# Analysis of *FastEddy*® Model Data on GPUs

***Shay Liu,***
*NCAR CISL SIParCS 2020 intern*

Mentors
**Supreeth Suresh** and **Cena Miller**

**July 29, 2020**

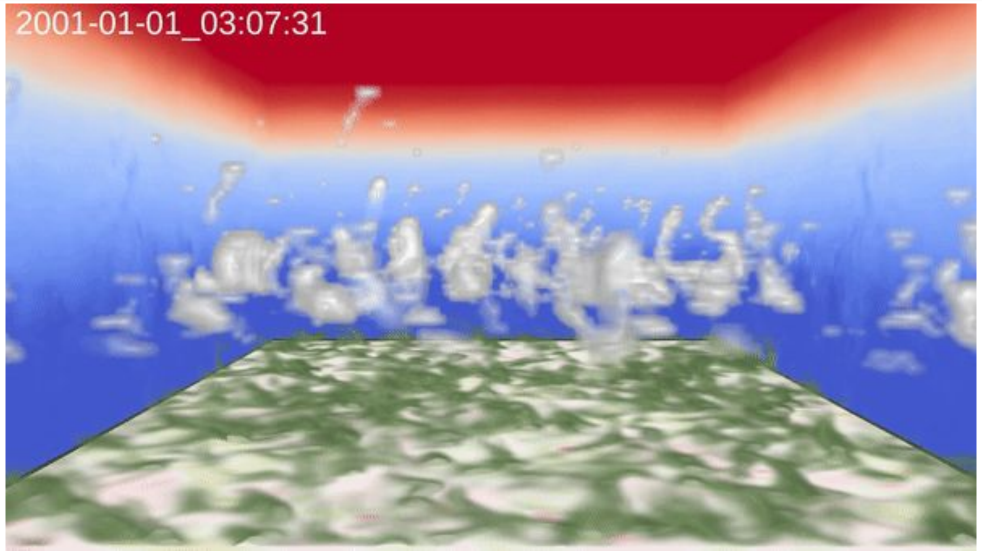
This material is based upon work supported by the National Center for Atmospheric Research, which is a major facility sponsored by the National Science Foundation under Cooperative Agreement No. 1852977.

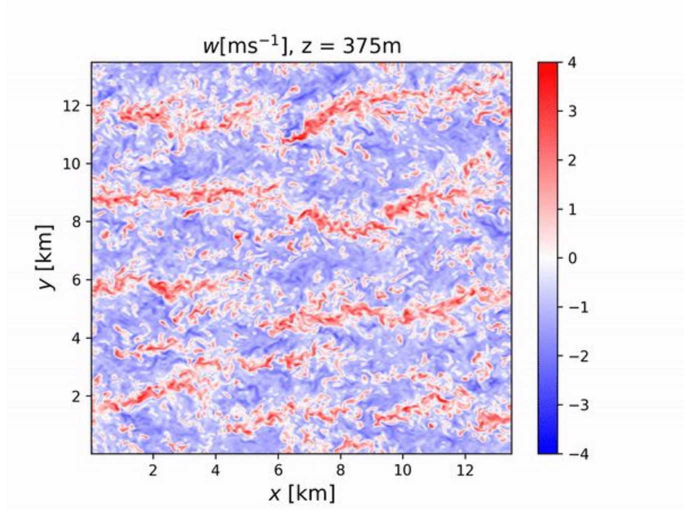
## Outline

- Introduction
- Project goals
- Timeline
- Single GPU data analysis
- Multi-node GPU data analysis
- Results
- Summary

# Introduction

- Model analysis is traditionally done on CPUs
- Model analysis is often embarrassingly parallel and compute intensive
- These types of tasks are well suited for GPU acceleration
- *FastEddy*® is a GPU-based large eddy simulation (LES) model developed in RAL, which produces large datasets
- Using GPUs for *FastEddy*® analysis potentially reduces I/O and helps create a faster process of analysis for the science team

