NYU HPC Cloud Bursting Tutorial

Course: CSCI-UA.0480-051 (Fall 2025)

Contents

1 Your Resource Allocation

Each student has been assigned:

- **Slurm Account:** `csci_ua_0480_051-2025fa`
- **GPU Hours:** 300 hours (18,000 minutes)
- **CPU Time:** Sufficient for coursework

1.1 Available Partitions

- `interactive` - For CPU-only interactive jobs
- `n2c48m24` - CPU partition
- `g2-standard-12`, `g2-standard-24`, `g2-standard-48` - GPU partitions with L4 GPUs
- `c12m85-a100-1`, `c24m170-a100-2` - A100 GPU partitions
- `n1s8-t4-1` - T4 GPU partition

2 Prerequisites

2.1 Network Connection

VPN Requirement

You need to connect to NYU VPN **only if you are NOT on NYU campus WiFi**. If you are on campus, you can skip the VPN setup.
For off-campus access: Download NYU VPN from `https://www.nyu.edu/life/information-technology/infrastructure/network-services/vpn.html`

3 Quick Start Guide

3.1 Method 1: Using Open OnDemand (Recommended)

This is the easiest way to get started and manage files - no complex command line setup required!

3.1.1 Step 1: Access Open OnDemand

- Connect to NYU VPN (if off-campus) or NYU WiFi (if on campus)
- Go to: `https://ood-burst-001.hpc.nyu.edu/`
- Log in with your NYU credentials (NetID and password)

What You Can Do with Open OnDemand

From the OOD server, you can:

- Launch compute nodes without logging into Greene cluster
- Run Jupyter notebooks (only if needed)
- Open terminal sessions directly in your browser
- Transfer files between your local computer and HPC
- Manage files with a graphical interface

3.1.2 Step 2: Upload Your Files

1. From the Open OnDemand homepage, click **Files** → **Home Directory**
2. Click the **Upload** button

3. Select your MPI or CUDA source files from your computer
4. Your files will be uploaded to your home directory on HPC

3.1.3 Step 3: Open a Terminal

1. Click **Clusters → Greene Shell Access**
2. This opens a terminal in your browser
3. Run `ssh burst` to connect to Cloud Burst
4. Now you can compile and run your programs

3.2 Method 2: Using Terminal/SSH

For users comfortable with command line access.

3.2.1 SSH Connection

Step 1: SSH to Greene Jump Host

```
ssh netID@gw.hpc.nyu.edu
```

Replace `netID` with your actual NYU NetID. For example, if your NetID is `ab1234`:

```
ssh ab1234@gw.hpc.nyu.edu
```

Step 2: SSH to Greene Cluster

```
ssh netID@greene.hpc.nyu.edu
```

Step 3: SSH to Cloud Burst

```
ssh burst
```

> **Note**
>
> When connecting for the first time, you may see: "This key is not known by any other names. Are you sure you want to continue connecting?"
>
> - Type `yes` and press Enter

3.3 Method 3: Using VS Code Remote SSH

VS Code provides an excellent development environment with syntax highlighting, IntelliSense, and integrated terminal.

3.3.1 Step 1: Install VS Code Extension

1. Download VS Code from `https://code.visualstudio.com/`
2. Install the "Remote - SSH" extension

3.3.2 Step 2: Configure SSH

Click on the icon on the bottom left corner: follow fig 1 then fig 2

Then add this to your SSH config file (`~/.ssh/config`):

```
# NYU HPC Gateway
Host greene.hpc.nyu.edu
  HostName greene.hpc.nyu.edu
  User <net_id>
```

```
Host greene.hpc.nyu.edu dtn.hpc.nyu.edu gw.hpc.nyu.edu
  StrictHostKeyChecking no
  ServerAliveInterval 60
  ForwardAgent yes
  StrictHostKeyChecking no
  UserKnownHostsFile /dev/null
  LogLevel ERROR

Host hpcgwtunnel
  HostName gw.hpc.nyu.edu
  ForwardX11 no
  StrictHostKeyChecking no
  LocalForward 8027 greene.hpc.nyu.edu:22
  UserKnownHostsFile /dev/null
  User <net_id>

Host greene
  HostName localhost
  Port 8027
  ForwardX11 yes
  StrictHostKeyChecking no
  UserKnownHostsFile /dev/null
  LogLevel ERROR
  User <net_id>

Host burst.hpc.nyu.edu
  HostName burst.hpc.nyu.edu
  User <netid>
  ServerAliveInterval 60
  ForwardAgent yes
```

Replace `<net_id>` with your actual NetID.

