Exploring Performance of GeoCAT data analysis routines on GPUs

Haniye Kashgarani

University of Wyoming

National Center for Atmospheric Research

GeoCAT

Geoscience **C**ommunity **A**nalysis **T**oolkit (GeoCAT) is a toolkit used by the geoscience community to analyze and visualize data.

GeoCAT-comp program is one of the GeoCAT repositories, including previous NCL's non-WRF computational routines and other geoscientific analysis functions in Python.

GeoCAT-comp is built on **Pangeo** software ecosystem. The routines in GeoCAT-comp are either sequential or take advantage of **Dask for parallelization on the CPU.**

Data processing and data analysis is an embarrassingly parallel task and computationally intensive.

The project's focus is on porting GeoCAT-comp routines to GPUs.

GPU Programming in Python

CPU cores have a fast clock cycle but a limited number of cores. GPUs have hundreds of cores. By using GPUs in computations where a task can be divided into many subtasks, we can take advantage of **massive parallelization** and accelerate the code.

There are different CUDA-enabled packages in Python to help optimizing programs by GPUs, e.g., **Numba, Pycuda, and CuPy**. We investigated different approaches, and chose **CuPy**. CuPy is very similar to NumPy and it can be used as **a drop-in replacement** with NumPy.

With CuPy the programmer is not required to do memory management on both host and device or set and launch kernels manually.

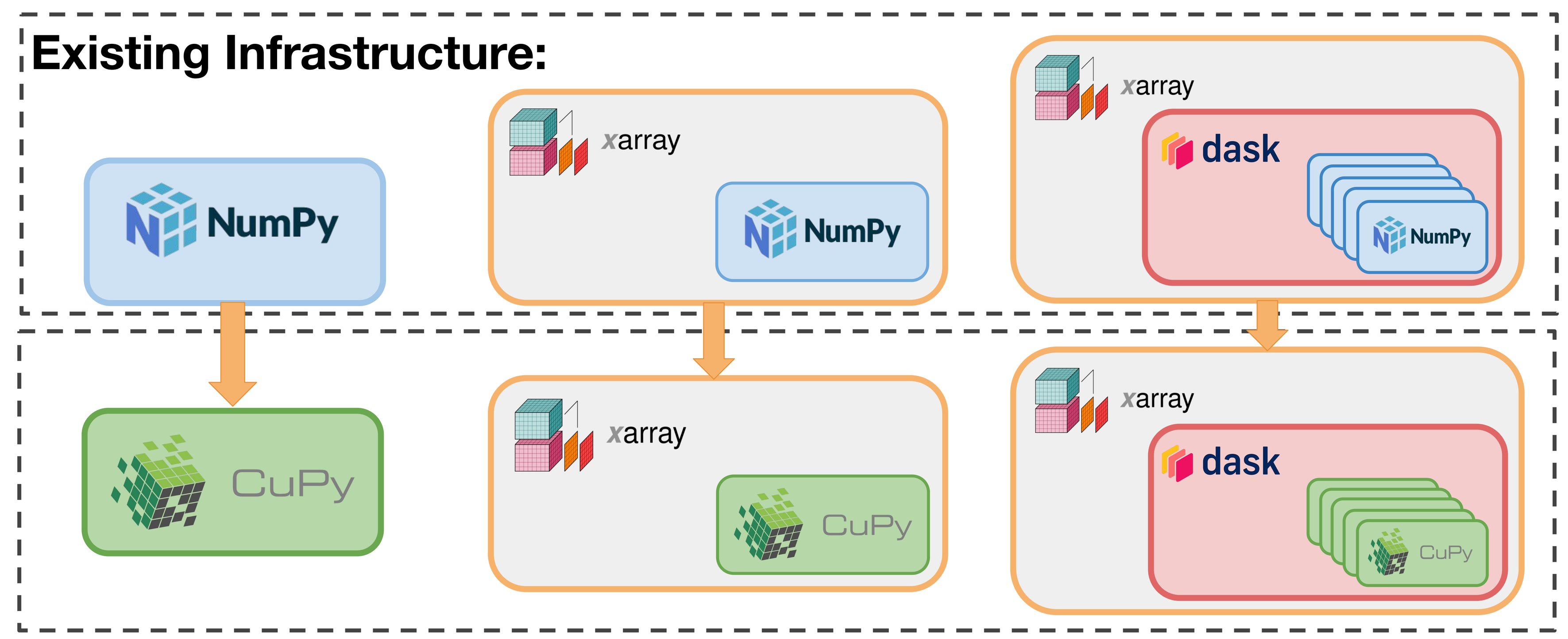
```
import cupy as cp
arr1 = cp.random.rand(10**2)
arr2 = cp.random.rand(10**2)
s = cp.add(arr1, arr2)
```

Xarray: Enables having labelled multi-dimensional arrays in Python.

