CSCI653 Assignment 5: MPI+OpenMP+CUDA Quantum Dynamics

Due: November 8 (Fri), 2019

The objective of this assignment is to gain hands-on experience of the standard programming model (hybrid MPI+OpenMP+CUDA) currently on GPU-accelerated clusters, using a real-world application. You will parallelize the one-dimensional quantum dynamics (QD) simulation program `qd1.c`, by combining MPI, OpenMP and CUDA (please see the lecture notes on parallelizing QD and running a hybrid MPI+OpenMP+CUDA program at HPC). Use spatial decomposition, so that process $p \in [0, P-1]$ (P is the number of processes or MPI ranks) is assigned a subsystem in the range $[pL_x, (p+1)L_x]$ of the total system length PL_x. Each subsystem of length L_x is discretized into N_x mesh points of interval $\Delta x = L_x/N_x$.

MPI Parallelization

In addition to properly setting up an MPI program (*e.g.*, MPI initialization—`MPI_Init()` and finalization—`MPI_Finalize()`, and finding out the rank—`MPI_Comm_rank()` and the total number of processes—`MPI_Comm_size()`, *etc*.), the following changes are required in `qd1.c` for the parallelization:

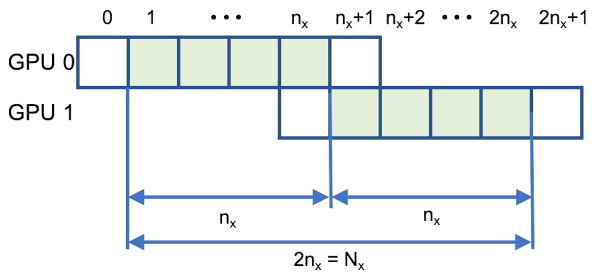
1. In functions `init_prop()` and `init_wavefn()`, the global position must be used to calculate the potential energy function and initialize the wave function, respectively. For process p, the global position of the i-th mesh point is $x = i\Delta x + pL_x$.
2. In `init_wavefn()`, the squared wave function values must be first summed over all the local mesh points for each process, and then `MPI_Allreduce()` must be used to calculate the global sum. Also use `MPI_Allreduce()` to calculate the global kinetic and potential energies, `ekin` and `epot`, in `calc_energy()`.
3. Rewrite `periodic_bc()` such that the wave function values at the subsystem boundaries, `psi[0][]` and `psi[NX+1][]`, of process p, are copied from the lower and upper neighbor processes, $p_{lw} = (p-1+P) \bmod P$ and $p_{up} = (p+1) \bmod P$, respectively. Use a 3-stage message-passing procedure—(i) message buffering to compose a message in array `dbuf[]`, (ii) asynchronous message receive (`MPI_Irecv()` of `dbufr[]`)/synchronous send (`MPI_Send()` of `dbuf[]`)/`MPI_Wait()`, and (iii) message storing by copying from `dbufr[]` to the destination array—as in the function `atom_copy()` in the parallel molecular dynamics program `pmd.c`.

Adding OpenMP and CUDA Parallelization

Each MPI process (with rank `myid`) spawns two OpenMP threads (each with a distinct OpenMP thread ID `mpid`) that handle the lower and higher halves of the mesh points, each using one GPU device specified by `cudaSetDevice(mpid%2)` (note that there are 2 GPUs per GPU-accelerated computing node at USC Center for High Performance Computing). We will only convert the most time-consuming functions for propagating the wave function in time (*i.e.*, `pot_prop` and `kin_prop`) into GPU kernel functions; let them be

```
__global__ void gpu_pot_prop() {...}
__global__ void gpu_kin_prop() {...}
```

The figure illustrates the assignment of mesh points to 2 OpenMP threads (or GPU devices) per MPI rank. The total number of mesh points per MPI rank is `NX`, while that per OpenMP thread is `nx = Nx/2`. Green-shaded mesh points `[1,nx]` are updated by GPU kernel functions, whereas white mesh points serve as augmented wave-function values to implement boundary conditions.



The following summarizes the required changes.

1. In `main()`, introduce an OpenMP parallel section (`#pragma omp parallel`) that only encloses the main simulation time-stepping `for` loop.
2. At the beginning of the parallel section, allocate necessary arrays in the global memory of the device: wave functions (`psi`, `wrk`) and temporal propagators (`u`, `al`, `blx`, `bux`). Then, copy their values from the host memory to the corresponding arrays in the device memory. We will use 1D deice arrays. For example, device (`dev_psi[2*i+s]`) and host (`psi[i][s]`) array elements stores the real (s = 0) and imaginary (s = 1) wave function values on the i-th mesh point.
3. In `pot_prop()` and `kin_prop()`, (i) copy the current values of the wave function, `psi[0:nx+1][]`, from the host to the corresponding array in the device (note the augmented two elements for imposing boundary conditions), (2) perform potential and kinetic propagations on GPU by invoking the kernel functions, `gpu_pot_prop()` and `gpu_kin_prop()`, respectively, and (iii) copy back the updated wave function `psi[1:nx][]` from device to host.
4. Functions involving MPI interprocess communications, `calc_energy()` in `main()` and `periodic_bc()` in `kin_prop()`, will be executed by a single thread like:

```
#pragma omp master
periodic_bc();    // Only master thread ID 0 executes, while letting the others wait
#pragma omp barrier
```

Assignment

1. Submit the source code of your MPI+OpenMP+CUDA program.
2. Run the resulting parallel program on 2 computing nodes with 2 processors and 2 GPUs per node, using Slurm options: `--nodes=2, --ntasks-per-node=1, --cpus-per-task=2, --gres=gpu:2`. In the header file, define the number of mesh points, `NX` = 4992, per process. For the run, use the following input parameters in file `qd1.in`: `LX` = 500.0, `DT` = 4.0e-3, `NSTEP` = 12500, `NECAL` = 100; `X0` = 250.0, `S0` = 60.0, `E0` = 100.0, `BH` = 5.0, `BW` = 20.0, `EH` = 50.0. The meaning of these input parameters is explained in the header file, `qd1.h`, for program `qd1.c`.
3. Submit the plot of the kinetic, potential and total energies (second, third and fourth columns of the standard output) as a function of the simulated time (first column of the standard output).