Problem description

1. 在使用 MPI_Isend()的部分若沒有用 MPI_Wait()的話，可能導致印出回傳結果的時候，印出來的是 rank 編號 0 要傳出去的內容而不是回傳回來的確認訊息。
2. 在使用 MPI_Gather()的時候，在 slave nodes 要先用一次 MPI_Gather()，然後在 master_node 再使用一次 MPI_Gather()，才能正確印出回傳訊息。

Code and explanations

Using “MPI_Send()”

```
# include "mpi.h"
# include <stdio.h>

int main(int argc, char *argv[])
{
        int rank, size;
        char message[100];

        MPI_Init(&argc, &argv);
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);

        if (size < 2){
                printf("Please run with at least two processes.\n");
                fflush(stdout);
                MPI_Finalize();

                return 0;
        }

        if (rank == 0){
                for (int i = 1; i < size; i++){
                        sprintf(message, "Hi rank %d, I'm 余明昌 from Parallel Programming Design Course in 2023", i);
                        MPI_Send(message, 100, MPI_CHAR, i, 0, MPI_COMM_WORLD);
                }
                for (int i = 1; i < size; i++){
                        MPI_Recv(message, 100, MPI_CHAR, i, MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
                        printf("Rank 0 got message from rank %d: %s\n", i, message);
                }
        } else{
                MPI_Recv(message, 100, MPI_CHAR, 0, MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

                printf("Rank %d got message from rank 0: %s\n", rank, message);
                fflush(stdout);

                sprintf(message, "Rank %d recieved. Thank you.", rank);
                MPI_Send(message, 100, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
        }

        MPI_Finalize();

        return 0;
}
```

首先初始化 MPI，將 rank 的數量代入 size 變數、rank 的編號代入 rank 變數。確認 rank 的數量是否小於 2，若小於 2 則印出錯誤訊息。

接著將處理過程分為 rank 編號為 0 以及其他 rank 編號的處理方法。若 rank 編號為 0，以 for 迴圈將訊息存進 message 變數分別傳送給各個 rank，接著接收各個 rank 回傳的確認接收訊息；若 rank 編號不是 0 則先接收 rank 編號 0 傳送來的訊息，印出後回傳確認接收訊息給 rank 編號 0。

Using “MPI_Isend()”

```c
# include "mpi.h"
# include <stdio.h>

int main(int argc, char *argv[])
{
        int rank, size;
        char message[100];
        MPI_Status status;
        MPI_Request request1, request2;

        MPI_Init(&argc, &argv);
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);

        if (size < 2){
                printf("Please run with at least two processes.\n");
                fflush(stdout);
                MPI_Finalize();

                return 0;
        }

        if (rank == 0){
                for (int i = 1; i < size; i++){
                        sprintf(message, "Hi rank %d, I'm 余明昌 from Parallel Programming Design Course in 2023", i);
                        MPI_Isend(message, 100, MPI_CHAR, i, 0, MPI_COMM_WORLD, &request1);
                        MPI_Wait(&request1, &status);
                }

                for (int i = 1; i < size; i++){
                        MPI_Irecv(message, 100, MPI_CHAR, i, 1, MPI_COMM_WORLD, &request2);
                        MPI_Wait(&request2, &status);
                        printf("Rank 0 got message from rank %d: %s\n", i, message);
                }
        } else{
                MPI_Irecv(message, 100, MPI_CHAR, 0, MPI_ANY_TAG, MPI_COMM_WORLD, &request1);
                MPI_Wait(&request1, &status);

                printf("Rank %d got message from rank 0: %s\n", rank, message);
                fflush(stdout);

                sprintf(message, "Rank %d recieved. Thank you.", rank);
                MPI_Isend(message, 100, MPI_CHAR, 0, 1, MPI_COMM_WORLD, &request2);
                MPI_Wait(&request2, &status);
        }

        MPI_Finalize();

        return 0;
}
```

首先初始化 MPI，將 rank 的數量代入 size 變數、rank 的編號代入 rank 變數。確認 rank 的數量是否小於 2，若小於 2 則印出錯誤訊息。
接著將處理過程分為 rank 編號為 0 以及其他 rank 編號的處理方法。若 rank 編號為 0，以 for 迴圈將訊息存進 message 變數分別傳送給各個 rank，接著接收各個 rank 回傳的確認接收訊息；若 rank 編號不是 0 則先接收 rank 編號 0 傳送來的訊息，印出後回傳確認接收訊息給 rank 編號 0。
與 MPI_Send()的差別為每次傳送訊息或接收訊息要先等待工作完成，否則可能印出自己要傳送給別人的訊息。