Dask: Flexible open-source Python library for parallel computing.

Implementation

- The project's focus in on porting CPU parallelized routines, i.e., **meteorology.py and crop.py.**
- Arrays and multidimensional arrays in the GeoCAT routine are either **NumPy** or **Xarray.** Some routines used **Dask** for parallelizing Xarray arrays on the CPU.



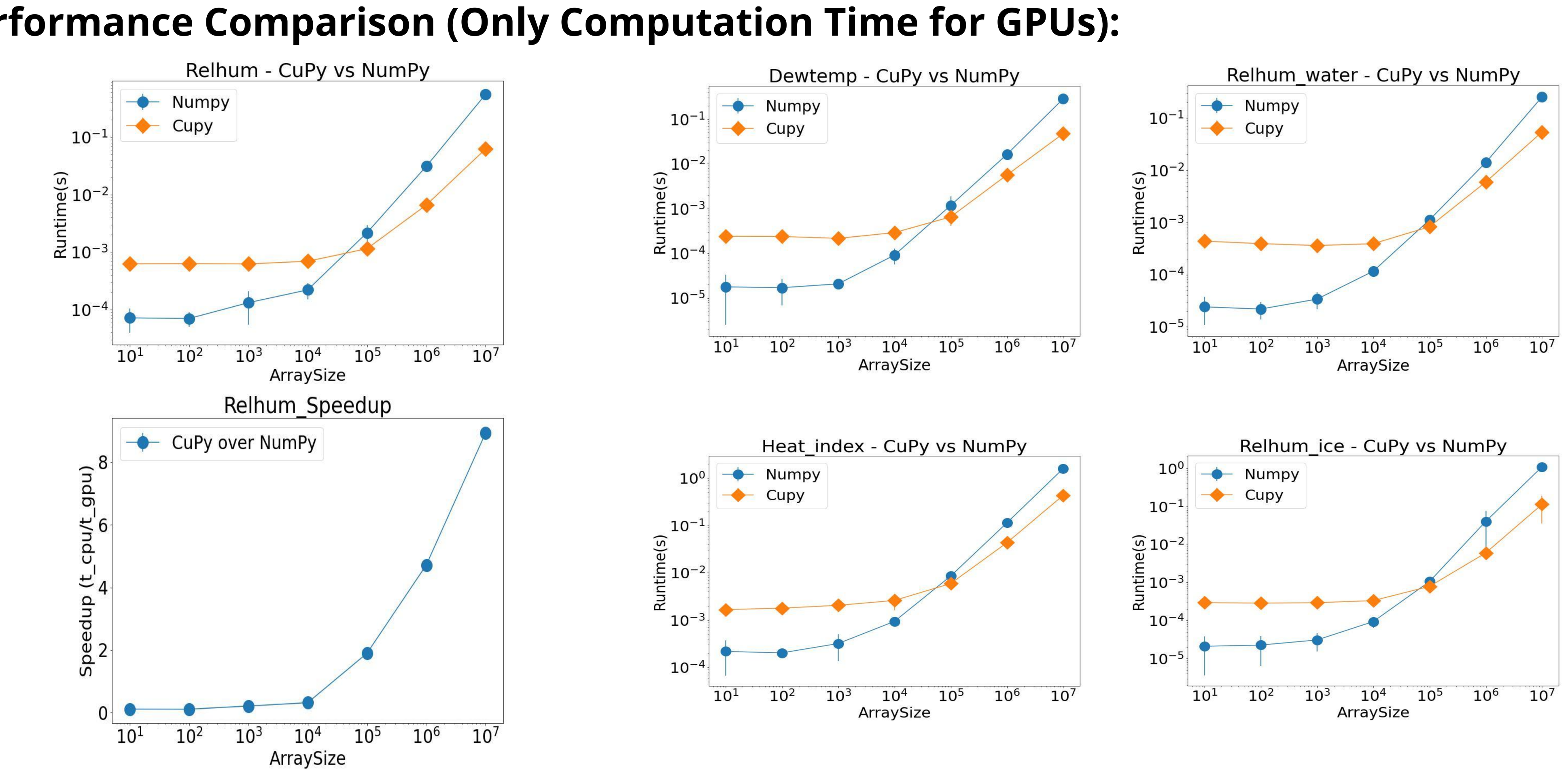

```
g_array = cupy.asarray(c_array)
```

```
g_array = xarray.DataArray(cupy.asarray(c_array.data))
```

