Running CUDA on the Discovery Cluster

March 2020

Note: This document can become outdated quickly due to the evolving nature of software installed on the Discovery Cluster. Every effort will be made to keep it up to date.

1. Login to Discovery using ssh: `ssh –X` ***`username`***`@discovery.neu.edu`.
2. Inspect your .bashrc file to insure you have the proper modules loaded. Your .bashrc looks like this:
   ```
   module load openmpi
   module load cuda/9.0
   ```
3. Write your CUDA program. Below is simple CUDA program for a 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;
}
```

Please try to do a majority of your editing and compiling on a login node, not on the node where you will run your code interactively. The compute nodes are shared, so be respectful of others.

4. Compile your CUDA code using the NVIDIA compiler as follows:
   ```
   nvcc vecadd.cu –o vecadd
   ```
5. You are now ready to run your program. Issue the following command to get an interactive node to run in the eece5640 partition:

   ```
   srun --pty --nodes 1 --job-name=interactive --partition=gpu
   --reservation=EECE5640 --gres=gpu:1 /bin/bash
   ```

6. Then run your CUDA program as follows: `./vecadd`

   or you can directly run your code:

   ```
   srun --pty --nodes 1 --job-name=interactive --partition=gpu
   --reservation=EECE5640 --gres=gpu:1 ./vecadd
   ```

7. If your program hangs for some reason, you can use the Slurm command `scancel`. Note that there are many options you can specify with `scancel`.

8. Make sure you exit out of the interactive session cleanly.