ICEBERG

Imagery Cyberinfrastructure and Extensible Building Blocks to Enhance Research in the Geosciences

Submitted by:
Jake Lewandowski
Aashrit Kondapalli
Alexander Dewey
Raj Patel

Team Project Number: 49

Advisor[s]:
Shantenu Jha
Matteo Turilli

Github Repo: https://github.com/lewandowskijake1123/Capstone2018

May 1, 2018

Submitted in partial fulfillment of the requirements for senior design project

Electrical and Computer Engineering Department
Rutgers University, Piscataway, NJ 08854

Table of Contents

.

Abstract

Currently, there exists petabytes of high resolution aerial view images of Antarctica, the Arctic, and Greenland available for study to the research institutions of the U.S.. The members of Project ICEBERG (scientists from various universities around the U.S.) have begun using this resource for various applications in the fields of biology, climatology, and geology by developing software to automate the processing of these images. In order to fully take advantage of the vast amount of data that has been provided, we began to develop high-performance computing pipelines for some of these applications.

Half of our group focused on a Pytorch machine learning algorithm used to scan these photos for seal populations. By using the PSC-Bridges HPC of the Pittsburgh Supercomputing Center, we were able to efficiently expand the rate of the algorithm's testing and validation capabilities, as well as provide a larger platform for neural network training. Our code was written in Python, utilizing Rutgers RADICAL's own RADICAL-Cybertools, libraries designed to aid in the structure and design of large, scalable applications aimed to be used in HPC and distributed computing systems.

The other half focused on an algorithm that delineates glacial streams from other water features in images of the Greenland Ice Sheet. The RADICAL Cybertools Ensemble Toolkit was integrated to model the code as a Pipeline, with Stages and Tasks to represent the delineation

process, and this code was run on the PSC Bridges supercomputer to provide a more efficient platform for the delineation process.

Contributions

Name	Contribution
Jake Lewandowski	File transfer to Bridges, training neural network model on Bridges, executing training on Bridges, Video Creation, Poster Creation, Presentation Creation, slight work with Ensemble Toolkit, wrote pipeline code
Aashrit Kondapalli	File transfer to Bridges, writing batch scripts, running jobs on Bridges, integration of Ensemble Toolkit, creating Use Case document, Poster/Presentation creation, wrote pipeline code
Alexander Dewey	File transfer to Bridges, Neural network model training on Bridges, Altering biological life detection code for usability on Bridges, Writing batch scripts and python scripts for testing, Poster/Presentation Creation, Interactive blob detection on Bridges
Raj Patel	File transfer to Bridges, Use Case Document, Poster/Presentation Creation, writing batch scripts, integration of EnTK

Overview

High-performance computing is becoming an important performance-enhancement tool within the scientific/computing communities, specifically within the geo-imaging and biological sciences field. Just as we are searching for seals/delineating streams in images, other scientists are using code to look for unique elements in their own satellite images. Due to the large amounts of image data and processing power required in training neural networks, it is difficult for scientists to run these programs on their own machines. Additionally, it may be difficult for scientists not experienced in computing to understand how to run the code on their machines. By providing a pipeline on a HPC to automate the image processing programs, the ICEBERG project gives scientists an efficient and scalable solution to process their large sets of images. We worked to create two different pipelines, one for the Seals Use Case and one for the Stream Delineation Use Case, using PSC Bridges and the RADICAL-Cybertools Ensemble Toolkit (EnTK), which will be detailed in the further sections.

Bridges

The Bridges Supercomputer contains Intel Haswell CPU's and NVIDIA Tesla GPU's for users to compute on. Our allocations on the computer came from the XSEDE research group, which also provided us the login hub in which to logon to Bridges with.

Method:

A large part of this project was contributed to learning how to connect to Bridges, setting up the environment and configuring for the Ensemble Toolkit to access via ssh.