3.3.3 Step 3: Connect via VS Code

1. Press `F1` or `Ctrl+Shift+P`
2. Type "Remote-SSH: Connect to Host"
3. Select `greene.hpc.nyu.edu`
4. Enter your NYU password
5. Open folder: `/scratch/your_netid`
6. Start new terminal and run `ssh burst`. look at fig 3 for reference.
7. Edit, compile, and run code directly in VS Code

4 Running Interactive Jobs

4.1 CPU Job for MPI Programs

IMPORTANT: Specify CPUs

When running MPI programs, you **MUST** specify the number of CPUs you need using `--cpus-per-task=N` or `--ntasks=N`, where N is the number of MPI processes you plan to run. Otherwise, you'll only get 1 CPU and cannot run parallel programs.

For 8 MPI processes:

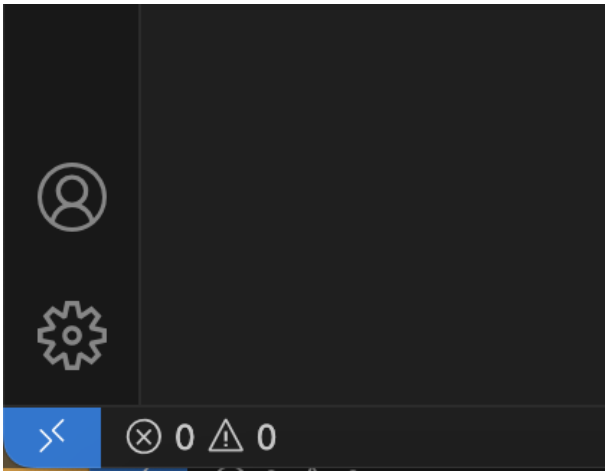

Figure 1: Click this icon to open a dialogue box

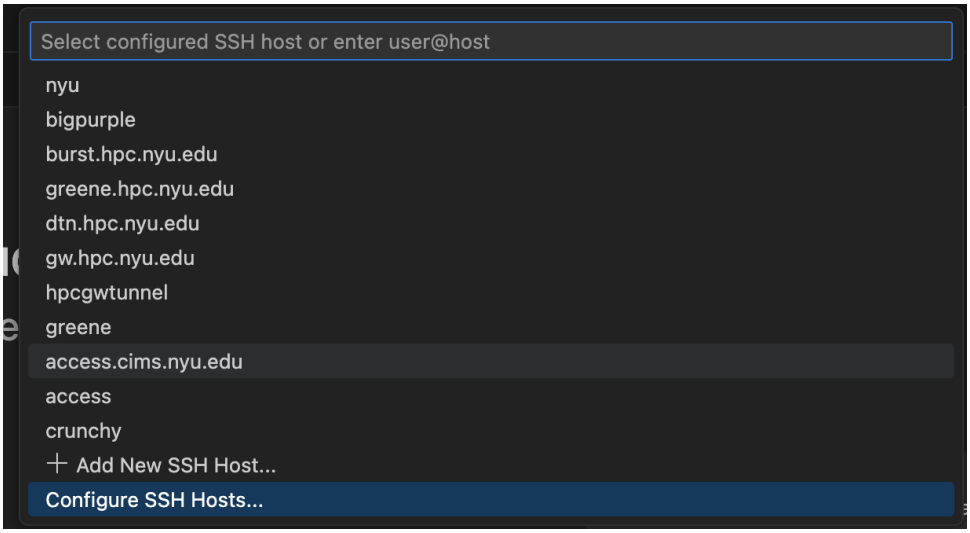

Figure 2: click on configure SSH Hosts (only for first time setup)

```
srun --account=csci_ua_0480_051-2025fa --partition=n2c48m24 --cpus-per-task=8 --time
    =04:00:00 --pty /bin/bash
```