Using “MPI_Scatter()” + “MPI_Gather()”

```c
# include "mpi.h"
# include <stdio.h>
# include <stdlib.h>

int main(int argc, char *argv[])
{
        int rank, size;

        MPI_Init(&argc, &argv);
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        char send_message[size][100];
        char* receive_message = (char*)malloc(100 * sizeof(char));

        if (size < 2){
                printf("Please run with at least two processes.\n");
                fflush(stdout);
                MPI_Finalize();

                return 0;
        }

        if (rank == 0){
                for (int i = 0; i < size; i++){
                        sprintf(send_message[i], "Hi rank %d, I'm 余明昌 from Parallel Programming Design Course in 2023", i);
                }
        }
        MPI_Scatter(send_message, 100, MPI_CHAR, receive_message, 100, MPI_CHAR, 0, MPI_COMM_WORLD);

        if (rank == 0){
                MPI_Gather(receive_message, 100, MPI_CHAR, send_message, 100, MPI_CHAR, 0, MPI_COMM_WORLD);
                for (int i = 1; i < size; i++){
                        printf("Rank 0 got message from rank %d: %s\n", i, send_message[i]);
                }
        } else{
                printf("Rank %d got message from rank 0: %s\n", rank, receive_message);
                fflush(stdout);

                sprintf(receive_message, "Rank %d received. Thank you.", rank);
                MPI_Gather(receive_message, 100, MPI_CHAR, send_message, 100, MPI_CHAR, 0, MPI_COMM_WORLD);
        }

        MPI_Finalize();

        return 0;
}
```

首先初始化 MPI，將 rank 的數量代入 size 變數、rank 的編號代入 rank 變數。確認 rank 的數量是否小於 2，若小於 2 則印出錯誤訊息。
接著在 rank 編號 0 將要傳送的訊息存在 send_message 變數，存好之後以 MPI_Scatter()傳送給各個 rank(包括 rank 編號 0)。
接著將處理過程分為 rank 編號為 0 以及其他 rank 編號的處理方法。若 rank 編號不是 0 則先印出從 rank 編號 0 接收到的訊息，接著將確認收到訊息以 MPI_Gather()回傳到 rank 編號 0；若 rank 編號為 0，先以 MPI_Gather()將各個 rank 回傳的訊息存到 send_message 變數，接著印出。

Sampled outputs

Using “MPI_Send()”

```
(base) ming@ming-MacBook-Pro HW1 % mpicc PPD_PA1_B0928007_1.c
```

產生 a.out

```
(base) ming@ming-MacBook-Pro HW1 % mpiexec -n 4 ./a.out
Rank 2 got message from rank 0: Hi rank 2, I'm 余明昌 from Parallel Programming Design Course in 2023
Rank 3 got message from rank 0: Hi rank 3, I'm 余明昌 from Parallel Programming Design Course in 2023
Rank 1 got message from rank 0: Hi rank 1, I'm 余明昌 from Parallel Programming Design Course in 2023
Rank 0 got message from rank 1: Rank 1 recieved. Thank you.
Rank 0 got message from rank 2: Rank 2 recieved. Thank you.
Rank 0 got message from rank 3: Rank 3 recieved. Thank you.
```

指定 rank 為 4, rank 0 是 master node, rank 1~3 是 slave nodes.

Using “MPI_Isend()”

```
(base) ming@ming-MacBook-Pro HW1 % mpicc PPD_PA1_B0928007_2.c
```

產生 a.out

```
(base) ming@ming-MacBook-Pro HW1 % mpiexec -n 4 ./a.out
Rank 3 got message from rank 0: Hi rank 3, I'm 余明昌 from Parallel Programming Design Course in 2023
Rank 0 got message from rank 1: Rank 1 recieved. Thank you.
Rank 0 got message from rank 2: Rank 2 recieved. Thank you.
Rank 0 got message from rank 3: Rank 3 recieved. Thank you.
Rank 1 got message from rank 0: Hi rank 1, I'm 余明昌 from Parallel Programming Design Course in 2023
Rank 2 got message from rank 0: Hi rank 2, I'm 余明昌 from Parallel Programming Design Course in 2023
```

指定 rank 為 4, rank 0 是 master node, rank 1~3 是 slave nodes.

Using ”MPI_Scatter()” + “MPI_Gather()“

```
(base) ming@ming-MacBook-Pro HW1 % mpicc PPD_PA1_B0928007_3.c
```

產生 a.out

```
(base) ming@ming-MacBook-Pro HW1 % mpiexec -n 4 ./a.out
Rank 2 got message from rank 0: Hi rank 2, I'm 余明昌 from Parallel Programming Design Course in 2023
Rank 3 got message from rank 0: Hi rank 3, I'm 余明昌 from Parallel Programming Design Course in 2023
Rank 1 got message from rank 0: Hi rank 1, I'm 余明昌 from Parallel Programming Design Course in 2023
Rank 0 got message from rank 1: Rank 1 received. Thank you.
Rank 0 got message from rank 2: Rank 2 received. Thank you.
Rank 0 got message from rank 3: Rank 3 received. Thank you.
```

指定 rank 為 4, rank 0 是 master node, rank 1~3 是 slave nodes.

Discussions or what I have learned

MPI_Send + MPI_Recv()與 MPI_Isend() + MPI_Irecv()使用上的差異在於在使用 MPI_Isend() + MPI_Irecv()時，必須在傳送以及接收後方皆使用 MPI_Wait()，這樣才不會導致印出結果有誤。使用 MPI_Scatter() + MPI_Gather()時，MPI_Scatter 負責將 master node 的訊息拆分給各個 slave nodes，而 MPI_Gather()則負責將各個 rank 的訊息整合合併到 master node。