To connect to Bridges through XSEDE, the user must download the DUO 2-Factor Authentication Mobile App and ssh into the XSEDE portal with the following command:

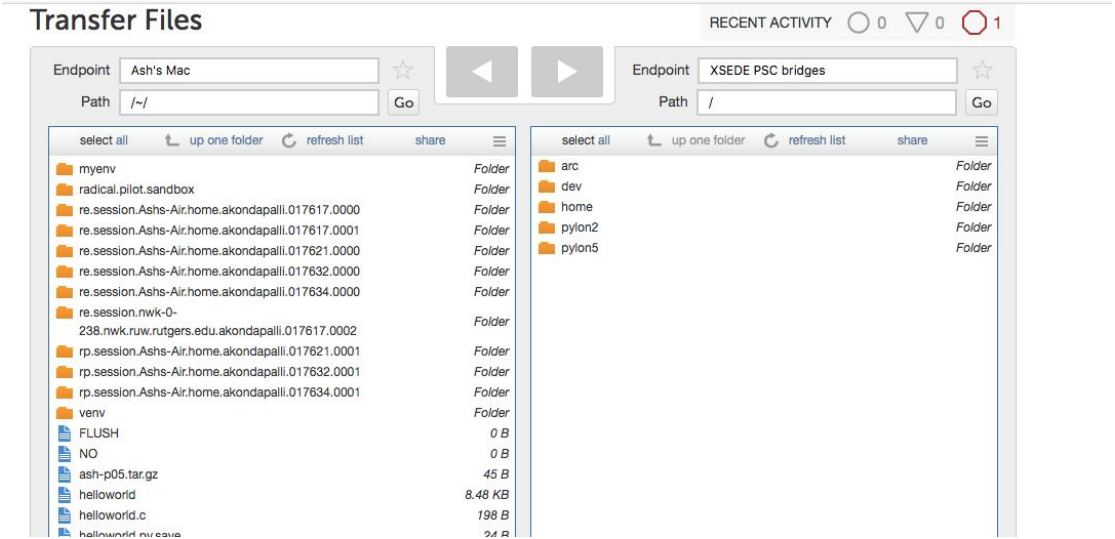
ssh -l <username> login.xsede.org

The user will be prompted for their XSEDE login credentials, after which they will be sent a DUO push notification to approve the login. Next, the user would gsissh into the Bridges system, using the following command:

gsissh bridges

The user will now be connected to the Bridges server and can start executing their desired commands. The system has multiple compilers and modules that can be loaded for use.

Transferring files to the Bridges file systems was done through the use of the Globus toolkit, a third-party software used for easy file transfer. The following screenshot shows the Dashboard used to transfer files from a personal computer to the Bridges system:



Files can easily be transferred from one endpoint to another by selecting the file and using the arrow keys to transfer the file from one endpoint to another.

Ensemble Toolkit

The Ensemble Toolkit is a Python library developed by Rutgers own RADICAL group used to develop and execute large-scale, ensemble-based workflows on the RADICAL-Pilot runtime system. It allows the user to worry only about "what" needs to be executed and "when" -- the toolkit takes care of the "where" and "how." EnTK code is modeled using the Pipeline, Stage, Task (PST) model, explained in the following figure:

