7. Parallel programming with MPI - II

[1]

[1] M. Ganesh, MATH440/540, SPRING 2018

[2] More MPI details and commands

- **For stopping a program (emergency) with MPI commands, use** `MPI_Abort`
- **To abort with a failure code, say,** `fail_code`**, use in Fortran, C, and C++ programming with MPI respectively:**

```
CALL MPI_Abort(MPI_COMM_WORLD, fail_code, ierror)
MPI_Abort(MPI_COMM_WORLD, fail_code);
MPI::COMM_WORLD.Abort(fail_code);
```

- **The state of MPI on a processing core can be tested using the following commands with** `started` **and** `stopped`**. In Fortran these two variables are logical and in C these are of the type** `int`

```
CALL MPI_Initialized ( started , ierror )
CALL MPI_Finalized ( stopped , ierror )
ierror = MPI_Initialized ( & started );
ierror = MPI_Finalized ( & stopped ) ;
ierror = MPI::Is_initialized ( );
ierror = MPI::Is_finalized ( ) ;
```

[2] M. Ganesh, MATH440/540, SPRING 2018

- **[3]To synchronize/block so that all processing cores in a communicator wait until all cores have entered the call (and start up independently), use the `MPI_Barrier` command, as below in Fortran, C, C++:**

```
CALL MPI_Barrier(MPI_COMM_WORLD, ierror)
MPI_Barrier(MPI_COMM_WORLD);
MPI::COMM_WORLD.Barrier( );
```

- **To broadcast copies of the *same data*, say, `buffer` with datatype, say, `DOUBLE_PRECISION` or `DOUBLE`, and total number of elements, say, `count`, to all processing cores in a communicator, say `MPI_COMM_WORLD` from, say, `master` use `MPI_Bcast` command, as below in Fortran, C, C++:**

```
CALL MPI_Bcast(buffer, count, MPI_DOUBLE_PRECISION, master, &
MPI_COMM_WORLD, ierror)
MPI_Bcast(buffer, count, MPI_DOUBLE, master, MPI_COMM_WORLD)
MPI::COMM_WORLD.Bcast(buffer, count, MPI::DOUBLE, master)
```

[3]M. Ganesh, MATH440/540, SPRING 2018

Exercise 1:

Create a file with some data containing, say, $N = 12$ numbers. Write a parallel (Fortran, C, or C++) program with MPI so that the `master` core reads the data from the file in, say, `my_array` and broadcasts `my_array` to all processing cores in a communicator. Your program should print `my_array` from each processing core and the associated core number.

Exercise 2:

Take a copy of your Exercise 1 code. Modify the copy as follows: `my_array` should be read from the file instead by the third processing core in a communicator. (If the communicator contains only two cores, the program should print an error message and abort.) The third processing core should broadcast only the middle six elements of `my_array`.

Your modified program should print the six elements of `my_array` from each processing core and the associated core number.

- [4]**To scatter copies of different data from, say,** `master` **to all processing cores in a communicator, by splitting a bigger data evenly between all processing cores in a communicator, use** `MPI_SCATTER`**:**
- **Let the total size of communicator be given by** `num_cores` **(That is, using previous notation,** `my_rank = 0, 1, ..., num_cores-1`**.)**
- **Suppose that we want** `master` **to send equal parts of a data, say,** `buffer` **with data type** `DOUBLE_PRECISION` **or** `DOUBLE` **and total number of elements** `count = num_cores * send_count`
- **There are** `num_cores` **equal sized blocks of data in** `buffer` **with each block consisting of** `send_count` **number of elements**
- **Suppose that we want the** `master` **to scatter** `buffer` **so that, for** `i = 1, ..., num_cores`**, the** i**-th processing core gets the** i**-th block of** `send_count` **number of elements from** `buffer`**, as a variable** `recv_buffer` **with its count** `recv_count`
- **The send process is used only on** `master` **and the receive process is used on all processing cores**
- **To achieve the above task, use the following (F, C, C++) commands:**

[4]M. Ganesh, MATH440/540, SPRING 2018

[5]
```
CALL MPI_Scatter( &
buffer, send_count, MPI_DOUBLE_PRECISION, &
recv_buffer, recv_count, MPI_DOUBLE_PRECISION, &
master, MPI_COMM_WORLD, ierror)
```

```
MPI_Scatter( buffer, send_count, MPI_DOUBLE, recv_buffer, recv_count,
MPI_DOUBLE, master, MPI_COMM_WORLD)
```

```
MPI::COMM_WORLD.Scatter( buffer, send_count, MPI::DOUBLE, recv_buffer,
recv_count, MPI::DOUBLE, master)
```

- **Note that `buffer` and `recv_buffer` may have different ranks. For example, `buffer` may be a matrix and `recv_buffer` may be a vector**

[5] M. Ganesh, MATH440/540, SPRING 2018

Exercise 3:

Create a file with some data containing, say, $N = 12$ numbers. Write a parallel (Fortran, C, or C++) program with MPI so that the `master` core reads the data from the file in a 3×4 matrix, say, `my_mat`.

