Compiling GPU code on Casper nodes

Below is an example of how to compile and run a GPU code interactively on Casper. You can use the CUDA C source code below to make your own **hello_world.cu** file.

Any libraries you build to support an application should be built with the same compiler, compiler version, and compatible flags that were used to compile the other parts of the application, including the main executable(s). Also, when you run the applications, be sure you have loaded the same module/version environment in which you created the applications. This will avoid job failures that can result from missing mpi launchers and library routines.

Log in to either Casper or Cheyenne and run **execcasper** with a GPU resource request to start an interactive job on a GPU-accelerated Casper node.

Example:

```
execcasper -l gpu_type=gp100 --ngpus=1
```

Load the CUDA module when your job starts.

```
module load cuda
```

Use the **nvcc** command to compile your code. (If your code is CUDA Fortran, first load the **nvhpc** module – the NVIDIA HPC collection – and use the compiler command **nvfortran** instead.)

```
nvcc -o hello hello_world.cu
```

Execute the compiled program as usual.

```
./hello
```

Output:

```
Contents of data before kernel call: HdjikhjcZ
Contents of data after kernel call:  Hello World!
```

Source code for hello_world.cu

```
/* hello_world.cu
 * -----------------------------------------------
 * A Hello World example in CUDA
 * -----------------------------------------------
 * This is a short program which uses multiple CUDA
 * threads to calculate a "Hello World" message which
 * is then printed to the screen.  It's intended to
 * demonstrate the execution of a CUDA kernel.
 * -----------------------------------------------
 */
#define SIZE 12
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>

/* CUDA kernel used to calculate hello world message */
__global__ void hello_world(char *a, int N);

int main(int argc, char **argv)
{
   /* data that will live on host */
   char *data;

   /* data that will live in device memory */
   char *d_data;

   /* allocate and initialize data array */
   data = (char*) malloc(SIZE*sizeof(char));
   data[0]  =  72; data[1]  = 100; data[2]  = 106;
   data[3]  = 105; data[4]  = 107; data[5]  =  27;
   data[6]  =  81; data[7]  = 104; data[8]  = 106;
   data[9]  =  99; data[10] =  90; data[11] =  22;

   /* print data before kernel call */
   printf("Contents of data before kernel call: %s\n", data);

   /* allocate memory on device */
   cudaMalloc(&d_data, SIZE*sizeof(char));

   /* copy memory to device array */
   cudaMemcpy(d_data, data, SIZE, cudaMemcpyHostToDevice);

   /* call kernel */
   hello_world<<<4,3>>>(d_data, SIZE);

   /* copy data back to host */
   cudaMemcpy(data, d_data, SIZE, cudaMemcpyDeviceToHost);

   /* print contents of array */
   printf("Contents of data after kernel call:  %s\n",data);

   /* clean up memory on host and device */
   cudaFree(d_data);
   free(data);
   return(0);
}

/* hello_world
 * Each thread increments an element of the input
 * array by its global thread id
 */
__global__ void hello_world(char *a, int N)
{
   int i = blockDim.x * blockIdx.x + threadIdx.x;
   if(i < N) a[i] = a[i] + i;
}
```