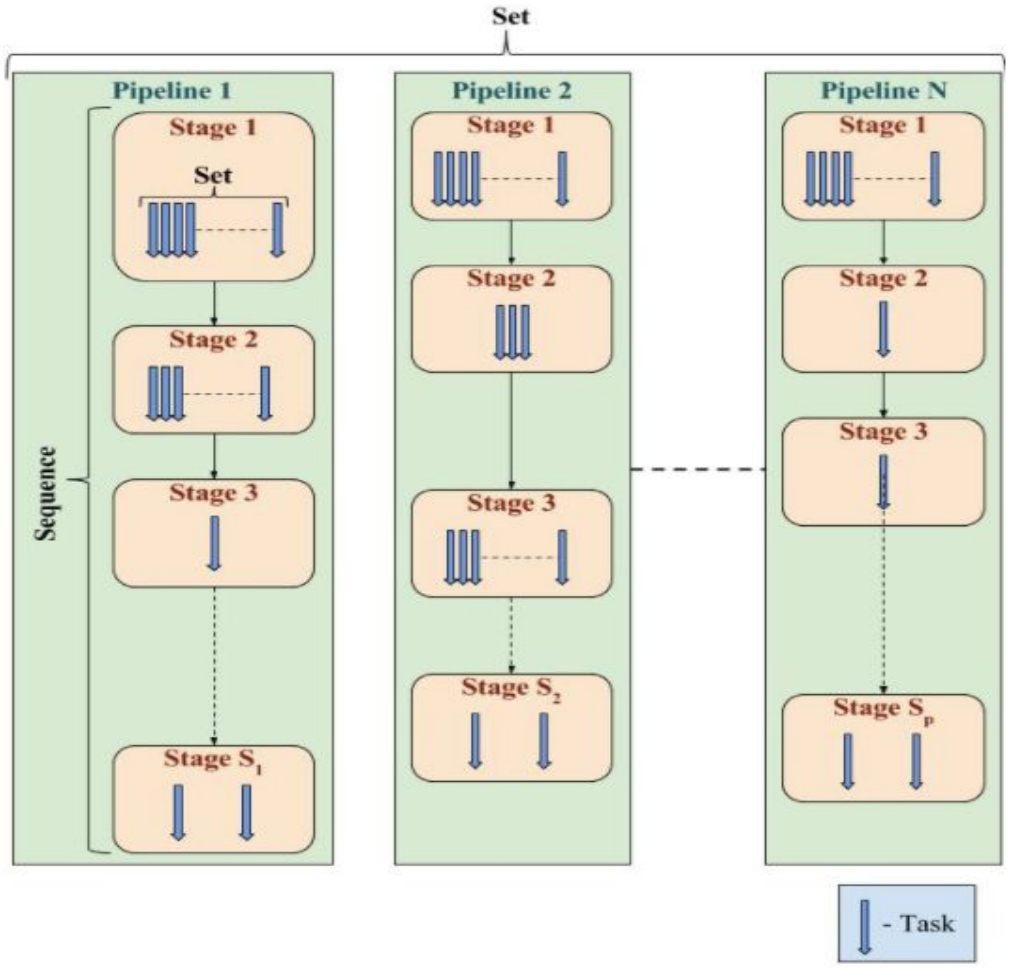


Figure 1: PST Model

Tasks are abstractions of computational tasks that contain information regarding an executable, it's software requirements, and any data dependencies. A stage is a set of tasks that can be executed concurrently, and a pipeline is a list of stages, where each stage can only be executed after the previous stage has completed. This allows users to account for dependencies in their code. To install EnTK, users must first create a virtual environment and then use pip to install:

```
virtualenv $HOME/myenv
source $HOME/myenv/bin/activate

pip install radical.entk
```

Next, users must install Rabbit-MQ Server and MongoDB to allow for message transfers and set up the MongoDB server used for the RADICAL Pilot system. This can be done by following the simple instructions on the two programs' respective web pages. Additionally, EnTK allows users to run their code on a HPC from their own machine without having to login to the remote server, through the use of passwordless gsissh. This was set-up by installing the Globus toolkit password and then creating an RSA key-pair on our personal machines through the following command:

```
ssh-keygen -t rsa -b 4096
```

The generated key is then submitted to Bridges through their key management system and will be approved for use shortly by the administrators. After setting up access, users can change their resource request specifically for the target machine. For example:

```
res_dict = {

    'resource': 'xsede.stampede',
```

```
    'walltime': 10,
    'cores': 1,
    'project': 'TG-MCB090174',
    'queue': 'development',
    'schema': 'gsissh'

}
```

In the above code, the user creates a dictionary to describe their resource request on the Stampede computer, the walltime/number of cores required, and project charge number, along with the queue on the HPC and the login schema.

Use Cases

Biological Feature Detection:

Description:

This use case will entail of processing a set of images from the WorldView3 satellite and sorting these images into folders with the corresponding type of seal labeled. A pipeline will be created to achieve this goal. This pipeline will transfer the set of images to be predicted to a remote supercomputer, train a neural network model (using the Python package Pytorch) on a set of test images, predict how many and what kind of seals appear in each image, run a “blob detector” which displays the set of images with the detected seals highlighted, and return the images in folders with corresponding labels. The Ensemble Toolkit(entk) will be integrated to simplify the file transfer, job submission and file return.

