Prof. Jingke Li (lij@pdx.edu); Classes/Labs: TR 1400-1550 @FAB 150, 88-10; Office Hours: TR 1300-1400 @FAB 120-06

Lab 7: Programming with MPI

1 Simple Send-Receive

1. Read and understand the provided program `simple.c`. It implements a pair of send-receive actions between two processes. The sender sends an integer to the receiver; the receiver decreases the value by one and sends it back. Even though the message-passing happens only between two processes, the program can run with any number of processes. It can also take a command-line argument, an integer to be used as the message value. Compile and run it. Here are some examples:

```
linux> mpicc -o simple simple.c     // compile
linux> mpirun -n 2 simple           // run with 2 procs
linux> mpirun -n 4 simple           // run with 4 procs
linux> mpirun -n 4 simple 100       // run with 4 procs, msg=100
```

2. Write a program `ring.c` based on the above program. Instead of send-receive actions between two processes, the new program will involve *all* active processes. In the program, process 0 (*i.e.* process with `rank==0`) sends an integer to process 1; upon receiving the integer, process 1 decreases its value by 1, and sends the new number to process 2; process 2 does the same, and sends a new number to process 3; and so on. The last process in the active set sends its modified number back to process 0. Like in `simple.c`, each process should make the sending and receiving actions visible by printing out a message showing its rank, its host name, and the involved integer's value. Note that the total number of active processes is not controlled by the MPI program itself. Compile your program with `mpicc`, and test it with multiple combinations of runtime parameters.

2 Collective Communication

1. The file `vecsum.c` contains the vector sum program shown in class. It provides examples of various collective routines. Read and understand the program, then compile and run it.

2. The file `scan.c` contains a simple example of the prefix scan routine with the sum operation. Each process supplies its rank number as the source data and receives the scan result in a variable, `psum`. Compile and run this program, you should see each process prints out its `psum` value. For process `i`, the value is $\sum_0^i k$, i.e. the partial sum of all the rank numbers up to `i`.

   Now, add a new gather routine, to collect all the `psum` values back to process 0. Have process 0 print out the array of received values from the gather routine, which should match those `psum` values.

3. The file `scatter.c` contain an example of the scatter routine. Process 0 scatters an array `cnt` of values to the other processes, value `cnt[i]` to process `i`. Compile and run this program.

   Now, we want to add another scatter routine. This time we want to scatter a *different* number of items to each process. Follow the instructions below to change the program:

   (a) Add declarations:

```
int *data, *disp, rbuf[2 * size];
if (rank == 0) {
  data = (int *) malloc(sizeof(int) * 128);   // source data
  disp = (int *) malloc(sizeof(int) * size);  // displacement array
}
```

(b) Have process 0 initialize the `data` array so that `data[k] = k` for all elements. This is the source array. We want to scatter a section of this array to each process; more specifically, we want to scatter `cnt[i]` number of `data` elements to process `i`.

(c) Have process 0 initialize the `disp` array so that each entry holds a index value to the `data` array, indicating the start of a section. As an example, since `cnt[0] = 0; cnt[1] = 2; cnt[2] = 4;` (i.e. the first three section sizes are 0, 2, 4), we should have

```
disp[0] = 0; disp[1] = 0; disp[2] = 2; disp[3] = 6;
```

(d) Add the following variable-sized scatter call to the program:

```
// scatter variable-sized data sections to all processes
MPI_Scatterv(data, cnt, disp, MPI_INT, rbuf, 2*size, MPI_INT, 0, MPI_COMM_WORLD);
```

Read MPI document to understand the parameters.

(e) Add print statements after the call, so you'll see messages as follows:

```
P[0] got 0 items:
P[1] got 2 items: 0,1,
P[2] got 4 items: 2,3,4,5,
P[3] got 6 items: 6,7,8,9,10,11,
```

Compile and run the new program.

3 Submission

As usual, write a short report summarizing your work. Submit it with your `ring.c` and the two modified programs, `scan.c` and `scatter.c`, through the "Lab 7" folder on Canvas. The submission deadline is the end of tomorrow (Friday).