Let `num_cores` be the total number of processing core in a communicator, with `master = 0` being the first processing core in the communicator.

Let `send_count = N/4`. The `master` core should scatter `my_mat` so that: the first processing core receives the first column of `my_mat`; the second processing core receives the second column of `my_mat` and so on. The processing cores should receive these elements in a new array `recv_array`.

Your program should check if `send_count*num_cores == N`.
If this condition is violated, the program should abort.
Your program should be such that the `master` prints `my_mat` and its rank and each processing core prints its rank and `recv_array`.

- [6]`MPI_Gather` **command is the converse of the operation** `MPI_Scatter`

- **For** `MPI_Gather`**, the** `recv_buffer` **is larger with size** `recv_count*num_cores`

- **In Fortran, C, and C++ programming with MPI, respectively, use:**

```
CALL MPI_Gather( &
send_buffer, send_count, MPI_DOUBLE_PRECISION, &
recv_buffer, recv_count, MPI_DOUBLE_PRECISION, &
master, MPI_COMM_WORLD, ierror)
```

```
MPI_Gather( send_buffer, send_count, MPI_DOUBLE, recv_buffer, recv_count,
MPI_DOUBLE, master, MPI_COMM_WORLD)
```

```
MPI::COMM_WORLD.Gather( send_buffer, send_count, MPI::DOUBLE, recv_buffer,
recv_count, MPI::DOUBLE, master)
```

[6]M. Ganesh, MATH440/540, SPRING 2018

- **For example, to implement Exercise 4 in Topic 7, we may avoid `MPI_Send` ad `MPI_Recv` and instead use**

```
IF (my_rank ==  master) THEN
    ALLOCATE(all_sum(0:num_cores-1))
END IF

call MPI_Gather(local_sum, 1,  MPI_DOUBLE_PRECISION, all_sum, 1, &
MPI_DOUBLE_PRECISION, master, MPI_COMM_WORLD,ierror)

full_sum = sum(all_sum)
```

Exercise 4:
Generalize your program in Exercise 3 (current section) so that the each processing core computes the square root of each element in `recv_array` and the `master` gathers all these results in a new 3×4 matrix, say, `my_mat_comp_sqrt` Your program should be such that the `master` prints `my_mat`, and `my_mat_comp_sqrt` and each processing core prints its rank and `recv_array`.

- [7]**We may gather the data on `master` and then broadcast it to all processing cores**
- **Instead this can be achieved using a single command `MPI_Allgather`**
- **In this symmetric operation, the command is similar to `MPI_Gather` except that we need to remove the `master` argument**

```
CALL MPI_Allgather( &
send_buffer, send_count, MPI_DOUBLE_PRECISION, &
recv_buffer, recv_count, MPI_DOUBLE_PRECISION, &
MPI_COMM_WORLD, ierror)
```

```
MPI_Allgather(send_buffer, send_count, MPI_DOUBLE, recv_buffer, recv_count,
MPI_DOUBLE, MPI_COMM_WORLD)
```

```
MPI::COMM_WORLD.Allgather(send_buffer, send_count, MPI::DOUBLE, recv_buffer
recv_count, MPI::DOUBLE)
```

[7]M. Ganesh, MATH440/540, SPRING 2018

[8]**Exercise 5:**

Generalize your program in Exercise 3 so that the each processing core computes the square root of each element in `recv_array` and each processing core receives the results in a 3×4 matrix, say, `my_mat_comp_sqrt`. Your program should be such that the `master` prints `my_mat`, and `my_mat_comp_sqrt` and each processing core prints its rank and `my_mat_comp_sqrt`.

[8]M. Ganesh, MATH440/540, SPRING 2018

- [9]**We can achieve a combination of the scatter and gather operation using** `MPI_Alltoall`
- **Use this if you want all processing cores to act like the** `master` **to scatter and gather data**
- **Such a powerful operation is useful for achieving, for example, a parallel transpose of a matrix:**

```
CALL MPI_Alltoall( &
send_buffer, send_count, MPI_DOUBLE_PRECISION, &
recv_buffer, recv_count, MPI_DOUBLE_PRECISION, &
MPI_COMM_WORLD, ierror)
```

```
MPI_Alltoall(send_buffer, send_count, MPI_DOUBLE, recv_buffer, recv_count,
MPI_DOUBLE, MPI_COMM_WORLD)
```

```
MPI::COMM_WORLD.Alltoall(send_buffer, send_count, MPI::DOUBLE, recv_buffer,
recv_count, MPI::DOUBLE)
```