Requirements:

- -Authentication to HPC resource (bridges)
- -RSA key verified by Pittsburgh Supercomputing Center to allow passwordless ssh to Bridges (requirement for entk)
- -Test set of images to train neural network
- Modules:
  - -python3
  - -cuda
  - -pytorch
  - -radical.entk
  - -tensorboardX

Actors:

User (initiating), Ensemble Toolkit(participating), Bridges Supercomputer (participating), Neural network model (participating)

Preconditions:
-High resolution satellite image which may or may not contain Arctic biological life.

Postconditions:
-Updated image with identified biological life.
-Updated neural network.

Flow of events for main success scenario:
1. User executes code to submit pipeline to HPC resource
2. Job sits in queue while resources are allocated
3. Pipeline is executed on Bridges
4. User is returned folders with images classified

Stream Delineation

Descriptions
- **Collaborators:** Prof. Vena Chu, Ash Kondapalli, Raj Patel
- **Field:** Glaciology
- **Requirements**
  - Access to PSC Bridges Supercomputer
  - RSA key authentication from Bridges Key Management System to set up passwordless ssh for EnsembleToolkit
  - Orthorectified WV-2 image of glacial sheet
- **Measure of Success:** Image with glacial streams delineated from other glacial features

Platforms Used:
- **Languages/Library:** Python, C#, RADICAL EnsembleToolkit (EnTK), Globus Toolkit, OSGeo, ArcPy
- **HPC System:** PSC Bridges CPU (RM-shared partition)

Primary Challenges: Provide a Pipeline on Bridges to run the stream delineation algorithms to process large amounts of images using EnTK's PST Model

Stages:
- **Stage 1:** Creating Directories/File Transferring
- **Stage 2:** NDWI_ice Image Conversion, Multi-threshold Water Extraction, Multi-points Fast Marching Method, Slush Elimination
- **Stage 3:** Morphological Closing/Thinning

Step Breakdowns

Stage 1

First, a directory must be created named “StreamExtraction”, and within that folder another named “multiinput” and one named “multioutput”. The orthorectified imagery is then copied into ~/StreamExtraction/multiinput. Five more folders must be created within the “multioutput” folder, named “ndwi”, “thinned_img”, “thinned_shapefile”, and “watermask,” and “output_images”. The entire directory ~/StreamExtraction will then be transferred to the $SCRATCH, or pylon5, directory of Bridges via the Globus Toolkit.

Stage 2

A normalized difference water index (NDWI) is first used to extract the wet features of the WV-2 image. Through spectral discrimination, the image is converted to one in which the water features are extracted. Further water extraction is done through the use of different thresholds that select random pixels from streams and average their NDWI_ice values. The thresholds are determined based on the size of the streams the pixels are sampled from – t_low data comes from the smallest streams, t_mod larger streams, and t_high from lakes only. T_mod was selected as the most effective method as it identified the most amount of streams while eliminating most slush. A multi-points fast marching method is then used to reconnect streams that have been separated. However, small patches of slush will still remain, which will require an edge detection method used to eliminate the unnecessary slush regions left over. The StreamExtraction program was first compiled from its C# form into an .exe file to be able to run.

Input Files: StreamExtraction.exe
Output: Updated images after multi-threshold water extraction and slush elimination
Executable files: StreamExtraction.bat
Est. time for completion: ~1 Hour
Storage Space: ~700MB image
Cores/Nodes: 2 Cores, 1 Node
Walltime: 2 Hours requested

Stage 3

A morphological closing operation is used to fill in missing water pixels or remove isolated ones, since some streams become disconnecting due to pixels being removed during the water extraction process.

Input: Thin_multi.py
Output: Updated, pixel-reduced image after morphological closing and thinning is completed
Executable files: Thin_multi.bat
Est. time for completion: ~1 Hour
Storage Space: ~700MB Image
Cores/Nodes: 2 Cores, 1 Node
Walltime: 2 Hours Requested

Setbacks

The group encountered a number of setbacks during the project's timeline. The first setback was encountered when setting up the correct environment on the Bridges machine. The modules required by the user were not available or just out of date on Bridges. Namely, the Pytorch model was out of date, the people at the Pittsburgh Supercomputing Center were contacted in order to fix this. Another module used, tensorboardX, was nonexistent on the machine, due to its compatibility with python3. After much troubleshooting, the module was added by installing the correct module off of the tensorboardX github page using pip. These two setbacks occurred during the training stage of the pipeline. Another major setback, specific to the Stream Delineation use case, was the lack of available code for a majority of the project. We did not receive the necessary code to be run on the Bridges computer from one of the ICEBERG scientists up until three weeks before the project was due, so we had a hard time analyzing and integrating the code. Instead, we created proxy scripts to match what the actual scripts would look like with the Stream Delineation code, running simple executables in place of the necessary files. Additionally, the ArcPy, OSGeo, and MonoDevelop modules necessary for Stream Delineation were not supported on the Bridges machine as well, with ArcPy specifically requiring a license to run.