Alternative with ntasks:

```
srun --account=csci_ua_0480_051-2025fa --partition=n2c48m24 --ntasks=2 --time=04:00:00
     --pty /bin/bash
```

4.2 GPU Jobs for CUDA Programs

1 L4 GPU for 4 Hours (Recommended):

```
srun --account=csci_ua_0480_051-2025fa --partition=g2-standard-12 --gres=gpu:1 --time
    =04:00:00 --pty /bin/bash
```

1 A100 GPU for 4 Hours:

```
srun --account=csci_ua_0480_051-2025fa --partition=c12m85-a100-1 --gres=gpu --time
    =04:00:00 --pty /bin/bash
```

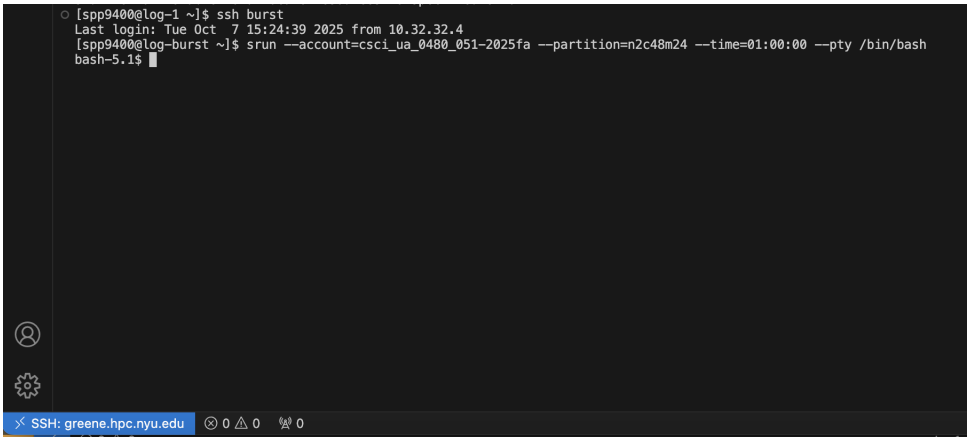

Figure 3: This is how it will look like

5 Setting Up Singularity for MPI Programs

REQUIRED for MPI Programs

Singularity is REQUIRED for MPI programs on burst nodes! The burst compute nodes have an incomplete MPI installation that causes `libpmix.so.2` errors. You must use Singularity containers to run MPI programs successfully.

5.1 One-Time Setup

You only need to do this setup once. After that, you can reuse the same Singularity container and overlay.

5.1.1 Step 1: Export PATH and Navigate to Scratch Directory

```
# Check if OpenMPI packages are available via the system package manager
which mpicc
ls /usr/lib64/openmpi/bin/
# If it exists in /usr/lib64/openmpi/bin/, you can add it to your PATH:
export PATH=/usr/lib64/openmpi/bin:$PATH
export LD_LIBRARY_PATH=/usr/lib64/openmpi/lib:$LD_LIBRARY_PATH

# Then try:
which mpicc
which mpirun
# After getting your compute node, go to scratch
cd /scratch/<your_netid> or cd ../../scratch/<your_netid>
# Example: cd /scratch/ab1234, based on your location: use pwd to check
```

5.1.2 Step 2: Download Singularity Image

```
# Download the Ubuntu Singularity image
scp -rp greene-dtn:/scratch/work/public/singularity/ubuntu-20.04.3.sif .
```

This will take a few minutes as the file is several GB.

