CS402 Lab Session: Using Advanced MPI Functions

1. Programming with derived datatype

In the lecture, we learned that the following segment of code can construct a derived datatype shown in figure 1.

```
MPI_Datatype floattype;
MPI_TYPE_vector (3, 2, 4, MPI_FLOAT, &floattype);
MPI_Type_commit (&floattype);
MPI_Send (data, 1, floattype, dest, tag, MPI_COMM_WORLD);
 MPI_Type_free (&floattype);
```

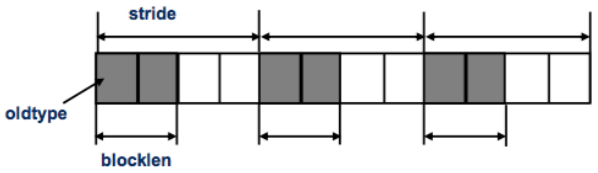

Figure 1. the memory layout of the data to be sent

Now let's use this MPI function to send some data. In C, the matrix is stored in the row-major order. Assume there is a 20*10 matrix of MPI_DOUBLE numbers (matrix *a*). How does a process send the elements in the order of columns to another process? Namely, process 0 sends all elements in the first column to process 1, and also sends the 2nd column, 3rd, ..., and 10th columns in order to process 1.

Hint: a column has 20 blocks of one element, spaced 10 elements apart. Namely, we can construct such a derived datatype using the followings codes:


```
MPI_Datatype MPI_column;
MPI_Type_vector(   /* count= */ 20, /* blocklength= */ 1, /* stride= */ 10, MPI_DOUBLE,
&MPI_column );
```

Then use the following statement to send the first column:

```
MPI_Send( a, 1,MPI_column, ... );
```

To send the second column, you need to adjust the starting address of the first block, which is the address of *a*[0][1].

```
MPI_Send( &(a[0][1]), 1,MPI_column, ... );
```

It is similar for sending other columns. So you need to write a loop to send the columns, in each iteration of which, one column is sent.

Task 1: Suppose there is a matrix *a* of size 48*48 and we need to send the elements *a*[4*i][4*i], i=0, …, 11. How can we send these elements with a single transfer?

Task 2. Following task 1, we now need to send the elements of a[i][j] (i, j=0, …, 11). Namely, the elements :

a[0][0], a[0][4], a[0][8], …, a[0][40], a[0][44],

a[1][0], a[1][4], a[1][8], …, a[1][40], a[1][44],

…

a[44][0], a[44][4], a[44][8], …, a[44][40], a[44][44]

How can we send all these elements with a single MPI_Send? (hint: the data type used to construct the derived datatype can be primitive data type, but can also be a derived datatype itself;)

if it were not for the derived datatype function, think how we can send the elements in Tasks 1 and 2. By comparison, hope you understand how the derived datatype provide the convenience and the performance improvement for sending non-contiguous data.

2. MPI blocking and non-blocking communications

The pingpong program is provided in the assignment 2 package to send and receive the messages between 2 processes.

MPI_Send and MPI_Recv, which are blocking send and receive functions, are used in the pingpong program to send and receive increasingly larger messages.

Run the pingpong program with 2 processes. Make sure the two processes are located in 2 different computing nodes.

The pingpong program outputs the total elapsed time for sending and receiving each message size for *iters* times.

Task 3: Change the program so that it outputs the average time for sending and receiving each message size. Use the excel sheet to plot the average times for all message sizes. Observe what shape these data points (average times) form. Can you obtain from the plotted figure t_s (the message startup time/overhead) and t_w (the time for sending one byte of data) taught in the lecture?

Task 4: If we keep increasing the size of the message being sent (by changing the value of the variable *maxsize* in the pingpong program), what message size will cause the average time for sending and receiving such size of message to increase “out of proportion”, compared with other increases in message size. What can such message size tell you, based on the knowledge you learned in the lecture? (Hint: you learned system buffer in MPI)

Task 5: Change blocking send and receive to non-blocking send and receive in the pingpong program. Compare the output time with the output of the program using blocking send and receive.