A huge setback was struck when integrating the Ensemble Toolkit(entk). The first error occurs when setting the username in the dictionary. In the API outline for the toolkit, username is an option when setting up the dictionary, part of the Resource manager. However, this contradicts the actual module where the username is not an attribute. A ticket has been opened and the issue will hopefully be resolved soon. The second error occurs in the "WFProcessor"

module of the Ensemble Toolkit. This module is missing the “wfp_terminate” module, which is needed to terminate the given process. Once these modules (“resource_manager.py” and “wfprocessor.py”) are both updated, the “radical.entk” package will need to be updated on the host machine and the code will likely be able to run.

Code

All code can be found on the following github:

https://github.com/lewandowskijake1123/Capstone2018

Seals Code

Below is the code for executing the pipeline on Bridges:

```
#Code to run pipeline on Bridges
#Author: Jake Lewandowski
from radical.entk import Pipeline, Stage, Task, AppManager, ResourceManager
import os

# ------------------------------------------------------------------------------

if os.environ.get('RADICAL_ENTK_VERBOSE') == None:
        os.environ['RADICAL_ENTK_VERBOSE'] = 'INFO'


def generate_pipeline(name, stages):  #generate the pipeline of prediction and blob detection

        # Create a Pipeline object
        p = Pipeline()
        p.name = name


        for s_cnt in range(stages):

        # Create a Stage object
        s = Stage()
        s.name = 'Stage %s'%s_cnt
    if(stage==1)
                # Create Task 1, training
                t = Task()
                t.name = 'my-task1'
```

```
                t.executable = ['sbatch']   # Assign executable to the task
                # Assign arguments for the task executable
                t.arguments = ['/Code/trainbatch.bat']
    else
                # Create Task 2,
                t = Task()
                t.name = 'my-task2'
                t.executable = ['sbatch']   # Assign executable to the task
                # Assign arguments for the task executable
                t.arguments = ['/Code/predscript.bat']
                t.download_output_data = ['classified_images'] #Download resuting images


    s.add_tasks(t)

        # Add Stage to the Pipeline
        p.add_stages(s)

        return p


if __name__ == '__main__':

        p1 = generate_pipeline(name='Pipeline 1', stages=2)

        res_dict = {

                'resource': 'xsede.bridges',
                'username': 'jel203' #username is not currently included in ensemble toolkit,
so an error arises
                'walltime': 10,
                'cores': 1,
                'project': '',
        }

        # Create Resource Manager object with the above resource description
        rman = ResourceManager(res_dict)

        # Create Application Manager
        appman = AppManager()

        # Assign resource manager to the Application Manager
        appman.resource_manager = rman

        # Execute pipeline
        appman.assign_workflow(p1)

        # Run the Application Manager
        appman.run()
```

The code above assumes that the user has the modules “sealnet_nas_scalable.py”, “terminalsetup.py”, “predictionsetup.py”, “prep_train.py”, “pt_predict.py” and

“blob_detector.py” in a folder named “Code” on the remote machine. Also, the user is required to have a set of images in a folder labeled “test_images”, a folder with training images labeled “nn_images” and a folder labeled “classified_images” which has 5 empty subfolders “weddell”, “pack-ice”, “crab-eater”, “emperor” and “other”. These subfolders will later be populated with predicted images.

