Running CUDA programs on Discovery

1. Login to Discovery using ssh: ssh -X username@discovery.neu.edu
2. Inspect your .bashrc file to ensure you have the proper modules loaded. Your .bashrc file should have the following lines included:
   module load openmpi
   module load cuda/9.2 (you can chang the cuda version)
3. Write your CUDA program. Below is a simple CUDA program for vector addition from Oak Ridge National Labs.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
__global__ void vecAdd(double *a, double *b, double *c, int n) // CUDA kernel
{
    int id = blockIdx.x*blockDim.x+threadIdx.x; // Get global thread ID
    if (id < n) // Make sure we do not go out of bounds
        c[id] = a[id] + b[id];
}
 int main( int argc, char* argv[] )
{
    int n = 100000; // Size of vectors
    double *h_a, *h_b;// Host input vectors
    double *h_c; // Host output vector
    double *d_a, *d_b; // Device input vectors
    double *d_c; // Device output vector
    size_t bytes = n*sizeof(double); // Size, in bytes, of each vector
    h_a = (double*)malloc(bytes); // Allocate memory for each vector on host
    h_b = (double*)malloc(bytes);
    h_c = (double*)malloc(bytes);
    cudaMalloc(&d_a, bytes); // Allocate memory for each vector on GPU
    cudaMalloc(&d_b, bytes);
    cudaMalloc(&d_c, bytes);
     int i;
    for( i = 0; i < n; i++ ) { // Initialize vector on host
        h_a[i] = sin(i)*sin(i);
        h_b[i] = cos(i)*cos(i);
     }
    cudaMemcpy( d_a, h_a, bytes, cudaMemcpyHostToDevice); //Copy vectors to device
    cudaMemcpy( d_b, h_b, bytes, cudaMemcpyHostToDevice);
     int blockSize, gridSize;
    blockSize = 1024; // Number of threads in each thread block
    gridSize = (int)ceil((float)n/blockSize); // Number of thread blocks in grid
     // Execute the kernel
    vecAdd<<<gridSize, blockSize>>>(d_a, d_b, d_c, n); // Execute kernel
    cudaMemcpy( h_c, d_c, bytes, cudaMemcpyDeviceToHost ); //Copy back to host
     // Sum up vector c and print result divided by n
    double sum = 0;
    for(i=0; i<n; i++)
        sum += h_c[i];
    printf("final result: %f\n", sum/n); // The answer should be 1.0
    cudaFree(d_a); // Release device memory
    cudaFree(d_b);
    cudaFree(d_c);
    free(h_a); // Release host memory
    free(h_b);
    free(h_c);
    return 0;
}
```

To get an interactive node that has a GPU:

srun --partition=gpu --nodes=1 --pty --export=All --gres=gpu:1 --mem=1G --time=00:30:00 /bin/bash

To run CUDA program in batch:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --time=0:10:00
#SBATCH --job-name=your_job_name
#SBATCH --mem=1G
#SBATCH --gres=gpu:1
#SBATCH --output=your_program_output
#SBATCH --partition=gpu
./name_of_your_CUDA_binary
```

If you need a specific GPU device (e.g., V100, P100) or need to use multiple GPUs, you can find information on this page:

https://rc-docs.northeastern.edu/en/latest/using-discovery/workingwithgpu.html

You can specify a specific type of GPU in the –gres command:

For example: --gres=gpu:k40m:1