5.1.3 Step 3: Download and Extract Overlay File

```
# Download overlay template
scp -rp greene-dtn:/scratch/work/public/overlay-fs-ext3/overlay-15GB-500K.ext3.gz .

# Extract it
gunzip overlay-15GB-500K.ext3.gz
```

This will take a few minutes as the file is several GB.

5.1.4 Step 4: Launch Singularity Container

```
# Start the container with overlay
singularity exec --overlay overlay-15GB-500K.ext3:rw ubuntu-20.04.3.sif /bin/bash
```

Your prompt will change to show you're inside Singularity (e.g., `Singularity>`).

5.1.5 Step 5: Install Miniconda (One-Time Only)

```
# Inside Singularity container
# Download Miniconda
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh

# Install to /ext3 (the overlay filesystem)
bash Miniconda3-latest-Linux-x86_64.sh -b -p /ext3/miniconda3

# Activate conda
source /ext3/miniconda3/bin/activate
```

5.1.6 Step 6: Install OpenMPI via Conda (One-Time Only)

```
# Still inside Singularity, with conda activated
# accepting the terms and conditions the first time
conda tos accept --override-channels --channel https://repo.anaconda.com/pkgs/main
conda tos accept --override-channels --channel https://repo.anaconda.com/pkgs/r
conda install gcc_linux-64 gxx_linux-64 openmpi -y
```

This installs a working MPI implementation that doesn't have the `libpmix` issues.

5.2 Regular Usage After Setup

After the one-time setup, your workflow for each session is:

```
# 1. Get compute node with enough CPUs
srun --account=csci_ua_0480_051-2025fa --partition=n2c48m24 --cpus-per-task=8 --time
    =04:00:00 --pty /bin/bash

# 2. Go to your scratch directory
cd /scratch/<your_netid>
# or
cd ../../scratch/<your_netid> if you are in home directory

# 3. Launch Singularity (note: using :ro for read-only)
singularity exec --overlay overlay-15GB-500K.ext3:ro ubuntu-20.04.3.sif /bin/bash

# 4. Activate conda
```

```
source /ext3/miniconda3/bin/activate

# 5. Now you can compile and run MPI programs!
mpicc -o hello_mpi hello_mpi.c
mpirun -np 2 ./hello_mpi
```

Pro Tip

Use `:ro` (read-only) instead of `:rw` when you don't need to install new software. This is safer and prevents accidental modifications.

6 Running MPI Programs

6.1 Complete MPI Workflow

```
# 1. SSH to burst
ssh <netid>@gw.hpc.nyu.edu
ssh <netid>@greene.hpc.nyu.edu
ssh burst

# 2. Request compute node with CPUs matching your MPI process count
srun --account=csci_ua_0480_051-2025fa --partition=n2c48m24 --cpus-per-task=8 --time
    =04:00:00 --pty /bin/bash

# 3. Go to scratch and launch Singularity
cd /scratch/<netid>
singularity exec --overlay overlay-15GB-500K.ext3:ro ubuntu-20.04.3.sif /bin/bash

# 4. Activate conda
source /ext3/miniconda3/bin/activate

# 5. Create your MPI program
vi hello_mpi.c

# 6. Compile with mpicc
mpicc -o hello_mpi hello_mpi.c

# 7. Run with mpirun
mpirun -np 2 ./hello_mpi

# 8. When done, exit
exit  # Exit Singularity
exit  # Exit compute node
```

6.2 Important Notes for MPI

- **Match CPUs to processes:** If you run `mpirun -np 8`, you need `--cpus-per-task=8` or more
- **Always use Singularity:** Running `mpicc/mpirun` directly on burst nodes will fail with `libpmix` errors
- **Activate conda:** Don't forget `source /ext3/miniconda3/bin/activate` inside Singularity
- **Work in /scratch:** Store your code in `/scratch/<netid>` for better performance