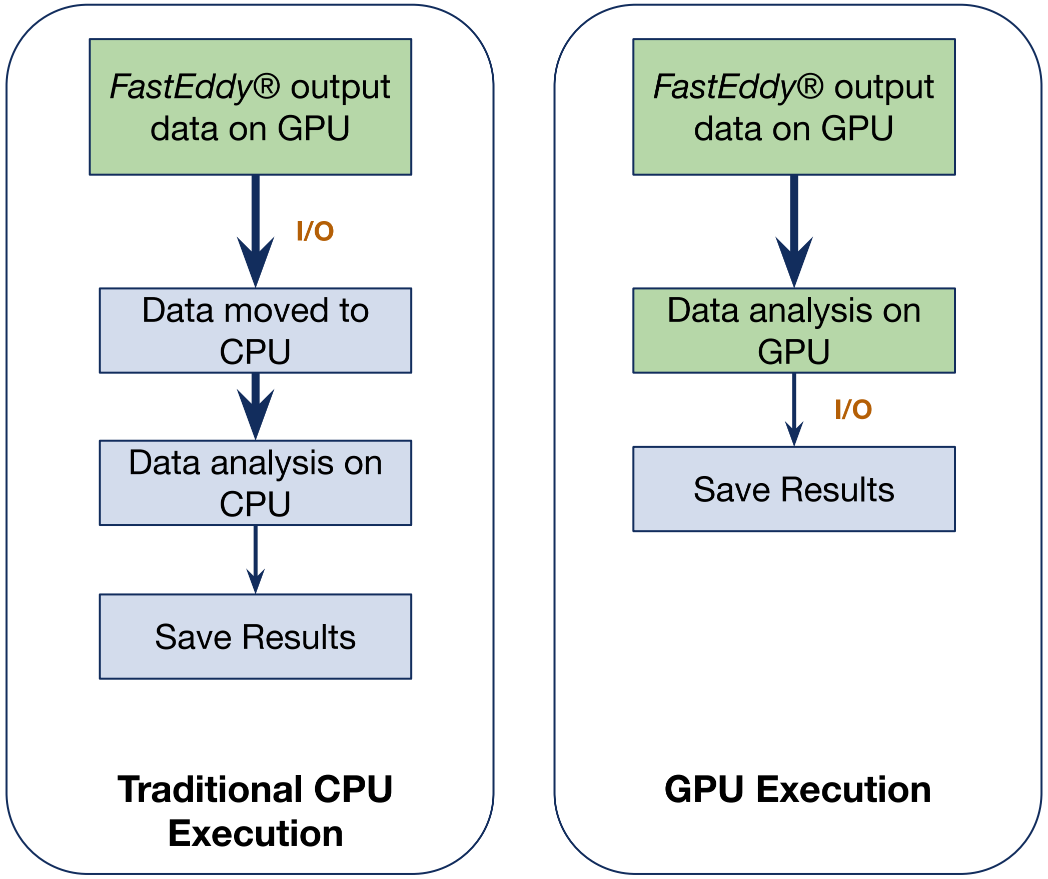




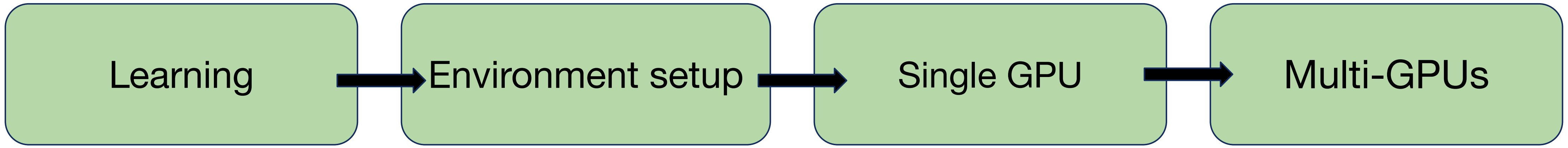
Video courtesy of Dr. Jeremy Sauer, *NCAR RAL*

## Project Goals

1. Become familiarized with the architecture of GPUs
2. Perform *FastEddy®* data analysis on GPUs
   a. Single GPU execution
   b. Multi-GPU execution
3. Prototype a simplified GPU acceleration of the data science phase
   a. Accelerate data analysis on GPUs to match the high-speed data production on GPUs



## Work Progress



## Initial Setup

### Learning

- NVIDIA Courses about RAPIDS
- Cupy, CuDF, CuGraph, and Dask

### Setting up environment

- Conda environment
- Package installation
- JupyterLab extensions

# Libraries

**Cupy**

- Cupy is a python library to do element-wise array operations on GPU
  - Analogous to numpy on CPU
- Cupy simplifies GPU acceleration process
- Cupy preserves data structures

