Assignment 6—Hybrid MPI+OpenMP+CUDA Programming
Due: November 16 (Mon), 2015

Write a triple-decker MPI+OpenMP+CUDA program to compute the value of π, by modifying the double-decker MPI+CUDA program, hypi_setdevice.cu, described in the lecture note on “Hybrid MPI+CUDA Programming”.

Your implementation should utilize two CPU cores and two GPU devices on each compute node. This is achieved by launching one MPI rank per node, where each rank spawns two OpenMP threads that run on different CPU cores and use different GPU devices as shown in the figure below.

You can employ spatial decomposition in the MPI+OpenMP layer as follows (for the CUDA layer, leave the interleaved assignment of quadrature points to CUDA threads in hypi_setdevice.cu as it is); see the figure below:

```c
#define NUM_DEVICE   2  // # of GPU devices = # of OpenMP threads
...
// In main()
MPI_Comm_rank(MPI_COMM_WORLD,&myid);  // My MPI rank
MPI_Comm_size(MPI_COMM_WORLD,&nproc);  // # of MPI processes
omp_set_num_threads(NUM_DEVICE);      // One OpenMP thread per GPU device
nbin = NBIN/(nproc*NUM_DEVICE);       // # of bins per OpenMP thread
step = 1.0/(float)(nbin*nproc*NUM_DEVICE);
...
#pragma omp parallel private(list the variables that need private copies)
{
...
  mpid = omp_get_thread_num();
  offset = (NUM_DEVICE*myid+mpid)*step*nbin; // Quadrature-point offset
cudaSetDevice(mpid%2);
...
}
```

Make sure to list all variables that need private copies in the private clause for the `omp parallel` directive.
The above OpenMP multithreading will introduce a race condition for variable \( \pi \). This can be circumvented by data privatization, i.e., by defining \( \text{float pid[NUM_DEVICE]} \) and using the array elements as dedicated accumulators for the OpenMP threads (or GPU devices).

To report which of the two GPUs have been used for the run, insert the following lines within the OpenMP parallel block:

```c
    cudaGetDevice(&dev_used);
    printf("myid = %d; mpid = %d; device used = %d; partial pi = %f\n", myid, mpid, dev_used, pid[mpid]);
```

where \( \text{int dev_used} \) is the ID of the GPU device (0 or 1) that was used, \( \text{myid} \) is the MPI rank, \( \text{mpid} \) is the OpenMP thread ID, \( \pi \) is a partial sum per OpenMP thread.

To compile your MPI+OpenMP+CUDA program, use the following sequence of commands:

```bash
    source /usr/usc/cuda/7.5/setup.csh
    source /usr/usc/openmpi/1.6.4/share/setup-cuda-gnu.csh
    mpicc -o pi3 -Xcompiler -fopenmp pi3.cu
```

**Assignment:**

1. *Submit your MPI+OpenMP+CUDA code.*

2. Run your code on 2 nodes, requesting 2 cores and 2 GPUs per node (i.e., `-l nodes=2:ppn=2:gpus=2` in your PBS script).

   ```bash
   myid = 0; mpid = 0: device used = 0; partial pi = 0.979926
   myid = 0; mpid = 1: device used = 1; partial pi = 0.874671
   myid = 1; mpid = 0: device used = 0; partial pi = 0.719409
   myid = 1; mpid = 1: device used = 1; partial pi = 0.567582
   PI = 3.141588
   ```