Exploring Performance of GeoCAT data analysis routines on GPUs

1st Haniye Kashgarani
University of Wyoming
hkashgar@uwyo.edu

2nd Cena Miller
National Center for Atmospheric Research
cmille73@ucar.edu

3rd Supreeth Suresh
National Center for Atmospheric Research
ssuresh@ucar.edu

4th Anissa Zacharias
National Center for Atmospheric Research
anissaz@ucar.edu

***Abstract*—The GeoCAT-comp program is a Python toolkit used by the geoscience community to analyze data. This project explores ways to port GeoCAT-comp to run on GPUs, as recent supercomputers are shifting to include GPU accelerators as the major resource. Although GeoCAT-comp's routines are all sequential or utilize Dask parallelization on the CPU, the data processing is embarrassingly parallel and computationally costly, enabling us to optimize using GPUs. GeoCAT uses NumPy, Xarray, and Dask arrays for CPU parallelization. In this project, we examined different GPU-accelerated Python packages (e.g., Numba and CuPy). Taking into account the deliverability of the final porting method to the GeoCAT team, CuPy is selected. CuPy is a Python CUDA-enabled array backend module that is quite similar to NumPy. We analyzed the performance of the GPU-accelerated code compared to the Dask CPU parallelized code over various array sizes and resources, and through strong and weak scaling.**

***Index Terms*—GPU porting, massive parallelization, GPU array backends, CuPy**

I. Introduction

Classically, we run all our programs on the CPU. CPU cores have a fast clock cycle but a limited number of cores. In contrast, GPU cores have slower clock cycles, but have thousands of cores. With GPUs, problems can be sliced into thousands of simple tasks and massive parallelization can improve performance. A major objective of this project is to port GeoCAT routines to run on GPU.

GeoCAT [1] is a python toolkit for the Geoscience community for data analysis. GeoCAT-comp is one of the GeoCAT repositories, containing previous NCAR Common Language (NCL) functionality and other geoscience analysis functions written in Python. Built on the Pangeo, it uses Xarray [3] to have multi-dimensional labeled arrays, and Dask [4] for CPU parallelization. The routines of geocat-comp are either sequential or take advantage of Dask to parallelize on CPUs. However, data processing and data analysis is an embarrassingly parallel task, and it's computationally intensive. GPUs can be used to accelerate the task through massive parallelization. The project focused on meteorology.py and crop.py in GeoCAT-comp, which exist in GeoCAT-comp's Github repository.

GPU programming with NVIDIA devices uses CUDA programming, which is a heterogeneous programming model that uses both CPU and GPU. In C++ CUDA code, we have to manually manage memory in both CPU and GPU, copy the CPU variables to GPU memory, and also manually launch our kernel function. This makes CUDA C++ programming a challenging task. In Python, the same code can be written in a few lines using CuPy, a GPU array backend [2]. The CuPy backend will take care of these steps automatically.

II. Methodology and Results

A. CuPy

Different Python packages can help programmers use GPUs to accelerate their code. The library we used in porting GeoCAT is CuPy [2],)which implements a subset of NumPy's interface, and it can be used as a drop-in replacement for NumPy. CuPy supports both Nvidia CUDA and AMD ROCm frameworks. NumPy executes the functions on the CPU, while CuPy executes them on the GPU.

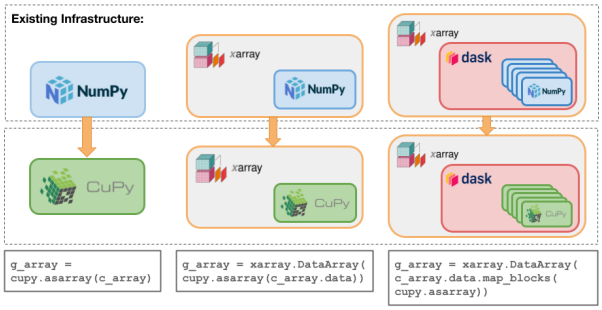

Fig. 1. Examples of Array Conversion.

As a result of drop-in replacement of NumPy with CuPy, several GeoCAT-comp functions were ported and the routines' performance was measured on both CPU and GPU. There are both pros and cons to the drop-in replacement technique. CuPy drop-in replacement offers different optimization techniques:

- "CUB environment variable" which can optimize the GPU performance for a reduction function, and
- "cutensor environment variable" which accelerates the tensor operations.

Limitations:

- The speedup of CuPy can be limited to large arrays and ndarrays.
- In addition, some tasks, such as searching, may not be optimally performed on a GPU.

B. Array Conversion

Input variables for the GeoCAT-comp routines include 1) NumPy, 2) Xarray with NumPy as its data, and 3) Xarray with Dask as its data, which include chunks of NumPy arrays. We converted NumPy to CuPy so that the Xarray and Dask would contain CuPy instead of NumPy. Figure 1 illustrates the results of converting the different types of data. All ported codes are available on Github [7].