**Dask**

- Dask schedules tasks for parallelism and distributes the workload for you
- Dask uses lazy evaluation and thus optimizes load and store of data
- Dask works with xarray, cupy, numpy, pandas, cudf, etc

**JupyterLab**

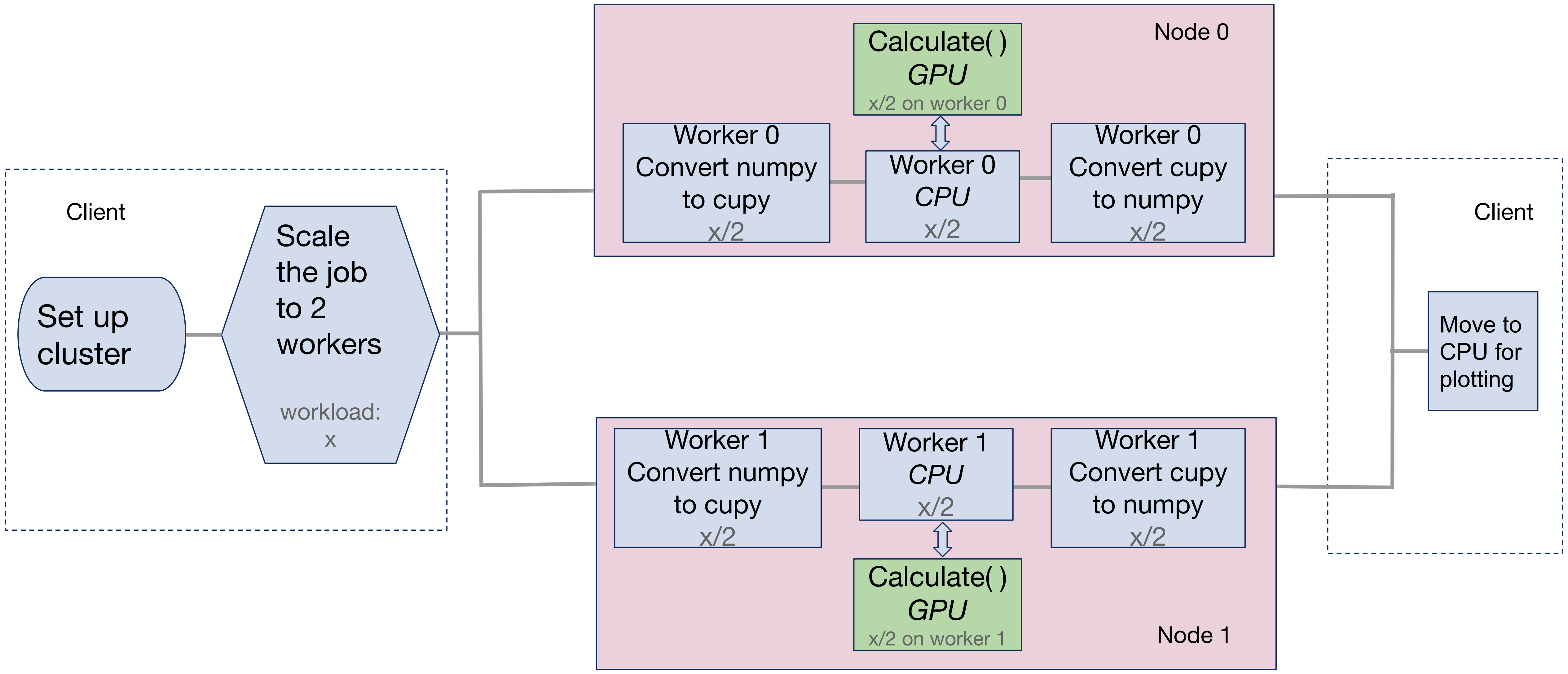
- JupyterLab is a web-based interactive development environment (IDE)
- JupyterLab supports dask lab extensions to monitor work processes

