Exploring performance of GeoCAT data analysis routines on GPUs

Haniye Kashgarani
University of Wyoming
Mentors: Cena Miller, Supreeth Suresh, Anissa Zacharias

July 27, 2022

Outline:

- Motivation: CPU vs GPU
- Introduction:
  - GPU Programming
  - CuPy
  - CuPy vs NumPy
- Implementation:
  - Drop-in Replacement
  - Array Types Conversions
- Results
- Challenges
- Conclusion and Future Work

Motivation: CPU vs. GPU

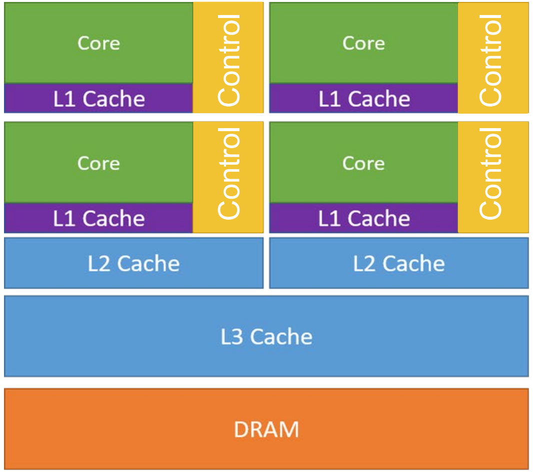
CPUs:

- Fast clock cycle
- Few cores

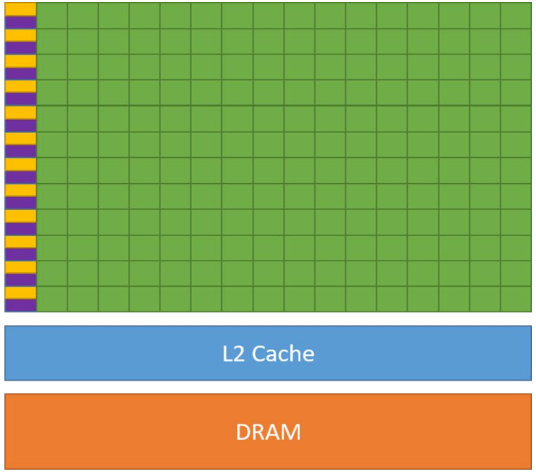
GPUs:

- Slower clock cycle
- Many cores
- Massive parallelism

Scientific Computation (GeoCAT), Deep Learning



CPU



GPU

GeoCAT

Geoscience community analysis toolkit:

- Pivot to Python - 2019
- GeoCAT-comp: previous NCL's non-WRF computational routines and other geoscientific analysis functions in **Python**.
- Built on **Pangeo** software ecosystem: **NumPy, Xarray, Dask**
- Sequential or Parallelized on the CPU using Dask.

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

The project's focus: Meteorology.py and Crop.py

https://geocat.ucar.edu/
https://github.com/NCAR/geocat-comp

GPU Programming

python

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

NVIDIA CUDA C/C++

```c
// Device code
__global__ void VecAdd(float* A, float* B, float* C, int N)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < N)
        C[i] = A[i] + B[i];
}
```

```c
// Host code
int main()
{
    int N = ...;
    size_t size = N * sizeof(float);

    // Allocate input vectors h_A and h_B in host memory
    float* h_A = (float*)malloc(size);
    float* h_B = (float*)malloc(size);
    float* h_C = (float*)malloc(size);

    // Initialize input vectors
    ...

    // Allocate vectors in device memory
    float* d_A;
    cudaMalloc(&d_A, size);
    float* d_B;
    cudaMalloc(&d_B, size);
    float* d_C;
    cudaMalloc(&d_C, size);

    // Copy vectors from host memory to device memory
    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

    // Invoke kernel
    int threadsPerBlock = 256;
    int blocksPerGrid =
            (N + threadsPerBlock - 1) / threadsPerBlock;
    VecAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);

    // Copy result from device memory to host memory
    // h_C contains the result in host memory
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);

    // Free device memory
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    // Free host memory
    ...
}
```

CuPy

GPU array backend in Python:

- drop-in replacement to run existing NumPy code on NVIDIA CUDA or AMD ROCm framework.

