Karina Santoso COURAGE REU

7 August 2020 Professor Chen

Sparse Matrices in Optimizing Model Efficiency

Overview

As a part of Clemson University's Summer 2020 COURAGE REU program, I worked with Professor Qingshan Chen on a project regarding the computation of geophysical flows. This project worked with the Mathematical Ocean Prototype, which is a Python model that calculates many different aspects of the ocean over time, such as changes in energy, enstrophy, and mass. The model can be run using a variety of different grid sizes, which divide up the surface of the Earth into little cells and are used one at a time to view different resolutions. A higher grid size corresponds to a higher number of cells used to cover the Earth's surface and therefore is more precise, but requires more calculations and time to run the model over a certain time frame.

The model utilizes matrix multiplication to carry out some of its calculations, most notably in one of its methods called update in the EllipticCpl2 class. When looking at the profiling results of the model, which showed a summary of the amount of time used by each function when running the model, a significant portion of time was consistently taken up by matrix multiplications. During the duration of the REU program, we aimed to implement modifications in this EllipticCpl2 class to attempt to decrease the time taken by the matrix multiplication methods, and subsequently the runtime of the model as a whole.

Methods

The matrices involved in the computations in the model are extremely large, and only get larger with higher resolution grid sizes. For example, the smallest grid size that the model works with is 2562, in which 2562x2562 matrices are used. Therefore, as we use higher resolution grid sizes, saving these large matrices will take up lots of memory. Since a relatively small portion of the entries in these matrices are nonzero, the matrices are stored in the form of sparse matrices. Instead of using memory to save every entry of the matrix, sparse matrices only save the nonzero entries of the matrix and their positions and are very beneficial for large matrices with mostly zero entries, such as these.

Before any modifications were made, all the matrices in the EllipticCpl2 class were stored in the form of Compressed Sparse Row (CSR) matrices, which is a type of sparse matrix. After being converted to a CSR matrix, the matrix retains its same number of rows, and each row of the CSR stores the positions and values of the nonzero entries in the corresponding row of the original matrix. Therefore, memory access of any given single position is extremely efficient, as is accessing the rows of the matrix. However, accessing the columns of the matrix is difficult, as each row of the CSR matrix would need to be searched at the column's position to determine if the entry is nonzero and access its value if that is the case.

Since matrix multiplication is carried out by multiplying the rows of the first matrix by the columns of the second matrix, and all matrices are currently being stored in the CSR format, accessing the columns of these second matrices could be a source of inefficiency as it is an expensive process. Compressed Sparse Column (CSC) matrices work in exactly the same way as CSR matrices, but the positions and values of the nonzero entries are stored by column, instead of by row. Similarly, accessing columns of CSC matrices is much easier than accessing its rows. Therefore, by changing the format of the second matrices involved in matrix multiplications, the time used to complete these calculations may be able to be decreased.

After looking at the different instances of matrix multiplication in the update method of the EllipticCpl2 class, I determined that there were four matrices consistently being used as the second matrix in matrix multiplication: vc.mSkewgrad_td, vc.mGrad_n_n, self.GN, and self.SN. Using the tocsc() function on each of these matrices in the class constructor, the format of these matrices were changed from CSR to CSC. Copies were made of the vc.mSkewgrad_td and vc.mGrad_n_n matrices before these changes were made, as to not modify the original vc object from a different class. Additionally, these changes were made in the EllipticCpl2 constructor instead of the update function itself to reduce the number of times these changes had to be made and increase efficiency, as the update method is called many more times than the constructor.

Results

After implementing these changes in the EllipticCpl2 file, I ran multiple trials of the model with various grid sizes, comparing the output and profiling results with that of the original model before these changes were made. The model still seemed to run its calculations accurately, producing the exact same output as the original. Additionally, initial test results with the smallest two grid sizes seemed promising, with run times about the same or slightly less than the original version. However, as we progressed up to the larger grid sizes and continued to run more trials on the smaller grid sizes, we saw that at times the new version ran faster, sometimes the old version ran faster, and sometimes their run times were almost identical. When averaging the run times of the different trials with both versions, they appeared to be about the same and had no significant improvements or deteriorations in time consumed.

As no consistent or notable differences in run time were observed with the different grid sizes, as well as with running the model with different linear solvers, no general conclusions can be drawn about the effectiveness of the alterations made to change some matrices from CSR format to CSC. The functions of the model are carried out by many different operations of Python and Scipy, much of which happen behind the scenes. It is also difficult to identify in the profiling results, which includes hundreds of different methods carried out during the model's calculations, which items correspond to specific sections of written code. Possibly by examining different functions in the profiling results outside of the matrix multiplication to see if any of them were affected by the CSC changes, or by similarly altering lines of code in a different class or function in the model to see if a more substantial change in matrix multiplication runtime can be observed, a method can be found to decrease the run time of these matrix multiplication functions. These ideas are left for future endeavors.