## System Details

**NCAR Casper Supercomputer**

- Up to **384 GB** DDR4-2666 memory per node
- **2 18-core** 2.3-GHz Intel Xeon Gold 6140 **(Skylake) processors** per node
- 2 TB local NVMe Solid State Disk
- 1 Mellanox ConnectX-4 100Gb Ethernet connection (GLADE, Campaign Storage, external connectivity)
- 1 Mellanox ConnectX-6 HDR100 InfiniBand link
- **1 NVIDIA Quadro GP100 GPU 16GB** PCIe on each of 8 nodes

# Dask Execution Workflow


Client
Set up cluster
Scale the job to 2 workers
workload: x
Node 0
Worker 0 Convert numpy to cupy x/2
Worker 0 CPU x/2
Calculate( ) GPU x/2 on worker 0
Worker 0 Convert cupy to numpy x/2
Node 1
Worker 1 Convert numpy to cupy x/2
Worker 1 CPU x/2
Calculate( ) GPU x/2 on worker 1
Worker 1 Convert cupy to numpy x/2
Client
Move to CPU for plotting

## Dask Working Example: Scheduler, Workers, & Delayed Objects

**Request 2 computing nodes with 1 GPU each (inside a Jupyter session)**

```
cluster = SLURMCluster(cores=1, processes=1, walltime='01:00:00',
                      scheduler_options={"dashboard_address" :'0.0.0.0'},
                      extra=['--resources GPU=1'],
                      job_extra=['--constraint=gpu','--account=ntdd0002',
                      '--reservation=TDD_2xgp100','--mem=0'],
                      env_extra=['module load cuda/10.1',])
client = Client(cluster)
cluster.scale(2)
```

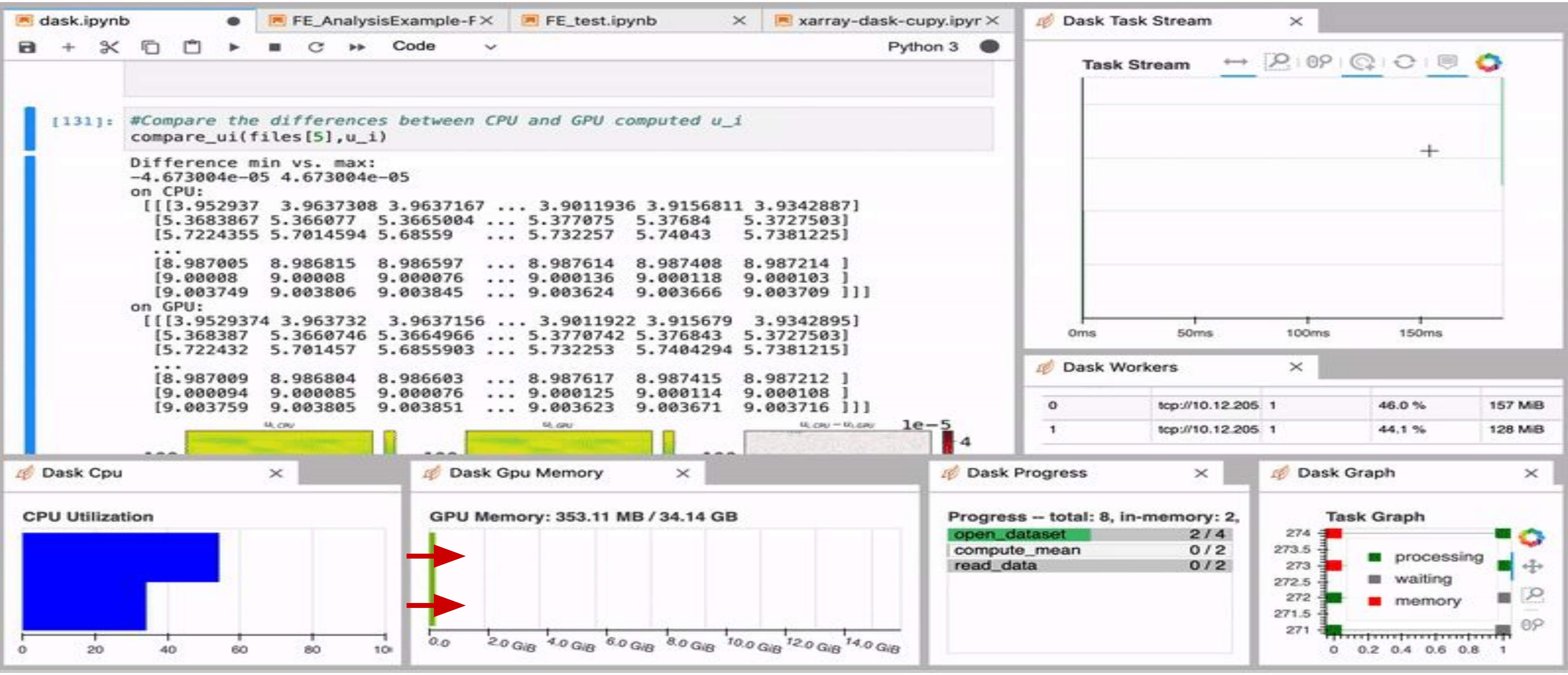
```
!squeue -u $USER -l
------------------------------------------------------------------------------------------
Mon Jul 20 17:53:42 2020
     JOBID PARTITION     NAME     USER    STATE       TIME TIME_LIMI  NODES
NODELIST(REASON)
   5614085       dav     dask-wor  xuecliu  RUNNING      0:02   1:00:00      1 casper06
   5614086       dav     dask-wor  xuecliu  RUNNING      0:02   1:00:00      1 casper07
   5613902       dav     srun      xuecliu  RUNNING     37:33   6:00:00      1 casper23
```

```
@dask.delayed
def my_func(filepath):
    x = cupy.array(y)
    return(x)
```

```
results = my_func(filepath)
x = results.compute()
```