Installation of CuPy on Casper for NVIDIA devices:

- conda activate geocat
- module load cuda/11.6
- conda install -c conda-forge cupy cudatoolkit=11.6

https://docs.cupy.dev/en/stable/user_guide/index.html

6

NumPy vs. CuPy

Example of Supported Functions

NumPy	CuPy
`numpy.array`	`cupy.array`
`numpy.sum`	`cupy.sum`
`numpy.allclose`	`cupy.allclose`
`numpy.matmult`	`cupy.matmult`
`numpy.asarray`	`cupy.asarray`
`scipy.fft.fft`	`cupyx.scipy.fft.fft`
`scipy.linalg.convolution_matrix`	`cupyx.scipy.linalg.convolution_matrix`

Not provided for these functions and dtypes:

`numpy.block`
`numpy.complex256`
`numpy.delete`
`numpy.insert`
`numpy.cast`
`numpy.float128`
`numpy.ScalarType`

CuPy Drop -in Replacement

Using cupy as the simple drop -in replacement of numpy and compare the performance of CuPy and NumPy in terms of speedup

Pros:

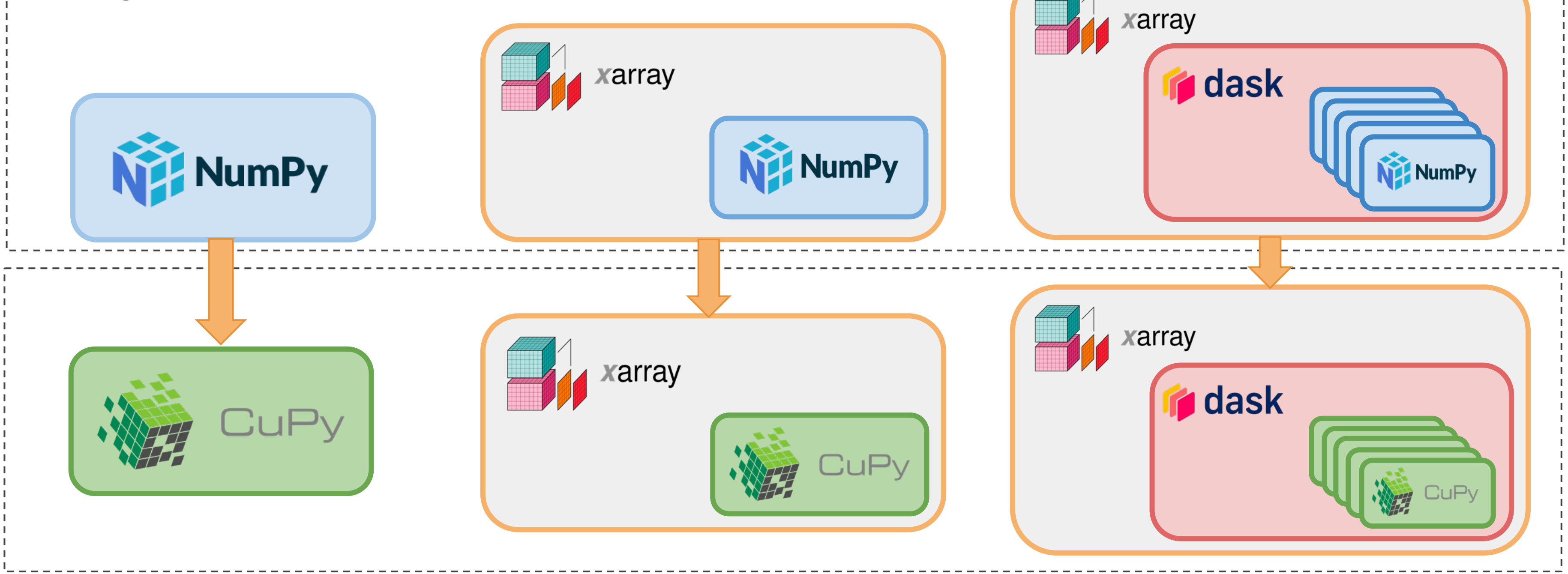
- **CUB variable:** os.environ['CUPY_ACCELERATORS'] ='cub'
    - Optimizes the GPU computation for reduction functions
- **CuTensor:** os.environ['CUPY_ACCELERATORS'] ='cutensor'
    - Optimizes the GPU computation for tensor operations

Cons:

- Speedup might be limited to the large array size
- May not speed up for some functions:
    - Search functions e.g., xarray.where()

Array Types Conversions

Existing Infrastructure:




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

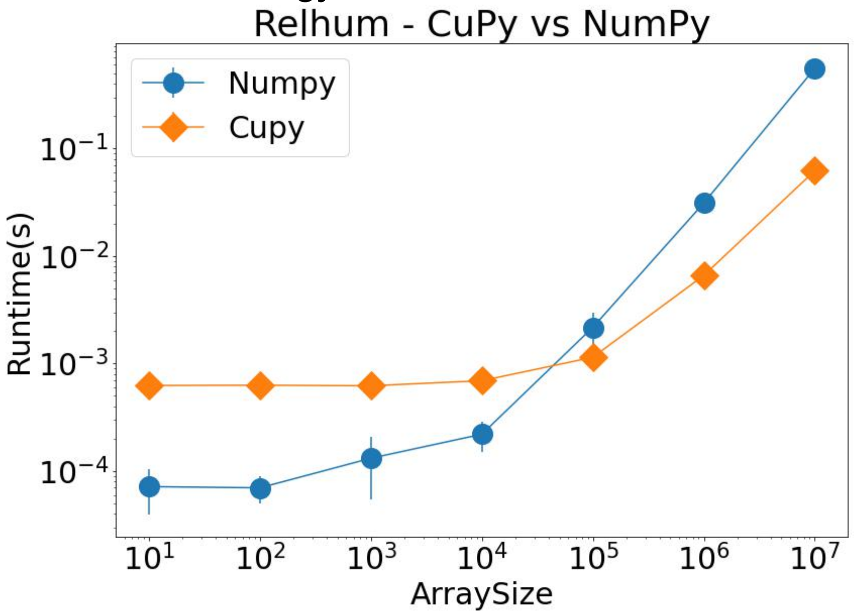
```
g_array = xarray.DataArray(
cupy.asarray(c_array.data))
```

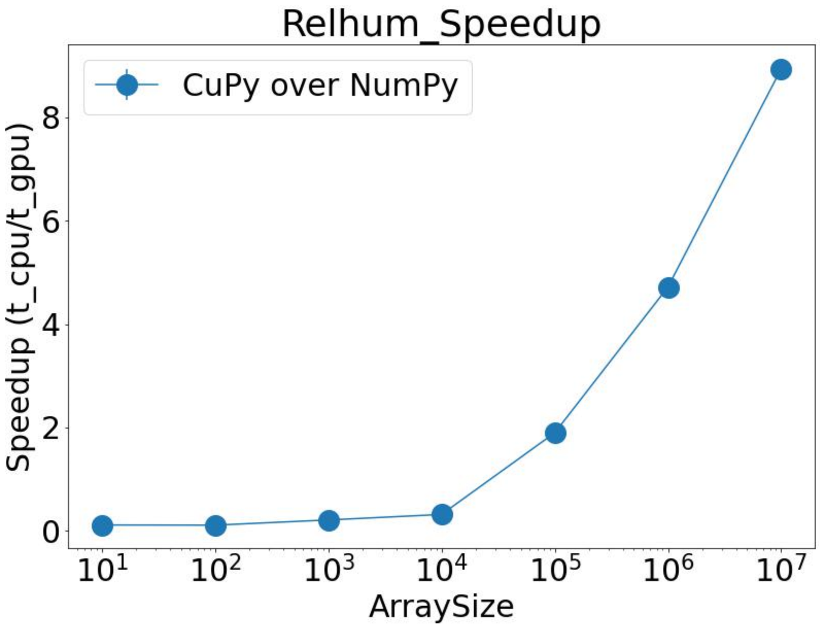
```
g_array = xarray.DataArray(
c_array.data.map_blocks(
cupy.asarray))
```

https://github.com/haniyeka/geocat-comp/blob/meteorology_and_crop_cupy/src/geocat/comp/comp_util.py

