two parts: a dynamic task scheduler that distributes operations across clients and parallel data collections. SimPEG, like many scientific python packages, makes heavy use of numpy operations. Dask is designed to mimic the numpy API which allows us to integrate it into the SimPEG framework with few changes. Dask will analyze computations involving distributed arrays to create a task graph that can be executed in parallel and will handle communication between clients when necessary to complete those tasks. Dask's parallel data collections allow for arrays to be distributed across multiple files, integrating capability for compressed array storage, or in distributed ram. Taken together, these allow for SimPEG's simulations to scale across clusters using the parallelization strategies we previously identified.

Most of our users interact with the package through Jupyter where they create their Python code in interactive environments, monitor their running code, or inspect variables during execution. One of our parallelization goals with SimPEG is to keep this functionality when scaling from local machines up to larger cluster computers, with minimal code changes. Dask decouples the API for parallelization from the type of hardware it runs on, which simplifies the implementation and the porting of prototype-code in scripts and Jupyter notebooks from a local machine to a cluster or to the cloud using Dask-Jobque or Dask-Kubernetes, respectively. From a user-perspective, only the definition of the cluster where you will run your code changes. Dask's schedulers also allow users to code inter-

actively on those machines within Jupyter notebooks and monitor task distribution and execution times (Figure 2).

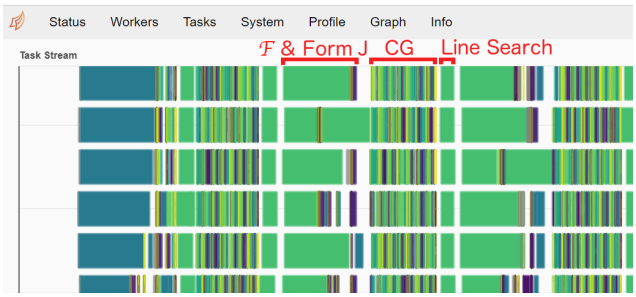

Figure 2: You can monitor currently running tasks and visualize how they are distributed and executed over time. This graph will show any communication time in red, operations will be color coded to identify common calls, and white space represents dead time.

4 CONCLUSIONS

SimPEG is in the early stages of being scaled to high performance computing. We have identified a few possibilities to extend the package for our inversion mechanics to perform well on clusters, parallelizing over different source locations and frequencies when solving the numerical PDEs, as well as decomposing the inversion domain into smaller local meshes that can be computed in parallel. Implementing these with Dask has allowed our users to continue to interact with the code in a manner they are accustomed to. Preliminary results show good task distribution for these operations with minimal communication overhead.

REFERENCES

[1] Cockett, R., Kang, S., Heagy, L.J., Pidlisecky, A., and Oldenburg, D.W. SimPEG: An open source framework for simulation and gradient based parameter estimation in geophysical applications. *Computers & Geosciences*, (2015) **85**, Pages 142-154.

[2] Oldenburg, D.W. and Li Y. *Inversion for applied geophysics: A tutorial*. Near Surface Geophysics, (2005), Pages 89-150.

[3] Yang, D., Oldenburg, D.W., and Haber, E. 3-D inversion of airborne electromagnetic data parallelized and accelerated by local mesh and adaptive soundings. *Geophysical Journal International*, (2014) Vol. 196, Issue 3, Pages 1492–1507.

[4] Haber, E. and Schwarzbach C. Parallel inversion of large-scale airborne time-domain electromagnetic data with multiple OcTree meshes. *Inverse Problems*, (2014) **30**:055011.

[5] Dask Development Team (2016). Dask: Library for dynamic task scheduling URL https://dask.org.