7 Running CUDA Programs

7.1 Basic CUDA Workflow

```
# 1. Request GPU node
srun --account=csci_ua_0480_051-2025fa --partition=g2-standard-12 --gres=gpu:1 --time
    =04:00:00 --pty /bin/bash

# 2. Load CUDA module
module load cuda/11.8.0

# 3. Verify GPU
nvidia-smi

# 4. Create your CUDA program
vi hello_cuda.cu

# 5. Compile
nvcc -o hello_cuda hello_cuda.cu

# 6. Run
./hello_cuda
```

7.2 Available CUDA Modules

Check available CUDA versions:

```
module avail cuda
```

Common versions:

- `cuda/11.8.0` - Recommended for most programs
- `cuda/12.0` - For newer GPU features

8 Example Programs

8.1 Example 1: Basic MPI Hello World (WITH Singularity)

This is the complete, corrected workflow that actually works on burst nodes.

Step 1: Create the MPI program

```
// hello_mpi.c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int world_size, world_rank;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    printf("Hello from processor %s, rank %d out of %d\n",
```

```
          processor_name, world_rank, world_size);

    MPI_Finalize();
    return 0;
}
```

Step 2: Complete execution workflow

```
# Connect to burst
ssh ab1234@gw.hpc.nyu.edu
ssh ab1234@greene.hpc.nyu.edu
ssh burst

# Get compute node with 8 CPUs
srun --account=csci_ua_0480_051-2025fa --partition=n2c48m24 --cpus-per-task=8 --time
    =04:00:00 --pty /bin/bash

# Navigate to scratch
cd /scratch/ab1234

# Launch Singularity (assuming you did the one-time setup)
singularity exec --overlay overlay-15GB-500K.ext3:ro ubuntu-20.04.3.sif /bin/bash

# Activate conda
source /ext3/miniconda3/bin/activate

# Create the program
vi hello_mpi.c
# (paste the code above)

# Compile
mpicc -o hello_mpi hello_mpi.c

# Run with 2 processes
mpirun -np 2 ./hello_mpi

# Expected output:
# Hello from processor b-9-77, rank 0 out of 8
# Hello from processor b-9-77, rank 1 out of 8
# Hello from processor b-9-77, rank 2 out of 8
# ... (8 lines total)

# Exit when done
exit  # Exit Singularity
exit  # Exit compute node
```

8.2 Example 2: CUDA Vector Addition

```
// vector_add.cu
#include <stdio.h>
#include <cuda_runtime.h>

__global__ void vectorAdd(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];
    }
```

```
}

int main() {
    int n = 1000;
    size_t size = n * sizeof(float);

    // Allocate host memory
    float *h_a = (float*)malloc(size);
    float *h_b = (float*)malloc(size);
    float *h_c = (float*)malloc(size);

    // Initialize vectors
    for (int i = 0; i < n; i++) {
        h_a[i] = i;
        h_b[i] = i * 2;
    }

    // Allocate device memory
    float *d_a, *d_b, *d_c;
    cudaMalloc(&d_a, size);
    cudaMalloc(&d_b, size);
    cudaMalloc(&d_c, size);

    // Copy to device
    cudaMemcpy(d_a, h_a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, h_b, size, cudaMemcpyHostToDevice);

    // Launch kernel
    int threadsPerBlock = 256;
    int blocksPerGrid = (n + threadsPerBlock - 1) / threadsPerBlock;
    vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d_a, d_b, d_c, n);

    // Copy result back
    cudaMemcpy(h_c, d_c, size, cudaMemcpyDeviceToHost);

    // Verify result
    printf("First 5 results:\n");
    for (int i = 0; i < 5; i++) {
        printf("%f + %f = %f\n", h_a[i], h_b[i], h_c[i]);
    }

    // Cleanup
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);
    free(h_a);
    free(h_b);
    free(h_c);

    return 0;
}
```

To run:

```
# Get GPU node (NO Singularity needed for CUDA)
srun --account=csci_ua_0480_051-2025fa --partition=g2-standard-12 --gres=gpu:1 --time
    =02:00:00 --pty /bin/bash

# Load CUDA
```

```
module load cuda/11.8.0

# Compile and run
nvcc -o vector_add vector_add.cu
./vector_add
```