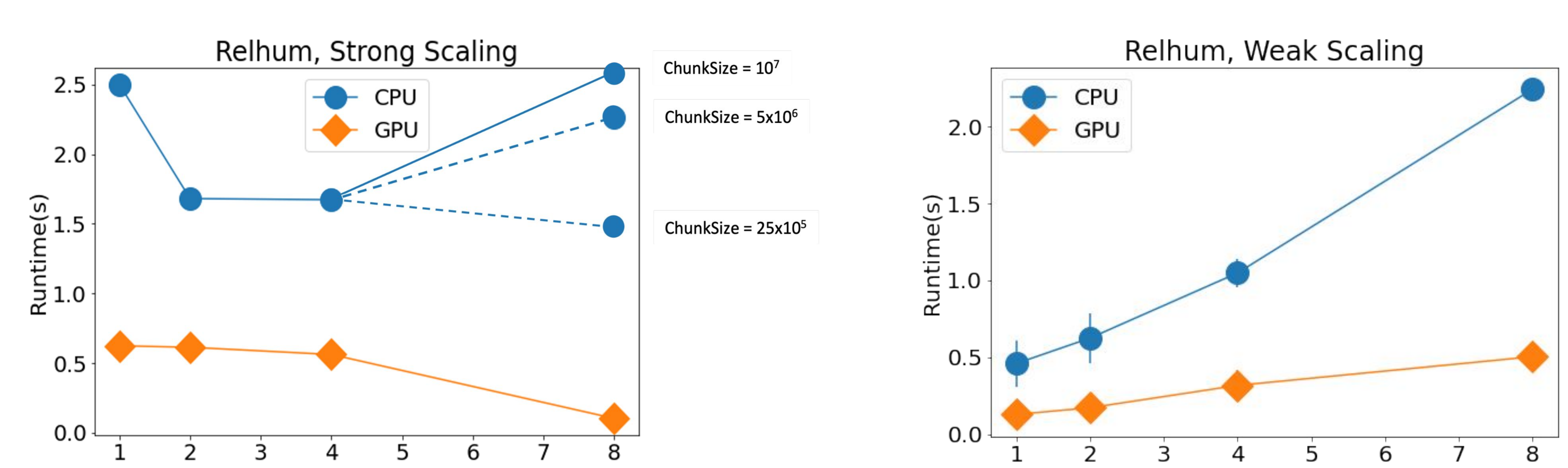
```
g_array = xarray.DataArray(c_array.data.map_blocks(cupy.asarray))
```

Performance Results

Performance Comparison (Only Computation Time for GPUs):



Scalability: Strong and Weak Scaling



Experimental Setup

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

Challenges

- Adapting Xarray and Dask with CuPy
- Inability to get performance improvements with some GPU tasks, e.g., Search functions: xarray.where().
- Numba JIT compiler auto-parallelizes NumPy arrays on CPU, but it is not adapted to CuPy arrays
- Correct way for benchmarking and gathering data:
  - Setting the correct chunksize

Conclusion and Future Work

- Explored ways to port GeoCAT-comp to run on GPUs, as recent supercomputers are shifting to include GPU accelerators as the major resource.
- Ported CPU parallelized routines to GPU.
- Validated output results with the precision of 10^{-7}.

Future Work:

→ Port all the routines.
→ Push to production.
→ Investigate writing kernel functions with CuPy's user-defined kernel capabilities or Numba and PyCuda.

Acknowledgement

Thanks to Cena Miller, Supreeth Suresh, and Anissa Zacharias for their mentorship.

Special thanks to ASAP, GeoCAT, CSG, and SIParCS Team!

Ported Branch: **GeoCAT Github:**

hkashgar@uwyo.edu