# Monitor Work with Dask Graphic Extensions

Dask-labextension + nvdashboard in a JupyterLab session



```
[131]: #Compare the differences between CPU and GPU computed u_i
       compare_ui(files[5],u_i)

Difference min vs. max:
-4.673004e-05 4.673004e-05
on CPU:
 [[[3.952937  3.9637308 3.9637167 ... 3.9011936 3.9156811 3.9342887]
  [5.3683867 5.366077  5.3665004 ... 5.377075  5.37684   5.3727503]
  [5.7224355 5.7014594 5.68559   ... 5.732257  5.74043   5.7381225]
  ...
  [8.987005  8.986815  8.986597  ... 8.987614  8.987408  8.987214 ]
  [9.00008   9.00008   9.000076  ... 9.000136  9.000118  9.000103 ]
  [9.003749  9.003806  9.003845  ... 9.003624  9.003666  9.003709 ]]]
on GPU:
 [[[3.9529374 3.963732  3.9637156 ... 3.9011922 3.915679  3.9342895]
  [5.368387  5.3660746 5.3664966 ... 5.3770742 5.376843  5.3727503]
  [5.722432  5.701457  5.6855903 ... 5.732253  5.7404294 5.7381215]
  ...
  [8.987009  8.986804  8.986603  ... 8.987617  8.987415  8.987212 ]
  [9.000094  9.000085  9.000076  ... 9.000125  9.000114  9.000108 ]
  [9.003759  9.003805  9.003851  ... 9.003623  9.003671  9.003716 ]]]
```

| | | | | |
|---|---|---|---|---|
| 0 | tcp://10.12.205 | 1 | 46.0 % | 157 MiB |
| 1 | tcp://10.12.205 | 1 | 44.1 % | 128 MiB |