8.3 Example 4: Complete Singularity Setup from Scratch

This shows the complete one-time setup process:

```
# 1. Connect to burst and get compute node
ssh ab1234@gw.hpc.nyu.edu
ssh ab1234@greene.hpc.nyu.edu
ssh burst
srun --account=csci_ua_0480_051-2025fa --partition=n2c48m24 --cpus-per-task=8 --time
    =02:00:00 --pty /bin/bash

# 2. Go to scratch directory
cd /scratch/ab1234

# 3. Download Singularity image
scp -rp greene-dtn:/scratch/work/public/singularity/ubuntu-20.04.3.sif .

# 4. Download and extract overlay
scp -rp greene-dtn:/scratch/work/public/overlay-fs-ext3/overlay-15GB-500K.ext3.gz .
gunzip overlay-15GB-500K.ext3.gz

# 5. Launch Singularity with read-write overlay
singularity exec --overlay overlay-15GB-500K.ext3:rw ubuntu-20.04.3.sif /bin/bash

# 6. Inside Singularity: Install Miniconda
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh -b -p /ext3/miniconda3

# 7. Activate conda
source /ext3/miniconda3/bin/activate

# 8. Install OpenMPI
conda install -c conda-forge openmpi -y

# 9. Test it works
echo '#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    printf("Hello from rank %d\n", rank);
    MPI_Finalize();
    return 0;
}' > test_mpi.c

mpicc -o test_mpi test_mpi.c
mpirun -np 4 ./test_mpi

# 10. If you see 4 lines of output, setup is complete!
# Exit and use :ro for future sessions
exit
```

9 Batch Job Scripts

9.1 MPI Batch Job

```
#!/bin/bash
#SBATCH --job-name=mpi_job
#SBATCH --account=csci_ua_0480_051-2025fa
#SBATCH --partition=n2c48m24
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --time=01:00:00
#SBATCH --output=%j_%x.out
#SBATCH --error=%j_%x.err
#SBATCH --requeue

# Navigate to scratch
cd /scratch/$USER

# Launch Singularity with conda and run program
singularity exec --overlay overlay-15GB-500K.ext3:ro \
    ubuntu-20.04.3.sif /bin/bash << 'EOF'
source /ext3/miniconda3/bin/activate
mpirun -np 32 ./my_mpi_program
EOF
```

9.2 CUDA Batch Job

```
#!/bin/bash
#SBATCH --job-name=cuda_job
#SBATCH --account=csci_ua_0480_051-2025fa
#SBATCH --partition=g2-standard-12
#SBATCH --gres=gpu:1
#SBATCH --time=01:00:00
#SBATCH --output=%j_%x.out
#SBATCH --error=%j_%x.err
#SBATCH --requeue

module load cuda/11.8.0

echo "GPU Information:"
nvidia-smi
echo ""

echo "Running CUDA program..."
./my_cuda_program
```

10 Troubleshooting

10.1 Common MPI Errors and Solutions

10.1.1 Error: libpmix.so.2: cannot open shared object file

Problem: You tried to run MPI programs directly without Singularity.

Solution: Always use Singularity with conda for MPI programs:

```
singularity exec --overlay overlay-15GB-500K.ext3:ro ubuntu-20.04.3.sif /bin/bash
source /ext3/miniconda3/bin/activate
mpirun -np 8 ./program
```

10.1.2 FATAL: while loading overlay images: failed to open overlay image

Problem: The overlay file might have a stale lock from a previous session that didn't exit cleanly.

Solution: Reboot Your Session:

```
mv overlay-15GB-500K.ext3 overlay-15GB-500K.ext3.backup
scp -rp greene-dtn:/scratch/work/public/overlay-fs-ext3/overlay-15GB-500K.ext3.gz .
gunzip overlay-15GB-500K.ext3.gz
singularity exec --overlay overlay-15GB-500K.ext3:rw ubuntu-20.04.3.sif /bin/bash
```

10.1.3 Error: Not enough slots available

Problem: You didn't request enough CPUs when launching your job.