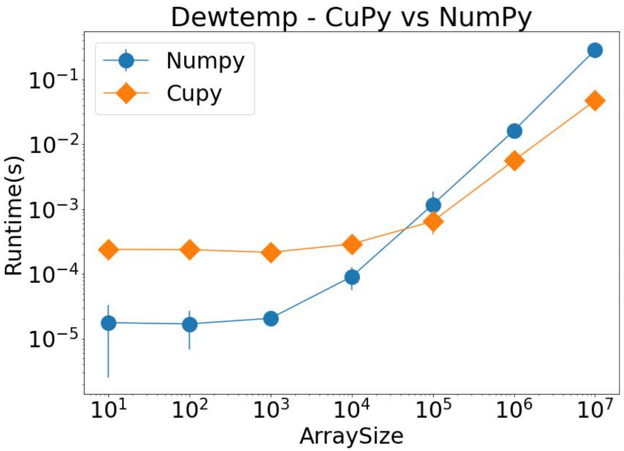
Results

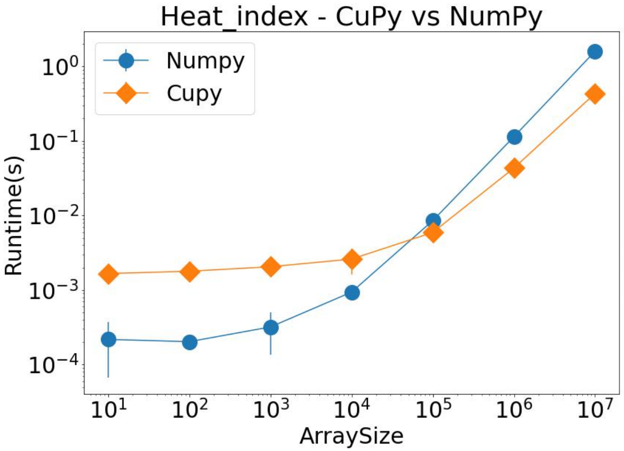
Performance for meteorology.relhum

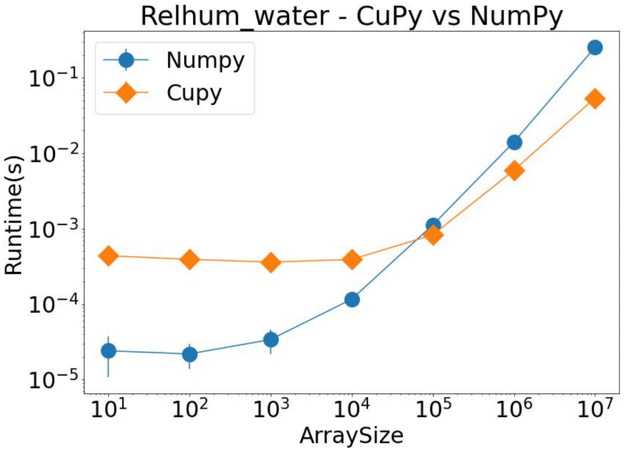


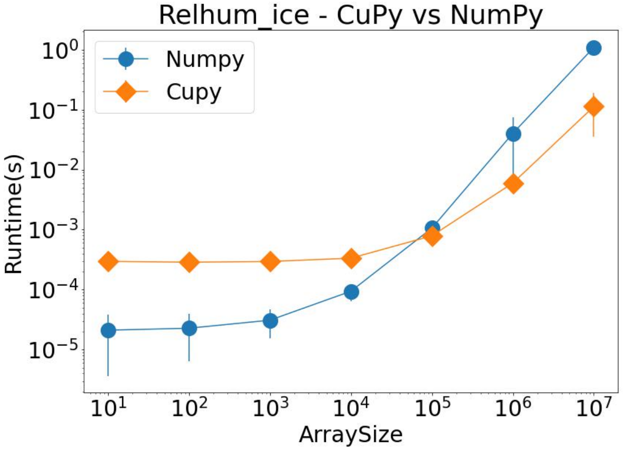


GPU node: 1 NVIDIA Tesla V100 32GB SXM2 GPUs with NVLink
1 CPU core from 2 18-core 2.3-GHz Intel Xeon Gold 6140 (Skylake) processors per node
CPU nodes: Dual-socket nodes, 18 cores per socket
2.3-GHz Intel Xeon E5-2697V4 (Broadwell) processors 16 flops per clock