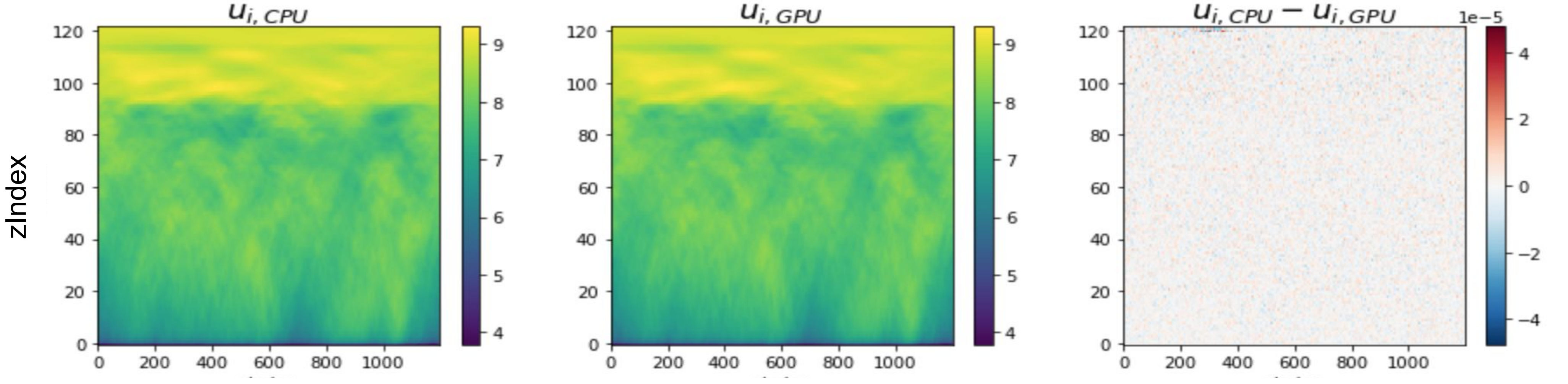
Progress -- total: 8, in-memory: 2,

| | |
|---|---|
| open_dataset | 2 / 4 |
| compute_mean | 0 / 2 |
| read_data | 0 / 2 |

