Assignment 6—Hybrid MPI+OpenMP+CUDA Programming
Due: October 25 (Mon), 2021

Part I: Pair-Distribution Computation with CUDA

In this part, you will write a CUDA program (name it `pdf.cu`) to compute a histogram `nhist` of atomic pair distances in molecular dynamics simulation:

```
for all histogram bins i
  nhist[i] = 0
for all atomic pairs (i,j)
  ++nhist[⌊|r_ij|/Δr⌋]
```

Here, $|\vec{r}_{ij}|$ is the distance between atomic pair (i, j) and Δr is the histogram bin size. The maximum atomic-pair distance with the periodic boundary condition is the diagonal of half the simulation box,

$$R_{\max} = \sqrt{\sum_{\alpha=x,y,z} \left(\frac{al[\alpha]}{2}\right)^2},$$

and with N_{hbin} bins for the histogram, the bin size is $\Delta r = R_{\max}/N_{\text{hbin}}$. Here, $al[\alpha]$ is the simulation box size in the α-th direction. With the minimal-image convention, however, the maximum distance, for which the histogram is meaningful, is half the simulation box length, $\min_{\alpha\in\{x,y,z\}}(al[\alpha]/2)$.

After computing the pair-distance histogram `nhist`, the pair distribution function (PDF) at distance $r_i = (i+1/2)\Delta r$ is defined as $g(r_i) = nhist(i)/2\pi r_i^2 \Delta r \rho N$, where ρ is the number density of atoms and N is the total number of atoms.

(Assignment)

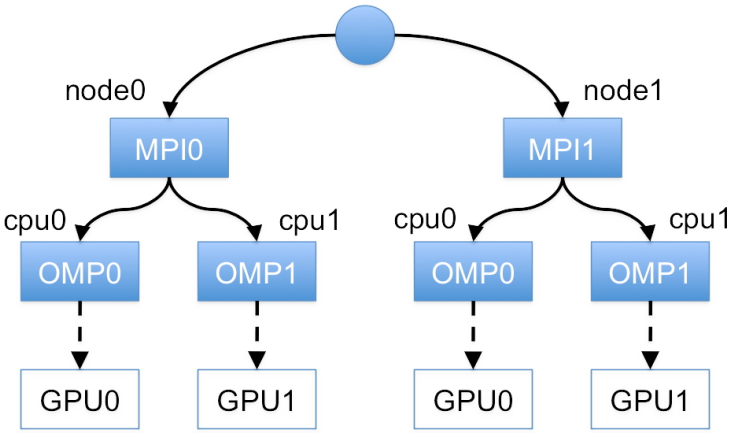
1. Modify the sequential PDF computation program `pdf0.c` to a CUDA program, following the lecture note on "Pair distribution computation on GPU". *Submit your code*.
2. Run your program by reading the atomic configuration `pos.d` (both `pdf0.c` and `pos.d` are provided in the assignment package). Plot the resulting pair distribution function, using $N_{\text{hbin}} = 2000$. *Submit your plot*.

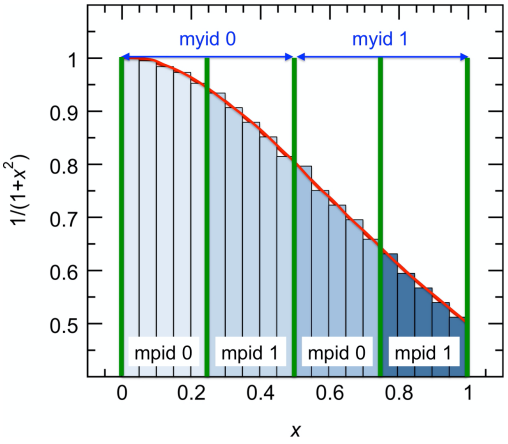
Part II: MPI+OpenMP+CUDA Computation of π

In this part, you will write a triple-decker MPI+OpenMP+CUDA program (name it `pi3.cu`) 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+OpenMP+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 left figure on the next page. 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 right figure on the next page.

```
#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 `float pid[NUM_DEVICE]` and using the array elements as dedicated accumulators for the OepnMP 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:

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

where `int dev_used` is the ID of the GPU device (0 or 1) that was used, `myid` is the MPI rank, `mpid` is the OpenMP thread ID, `pid[mpid]` is a partial sum per OpenMP thread.

(Assignment)

1. *Submit your MPI+OpenMP+CUDA code.*
2. Run your code on 2 nodes, requesting 2 cores and 2 GPUs per node.

   Submit your output, which should look like the following:

```
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
```