As you can see, the pipeline is split into 2 stages, each composed of one task. The training and prediction models require python3, cuda and pytorch in order to run, so these are the only modules that need to be loaded. Stage 1 of the pipeline executed the neural network training using the batch script “trainbatch.bat”. This stage trains the neural network using the images in the folder “nn_images” and the python package pytorch. This stage of the pipeline executes the code “sealnet_nas_scalable” and takes around 3 hours to complete. The previous stage can be omitted once the training module is run once on a user’s account on the remote machine and the neural network model is saved. The second and final stage of the pipeline executes the prediction by running the batch script “predscript.bat”. This module is responsible for taking a set of images (which should be located in the nn_images folder on the host machine), identifying the seals in each image, placing the images in a corresponding folder with the seal names and downloading these images back to the local machine. It also runs the “blob_detector.py” module, which outputs a set of images with the seals circles in the image as well as a .csv file that notes that coordinate location of each seal picture by picture. In order for the images of the seals to appear while running “blob_detector.py” on Bridges, you must enable X11 forwarding when initially logging on to the machine. This is as simple as adding the argument “-x” to the ssh log in line presented in the “Method” section of this document. Furthermore, an X11 forwarding

client must be installed and running on your local machine. The runtime of these two images varies depending on the given input of images.

The code above runs into two errors with the Ensemble Toolkit.

The batch scripts used can be seen below:

trainbatch.bat:

```
#!/bin/bash
#SBATCH -N 2
#SBATCH -p GPU
#SBATCH --ntasks-per-node 28
#SBATCH -t 5:00:00
#SBATCH --gres=gpu:p100:2

echo "========JOB START========"
set -x

module load python3
module load cuda
module load pytorch
module load gcc

::move to working directory
cd $SCRATCH/SealCode

::run GPU program
python3 terminalsetup.py
```

predscript.bat:

```
#!/bin/bash
#SBATCH -N 2
#SBATCH -p GPU
#SBATCH --ntasks-per-node 28
#SBATCH -t 1:00:00
#SBATCH --gres=gpu:p100:2

echo "========JOB START========"
set -x

module load python3
module load cuda
module load opencv
module load gcc

cd $SCRATCH/predict_temp4

python3 predictionsetup.py

python3 pt_predict.py -class_names 'crabeater' 'weddel' 'pack-ice' 'other'

python3 blob_detector.py
```

Stream Delineation Code

```
#Code to run StreamExtraction Pipeline on Bridges
from radical.entk import Pipeline, Stage, Task, AppManager, ResourceManager
import os

# ------------------------------------------------------------------------------

if os.environ.get('RADICAL_ENTK_VERBOSE') == None:
        os.environ['RADICAL_ENTK_VERBOSE'] = 'INFO'


def generate_pipeline(name, stages):  #Generate Pipeline for Stream Extraction/Morphological
Thinning

    # Create a Pipeline object
    p = Pipeline()
    p.name = 'p1'

    for s_cnt in range(stages):

    # Create a Stage object, Stream Extraction
```

```
    s1 = Stage()
    s1.name = ‘Stage 1’

    # Create a Stage object, Morphological Thinning
    s2 = Stage()
    s2.name = ‘Stage 2’


    # Create Task 1, Stream Extraction
    t1 = Task()
    t1.name = 'Task 1'
    t1.executable = ['sbatch']   # Assign executable to the task
    t1.arguments = ['$SCRATCH/ashk96/StreamExtraction.bat'] # Assign arguments for the
StreamExtraction

    # Add Task 1 to Stage 1
    s1.add_tasks(t1)

    # Add Stage 1 to the Pipeline
    p.add_stages(s1)

    # Create Task 2, Morphological Thinning
    t2 = Task()
    t2.name = 'Task 2'
    t2.executable = ['sbatch']   # Assign executable to the task
    t2.arguments = ['$SCRATCH/ashk96/Thin_Multi.bat'] # Assign arguments for the task
executable


    # Add Task 2 to Stage 2
    s2.add_tasks(t2)

    # Add Stage 2 to the Pipeline
    p.add_stages(s2)

    return p


if __name__ == '__main__':


    p1 = generate_pipeline(name='Stream Extraction', stages=2)
    res_dict = {

            'resource': 'xsede.bridges',
            'walltime': 02:00:00,
            'cores': 2,
            'project': 'TG-MCB090174',
            'queue': '',
            'schema': 'ssh'

}
    # Create Resource Manager object with the above resource description
    rman = ResourceManager(res_dict)

    # Create Application Manager
    appman = AppManager()

    # Assign resource manager to the Application Manager
    appman.resource_manager = rman
```

```
# Execute pipeline
appman.assign_workflow(p1)

# Run the Application Manager
appman.run()
```