Solution: Match CPU count to MPI processes:

```
# If running mpirun -np 8, need at least 8 CPUs:
srun --account=csci_ua_0480_051-2025fa --partition=n2c48m24 --cpus-per-task=8 --time
    =04:00:00 --pty /bin/bash
```

10.1.4 Error: File not found when copying Singularity image

Problem: Wrong file path.

Solution: Use the correct path:

```
# CORRECT path:
scp -rp greene-dtn:/scratch/work/public/singularity/ubuntu-20.04.3.sif .

# NOT this:
# scp -rp greene-dtn:/share/apps/images/ubuntu-20.04.3.sif .
```

10.1.5 Error: command not found: conda

Problem: Forgot to activate conda inside Singularity.

Solution: Always run after entering Singularity:

```
source /ext3/miniconda3/bin/activate
```

10.2 For CUDA Programs

10.2.1 Error: nvcc: command not found

Solution: Load CUDA module first:

```
module load cuda/11.8.0
```

All "module load" commands are for greene and not burst.

10.2.2 Error: No CUDA-capable device

Problem: Not on a GPU node.

Solution: Request GPU node:

```
srun --account=csci_ua_0480_051-2025fa --partition=g2-standard-12 --gres=gpu:1 --time
    =04:00:00 --pty /bin/bash
```

10.3 SSH Host Key Issues

If you see "Host key verification failed" when copying from greene-dtn:

```
# Remove old host key
ssh-keygen -R greene-dtn
ssh-keygen -R greene-dtn.hpc.nyu.edu

# Then retry your scp command
```

11 Debugging Tips

11.1 For MPI Programs

- **Start with 1 process:** `mpirun -np 1 ./program` to verify basic functionality
- **Add rank-specific output:** Help identify which process has issues
- **Use printf with fflush:** Ensures output appears immediately
- **Check MPI return codes:** Many MPI functions return error codes

Example debugging code:

```
printf("Rank %d: Before MPI_Send\n", rank);
fflush(stdout);
int result = MPI_Send(...);
if (result != MPI_SUCCESS) {
    printf("Rank %d: MPI_Send failed with code %d\n", rank, result);
}
printf("Rank %d: After MPI_Send\n", rank);
fflush(stdout);
```

11.2 For CUDA Programs

- **Check every CUDA call:** Don't skip error checking
- **Use cuda-memcheck:** Detects memory errors
- **Start with CPU version:** Verify algorithm before GPU
- **Use small test cases:** Easier to debug

Example error checking:

```
cudaError_t err = cudaMalloc(&d_array, size);
if (err != cudaSuccess) {
    printf("CUDA Error: %s\n", cudaGetErrorString(err));
    exit(1);
}

// After kernel launch
err = cudaGetLastError();
if (err != cudaSuccess) {
```

```
    printf("Kernel launch error: %s\n", cudaGetErrorString(err));
}
```

11.3 General Debugging

- **Compile with debug symbols:** `-g` flag
- **Check you're in the right environment:** Inside Singularity for MPI, GPU node for CUDA
- **Verify file locations:** Use `ls` and `pwd`
- **Check resource allocation:** Use `squeue -u $USER`

12 Time and Resource Management

12.1 GPU Hour Budget

With 300 GPU hours for the semester:

- **Development:** 15-30 min sessions for debugging
- **Testing:** 1 hour sessions for testing
- **Production:** 4+ hour sessions for final runs
- **Monitor usage:** Check regularly with `sacct`

Check your usage:

```
# View recent jobs
sacct -u $USER --format=JobID,JobName,Partition,Elapsed,State

# View all jobs since semester start
sacct -u $USER --format=JobID,JobName,Elapsed,State,AllocGRES -S 2025-01-01
```

12.2 Best Practices

- **Test interactively first:** Debug before submitting batch jobs
- **Use appropriate partitions:** CPU for MPI, GPU for CUDA
- **Request appropriate time:** Don't request 4 hours if you only need 30 minutes
- **Use –requeue:** Allows jobs to restart if nodes shut down
- **Save checkpoints:** For long-running jobs