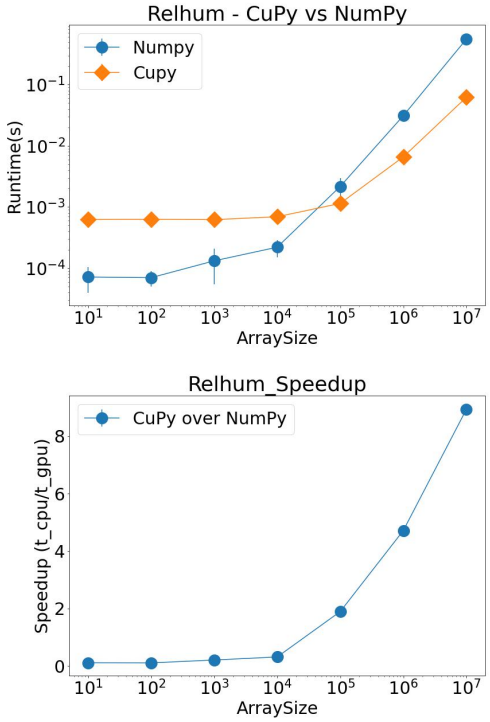

Fig. 2. Performance Comparison and Speedup

C. Performance Comparison

Next, we took a closer look at the five functions within the meteorology.py. With randomly generated data, we created arrays of different sizes and compared their computational performance on the CPU and GPU. Figure 2 provides a log scale plot of the runtime for the Relhum function using GPUs and CPUs. Orange represents CuPy or GPU computation, while blue represents NumPy. When the array size was 10^4 and smaller, the CPU did the computation faster. When we used an array size of 10^5 or larger, the GPU outperformed the CPU.

The bottom plot in Figure 2 shows the speed up for the same function. Using the GPU accelerated the computation by about 9x when the array size was 10^7. We observed the same trendline for the other four functions in meteorology.py that have been ported to GPU.

D. Scalability

Scalability is important in parallel computing to indicate how well software can deliver more computational performance as the amount of resources increases. NVIDIA Tesla V100 GPUs and CPU nodes with 36 cores were used for these experiments. Hardware specifications:

- GPU nodes: 2 18-core 2.3-GHz Intel Xeon Gold 6140 (Skylake) processors per node 8 NVIDIA Tesla V100 32GB SXM2 GPUs with NVLink
- CPU nodes: Dual-socket nodes, 18 cores per socket 2.3-GHz Intel Xeon E5-2697V4 (Broadwell) processors 16 flops per clock

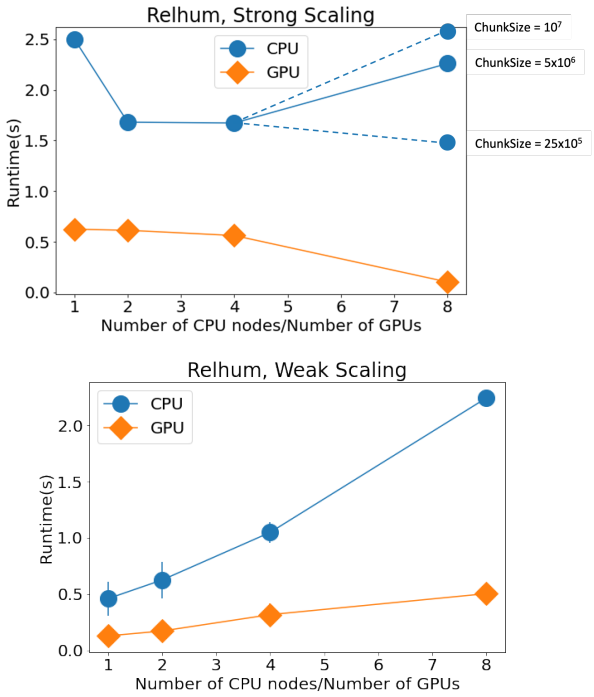

Fig. 3. Strong and Weak Scaling

Strong scaling means increasing CPU or GPU nodes and measuring the runtime by fixing the array size. For a fixed problem size, strong scaling measures speedup according to the number of processors. We maintained an array size of 10^7. In Figure 3, with the GPU computation running on 8 GPUs, we saw speedup and runtime decrease by 6x. A very important thing to keep in mind when using Dask for CPU parallelization is the chunk size. For the CPU, we need to tune this parameter based on the problem and the number of cores we have since very small or large chunks result in longer runtimes. As a result, we observed an effect on performance when using different chunk sizes. GPUs, however, can handle parallelization by simply setting the chunk size equal to the array size.

We also performed weak scaling starting from an array size of 10^6. In weak scaling, the speedup is measured in relation to the number of processors for a scaled problem size. Therefore, when we increase the number of resources, we multiply the array size by the same number. Figure 3 indicates that CPU computations result in more overhead when the number of processors increases.

REFERENCES

[1] Visualization Analysis Systems Technologies. (2019). Geoscience Community Analysis Toolkit: GeoCAT-comp [Software]. Boulder, CO: UCAR/NCAR - Computational and Informational System Lab. doi:10.5281/zenodo.6607205.

[2] Okuta, R., Unno, Y., Nishino, D., Hido, S., Crissman (2017). CuPy : A NumPy-Compatible Library for NVIDIA GPU Calculations

[3] Hoyer, S. Hamman, J., (2017). xarray: N-D labeled Arrays and Datasets in Python. Journal of Open Research Software. 5(1), p.10. DOI: https://doi.org/10.5334/jors.148.

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

[5] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2.

[6] Computational and Information Systems Laboratory. 2019. Cheyenne: HPE/SGI ICE XA System (Wyoming-NCAR Alliance). Boulder, CO: National Center for Atmospheric Research. doi:10.5065/D6RX99HX

[7] https://github.com/haniyeka/geocat-comp/tree/meteorology_and_crop_cupy