The script above generates a pipeline with two stages; Stage 1 for Stream Extraction and Stage 2 for Morphological Thinning. The necessary batch files needed to be run are located within the "StreamExtraction" directory on Bridges. In the first stage, the code for Stream Extraction is run by creating a task for the process and adding it to Stage 1. Stage 1 is then added to the Pipeline. Next, Stage 2 is created for Morphological Thinning. The code for thinning is modeled as Task 2 and also found on the "StreamExtraction" directory. Next, Task 2 is added to Stage 2 and Stage 2 is added to the Pipeline. After generating the pipeline, a resource dictionary is created to request the necessary server time. A walltime of 2 hours on 2 cores is used, and will be run on the Bridges RM-shared CPU. The project charge # is given in our XSEDE allocation, and the schema to login is ssh.

The following are the two batch scripts used to run the respective programs on Bridges:

StreamExtraction.bat

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -p RM-shared
#SBATCH -t 02:00:00
#SBATCH --ntasks-per-node 2

# echo commands to stdout
echo 'JOB START'
set -x

module load mono
module load OSGeo

# move to working directory
cd $SCRATCH

#run StreamExtraction program
mono StreamExtraction.exe
```

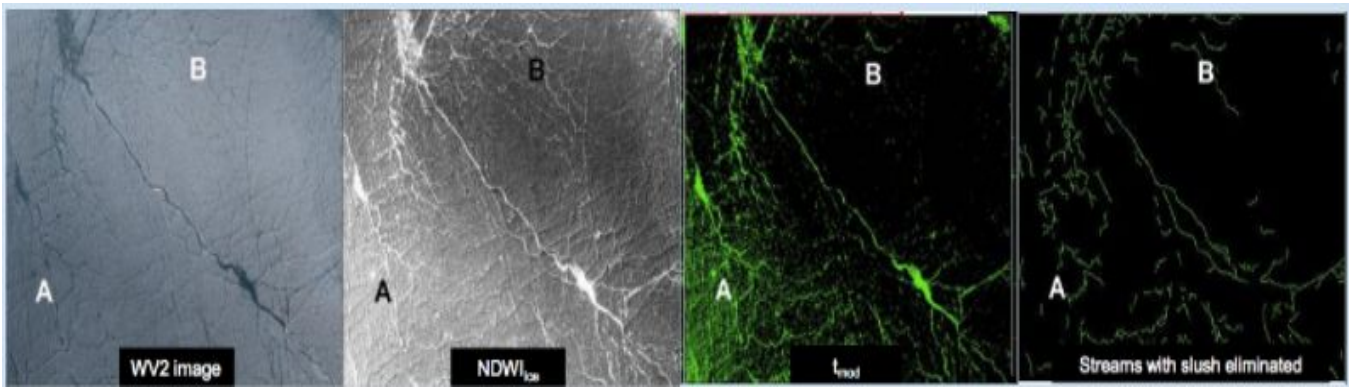
Thin_multi.bat

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -p RM-shared
#SBATCH -t 5:00:00
#SBATCH --ntasks-per-node 2

# echo commands to stdout
echo 'JOB START'
set -x

module load python3
module load arcpy

# move to working directory
cd $SCRATCH/StreamExtraction

# run Morphological Thinning program
python3 Thin_multi.py
```