13 Quick Reference

13.1 Essential Commands

```
# SSH to burst
ssh <netid>@gw.hpc.nyu.edu
ssh <netid>@greene.hpc.nyu.edu
ssh burst

# Interactive MPI job (with Singularity)
srun --account=csci_ua_0480_051-2025fa --partition=n2c48m24 --cpus-per-task=8 --time
    =04:00:00 --pty /bin/bash
cd /scratch/$USER
singularity exec --overlay overlay-15GB-500K.ext3:ro ubuntu-20.04.3.sif /bin/bash
source /ext3/miniconda3/bin/activate

```

```
# Interactive GPU job
srun --account=csci_ua_0480_051-2025fa --partition=g2-standard-12 --gres=gpu:1 --time
    =04:00:00 --pty /bin/bash
module load cuda/11.8.0

# Submit batch job
sbatch job_script.sh

# Check job status
squeue -u $USER

# Cancel job
scancel <job_id>

# View job history
sacct -u $USER
```

13.2 MPI Workflow Summary

```
# 1. Get compute node with CPUs
srun --account=csci_ua_0480_051-2025fa --partition=n2c48m24 --cpus-per-task=8 --time
    =04:00:00 --pty /bin/bash

# 2. Navigate to scratch
cd /scratch/<netid>

# 3. Launch Singularity
singularity exec --overlay overlay-15GB-500K.ext3:ro ubuntu-20.04.3.sif /bin/bash

# 4. Activate conda
source /ext3/miniconda3/bin/activate

# 5. Compile and run
mpicc -o program program.c
mpirun -np 2 ./program
```

13.3 CUDA Workflow Summary

```
# 1. Get GPU node
srun --account=csci_ua_0480_051-2025fa --partition=g2-standard-12 --gres=gpu:1 --time
    =04:00:00 --pty /bin/bash

# 2. Load CUDA
module load cuda/11.8.0

# 3. Compile and run
nvcc -o program program.cu
./program
```

14 Summary

14.1 Key Takeaways

- **Singularity is REQUIRED for MPI:** Burst nodes have broken MPI, use Singularity + conda

- **Singularity NOT needed for CUDA:** CUDA works directly with module load
- **Specify CPUs for MPI:** Use `--cpus-per-task=N` matching your MPI process count
- **Use correct file paths:** `/scratch/work/public/singularity/...`
- **Activate conda in Singularity:** `source /ext3/miniconda3/bin/activate`
- **Work in /scratch:** Better performance than /home
- **Open OnDemand is easiest:** `https://ood-burst-001.hpc.nyu.edu/`
- **Use –requeue:** In all batch scripts for spot instances
- **Monitor GPU hours:** 300 hours total for semester
- **Test interactively first:** Debug before batch submission

14.2 Recommended Workflow

1. Develop code locally or using VS Code Remote-SSH
2. Upload to HPC via Open OnDemand or SCP
3. For MPI: Set up Singularity once, then reuse
4. Test interactively with small inputs
5. Debug and verify correctness
6. Create batch script for production runs
7. Submit batch job with `--requeue`
8. Monitor with `squeue` and download results

14.3 Important Links

- **Open OnDemand:** `https://ood-burst-001.hpc.nyu.edu/`
- **NYU HPC Docs:** `https://sites.google.com/nyu.edu/nyu-hpc/`
- **Singularity Guide:** `https://sites.google.com/nyu.edu/nyu-hpc/hpc-systems/greene/software/singularity-with-miniconda`
- **HPC Support:** hpc@nyu.edu

Good Luck!

You now have everything you need to successfully use HPC Cloud Bursting! Remember:

- MPI requires Singularity (one-time setup, then easy to use)
- CUDA works directly without Singularity
- Always specify CPUs when running MPI programs
- Start early and test often
- Ask questions when stuck - check Brightspace or email hpc@nyu.edu
- Have fun learning parallel computing!