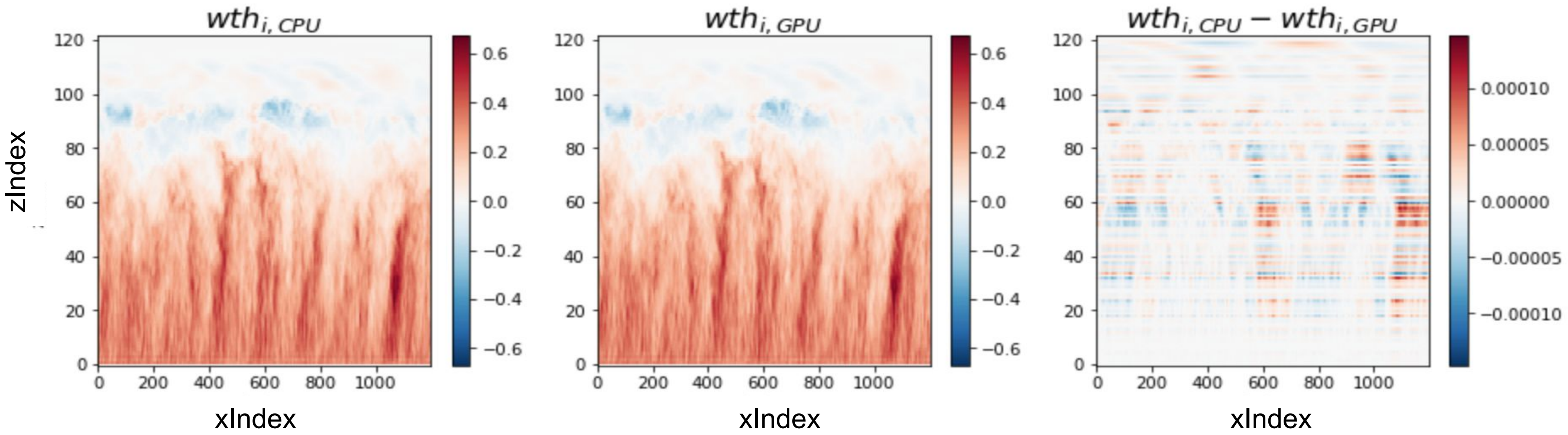
# Validation

**Reference (CPU)** **Test on GPU** **Difference**

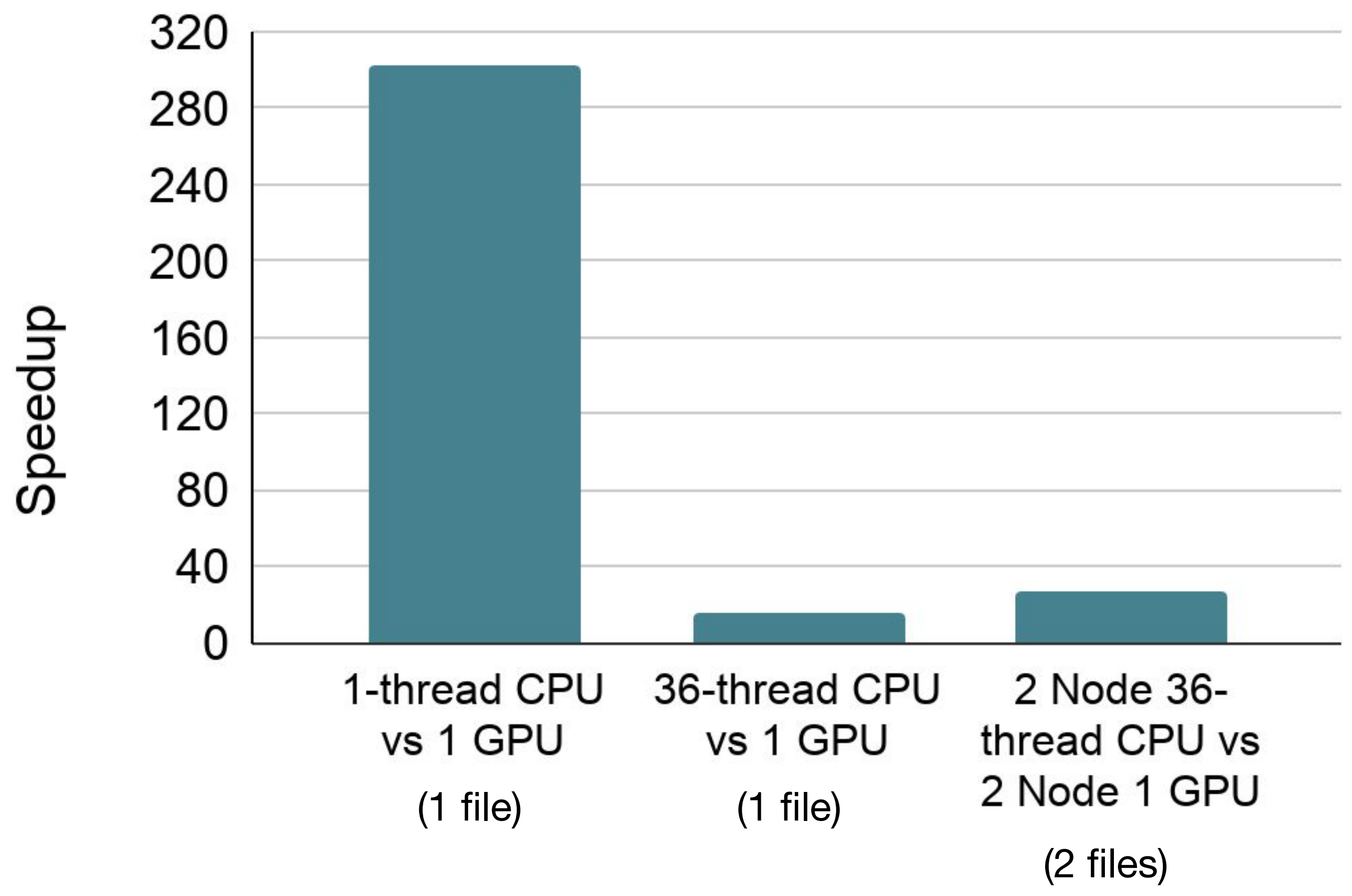
Horizontal perturbation wind speed



Turbulent heat flux



# Results



- Speedup for analysis of a single file:
  - 302x for 1-thread CPU vs. 1 GPU
  - 15.3x for 36-thread CPU vs. 1 GPU
- Speedup for analysis of two files:
  - 26.3x for 2 nodes, 36-thread CPU vs. 2 nodes with 1 GPU on each node

## Summary and Future Work

### Summary

- Cupy significantly improves and simplifies the process of GPU acceleration for data analysis
- Dask + cupy together facilitate data analysis on multi-GPUs

### Future work

- Incorporate an in-situ GPU acceleration workflow in *FastEddy®*

# Acknowledgements

## Technical & Scientific Support

- Anderson Banihirwe, *NCAR CISL*
- Mick Coady, *NCAR CISL*
- Dr. Raghu Raj Kumar, *NVIDIA*
- Dr. Richard Loft, *NCAR CISL*
- Dr. Jeremy Sauer, *NCAR RAL*

## Administrative Support

- AJ Lauer, *NCAR CODE*
- Virginia Do, *NCAR CODE*
- Jerry Cyccone, *NCAR Education & Outreach*
- Jess Hoopengardner, *NCAR CODE*

**Thank you.**

**Questions?**