Dewtemp - CuPy vs NumPy
Numpy
Cupy
Runtime(s)
ArraySize


Heat_index - CuPy vs NumPy
Numpy
Cupy
Runtime(s)
ArraySize

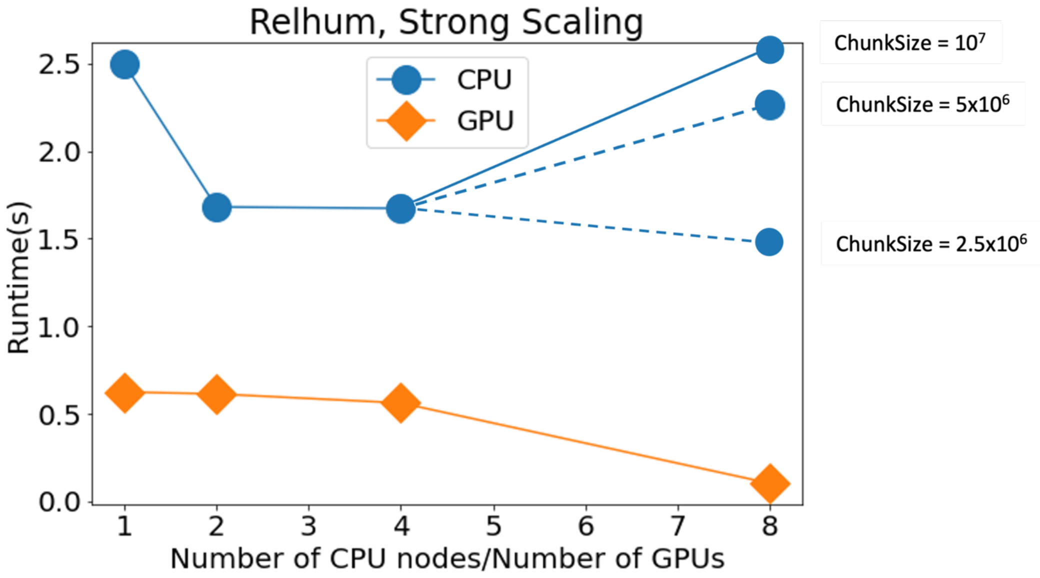

Relhum_water - CuPy vs NumPy
Numpy
Cupy
Runtime(s)
ArraySize


Relhum_ice - CuPy vs NumPy
Numpy
Cupy
Runtime(s)
ArraySize

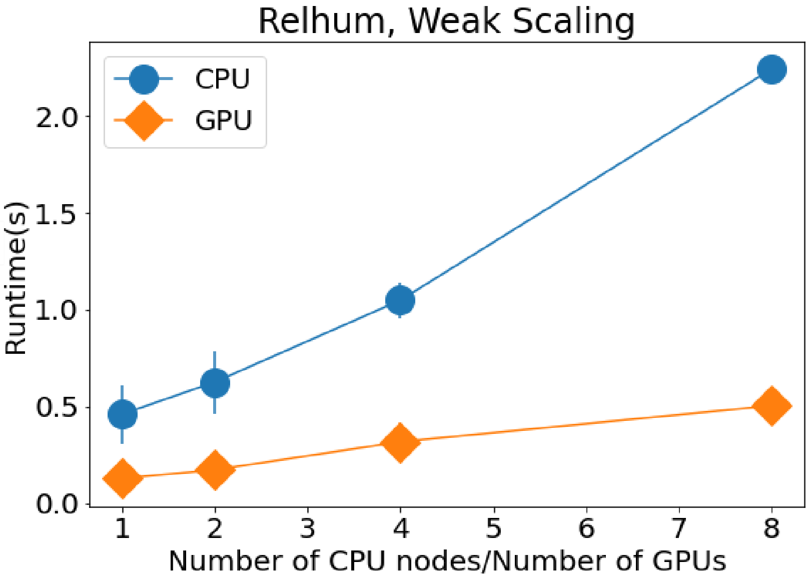
NumPy → CuPy

Scalability: Strong and Weak Scaling

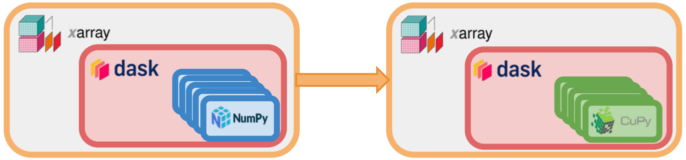
Array size: 10^{7}



Array size: 10^{6}



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



Challenges

- Learning Dask
- CuPy support for Numba JIT compiler
- Correct way for benchmarking and gathering data

Conclusion and Future Work

- ❖ Explored ways to port GeoCAT-comp to run on GPUs
- ❖ Provided a template to port other GeoCAT-comp routines to GPU
- ❖ Ported some serial and CPU parallelized GeoCAT-comp routines to GPU, and analyzed the performance
- ❖ Validated the results of NumPy and CuPy to a precision of 10^{-7}

The priority was on delivering an easy implementation to the GeoCAT team. Refactoring of the code is required if better performance is desired.

Future Work:

- ➢ Port other GeoCAT-comp routines
- ➢ Push the ported code to production
- ➢ Investigate writing kernel functions with Numba, and cuNumeric

Thank you!

Mentors: Cena Miller, Supreeth Suresh, Anissa Zacharias

ASAP Team!

GeoCAT Team!

Orhan Eroglu, Anissa Zacharias

CSG Team!

Brian Vanderwende

SIParCS Team!

Virginia Do, AJ Lauer, Jerry Cyccone, Francesgladys Pulido and other 2022 interns.

GeoCAT Github:

Ported Branch Github:

Questions?

hkashgar@uwyo.edu