[9]M. Ganesh, MATH440/540, SPRING 2018

[10]**Exercise 6:**

Let P be the size (total number of cores) in a communicator.
Let A_P be a $P \times P$ matrix defined by

$$A_P = \begin{bmatrix} 1 & 2 & \cdots & P \\ P+1 & P+2 & \cdots & 2P \\ 2P+1 & 2P+2 & \cdots & 3P \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ (P-1)P+1 & (P-1)P+2 & \cdots & P^2 \end{bmatrix}. \tag{7.1}$$

Write a parallel (Fortran, or C, or C++) program with MPI using P cores (determined by `MPI_COMM_SIZE`) so that for $i = 1, \cdots, P$:

- **The i-th processing core allocates a P dimensional vector `arow` with elements being the i-th row of A_P**
- **Using the `arow` vectors and `MPI_Alltoall` command, the i-th processing core also gets the i-th row of A_P^T**
- **The `master` prints the matrix A_P in a file `A_mat.txt` and the matrix A_P^T in a file `transp_A_mat.txt`. (Check these two files to validate code.)**

[10]M. Ganesh, MATH440/540, SPRING 2018

- [11]**If the purpose of the `MPI_Gather` command in your MPI code is to achieve some reduction (such as summation, find maximum of values, or ...) of the gathered data using `master`, the reduction can be achieved using `MPI_Reduce` operations, without the need to use `MPI_Gather`:**
- **Unlike in `MPI_Scatter` and `MPI_Gather` commands, for reduce operations, it is sufficient to specify the datatype and count of the `buffer` only once**
- **It is important to note that the reduction process does not reduce over the vector (that is if the `count` of the `buffer` is more than one)**
- **If `count` is more than one, then the reduction operation is applied for each index in `buffer` separately**
- **In case of requiring reduction over a vector, you need to get this done by yourself first**
- **Some of the MPI operations are:**
  `MPI_SUM, MPI_MIN, MPI_MAX, MPI_PROD, MPI_MAXLOC, MPI_MINLOC, ....`
- **Below, we use `oper` for any one of these reduction operations**

[11]M. Ganesh, MATH440/540, SPRING 2018

[12]
```
CALL MPI_Reduce( &
send_buffer, recv_buffer, count, MPI_DOUBLE_PRECISION, &
oper, master, MPI_COMM_WORLD, ierror)
```

```
MPI_Reduce(send_buffer, recv_buffer, count, MPI_DOUBLE, oper, master,
MPI_COMM_WORLD)
```

```
MPI::COMM_WORLD.Reduce(send_buffer, recv_buffer, count, MPI::DOUBLE, oper,
master)
```

- **For example, to implement Exercise 4 in Topic 7, we may avoid `MPI_Send` and `MPI_Recv`, or `MPI_Gather` and instead use just one main MPI command:**

```
CALL MPI_Reduce(local_sum, full_sum,  1, MPI_DOUBLE_PRECISION, &
          MPI_SUM, master,  MPI_COMM_WORLD, ierror)
```

[12] M. Ganesh, MATH440/540, SPRING 2018

- [13] **Exercise 7:**

  Generalize your program in Exercise 3 (current section) so that the `master` deallocates `my_mat` and then receives the maximum element in each column of `my_mat` from all processing cores. Then the `master` computes the maximum value in the destroyed `my_mat`.

  Your program should be such that the `master` prints `my_mat` and the maximum element in `my_mat`.

- **We may be interested in all processing cores (instead of just the `master`) to have the reduced result**
- **We can achieve this using a single command `MPI_Allreduce`**
- **In this symmetric operation, the command is similar to `MPI_Reduce` except that we need to remove the `master` argument**

[13] M. Ganesh, MATH440/540, SPRING 2018

[14]
```
CALL MPI_Allreduce( &
send_buffer, recv_buffer, count, MPI_DOUBLE_PRECISION, &
oper, MPI_COMM_WORLD, ierror)
```

```
MPI_Allreduce(send_buffer, recv_buffer, count, MPI_DOUBLE, oper,
MPI_COMM_WORLD)
```

```
MPI::COMM_WORLD.Allreduce(send_buffer, recv_buffer, count, MPI::DOUBLE,
oper)
```

Exercise 8:

Generalize your program in Exercise 3 (current section) so that the `master` deallocates `my_mat`. Then all processing cores compute the maximum value in the destroyed `my_mat`.

Your program should be such that all processing core should print its rank and the maximum element in `my_mat`.

[14] M. Ganesh, MATH440/540, SPRING 2018