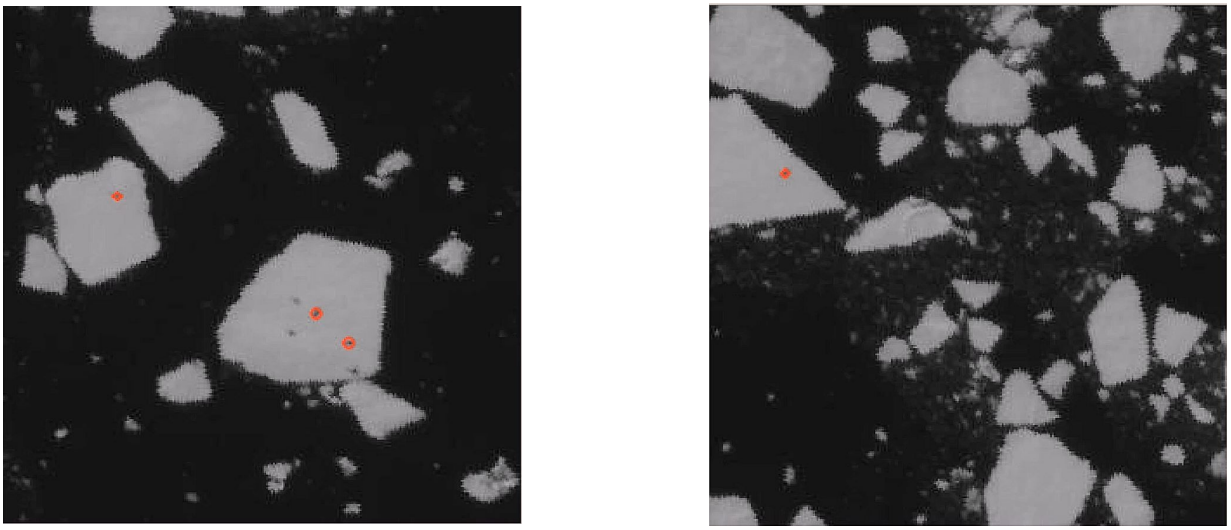
The "#SBATCH" commands must be included at the start of the code, and include details as to what partition the user requests on Bridges, the walltime, # of cores and # of tasks per node. Then, the commands are echo'd to "stdout" and the two necessary modules are loaded. Finally, the user moves to the working "StreamExtraction" directory and runs the program (mono for "StreamExtraction", python3 for Thin_multi.

Results

A full run of the pipeline has been completed on the Bridges machine. Once all the code was received, transferred to Bridges and the environment was setup correctly, execution was achieved. The results show the images from the "test_images" folder being placed in the correct subfolders in the "classified_images" folder. Also, the blob detector gave us a set of images with the seals circled in the images. The group was never were able to successfully integrate the Ensemble Toolkit due to setbacks encountered.



The above image shows the the different stages of the image transformation, from original WV2 image through to the stream extraction. The streams are highlighted in green and are delineated from all the other unnecessary features seen in the original WV2 image.



Output of the blob detector (seals circled in red)

Future

In the current state, the pipeline still needs to be integrated with the Ensemble Toolkit. This step cannot be achieved until the radical.entk modules are updated. These specific modules

being “wfprocessor.py” and “resource_manager.py”. Once these two modules are updated, the user can login with their correct username and the processes can terminate properly. Modules such as Arcpy also still require installation and implementation within Bridges.

Over the next few years, the ICEBERG team will continue to integrate the Ensemble Toolkit to pipeline multiple different use cases across various scientific disciplines, including geological land cover and geomorphology. The goal is to have all the use cases run on the different XSEDE supercomputers and create an easy interface for scientists to perform their research on, regardless of their coding experience. The project will be a major contribution to the sciences and especially will help researchers combat the ongoing global warming crisis.

Conclusion

Regardless of whether or not the group was able to integrate the Ensemble Toolkit, a lot was learned from this project. Before working with the ICEBERG group, not a single member of the Capstone group had even logged into a supercomputer. Now, every member is comfortable logging into, setting up an environment and running code on an HPC environment. Working on on this project gave the group much insight to the powerful tools in the world of computing. Being able to physically train a neural network and watching it work on a set of images gave the group knowledge on how this process works and familiarity with the Python packages involved in doing this. The group may not have entirely achieved its end goal, but a lot of valuable information was acquired from the experience.

Acknowledgements

The group would like to give its thanks to Heather Lynch and Bento from Stony Brook, the ICEBERG team, XSEDE team, Pittsburgh Supercomputing Center, National Science Foundation, RADICAL team and, of course, Professor Jha and Professor Turilli.

References

[1]https://www.psc.edu/resources/computing/bridges
[2]https://github.com/iceberg-project
[3]https://radical-cybertools.github.io/
[4]http://radicalentk.readthedocs.io/en/latest/index.html