MPI 2

1)**MPI_Gather() & MPI_Scatter()**

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <mpi.h>
#include <assert.h>

// Creates an array of random numbers. Each number has a value from 0
- 1
float *create_rand_nums(int num_elements) {
  float *rand_nums = (float *)malloc(sizeof(float) * num_elements);
  assert(rand_nums != NULL);
  int i;
  for (i = 0; i < num_elements; i++) {
     rand_nums[i] = (rand() / (float)RAND_MAX);
  }
  return rand_nums;
}

// Computes the average of an array of numbers
float compute_avg(float *array, int num_elements) {
  float sum = 0.f;
  int i;
  for (i = 0; i < num_elements; i++) {
     sum += array[i];
  }
  return sum / num_elements;
}

int main(int argc, char** argv) {
  if (argc != 2) {
     fprintf(stderr, "Usage: avg num_elements_per_proc\n");
     exit(1);
  }

  int num_elements_per_proc = atoi(argv[1]);
  // Seed the random number generator to get different results each
time
  srand(time(NULL));
```

```
  MPI_Init(NULL, NULL);

  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  // Create a random array of elements on the root process. Its total
  // size will be the number of elements per process times the number
  // of processes
  float *rand_nums = NULL;
  if (world_rank == 0) {
     rand_nums = create_rand_nums(num_elements_per_proc *
world_size);
  }

  // For each process, create a buffer that will hold a subset of the
entire
  // array
  float *sub_rand_nums = (float *)malloc(sizeof(float) *
num_elements_per_proc);
  assert(sub_rand_nums != NULL);

  // Scatter the random numbers from the root process to all
processes in
  // the MPI world
  MPI_Scatter(rand_nums, num_elements_per_proc, MPI_FLOAT,
sub_rand_nums,
           num_elements_per_proc, MPI_FLOAT, 0, MPI_COMM_WORLD);

  // Compute the average of your subset
  float sub_avg = compute_avg(sub_rand_nums, num_elements_per_proc);

  // Gather all partial averages down to the root process
  float *sub_avgs = NULL;
  if (world_rank == 0) {
     sub_avgs = (float *)malloc(sizeof(float) * world_size);
     assert(sub_avgs != NULL);
  }
  MPI_Gather(&sub_avg, 1, MPI_FLOAT, sub_avgs, 1, MPI_FLOAT, 0,
MPI_COMM_WORLD);
```

```
  // Now that we have all of the partial averages on the root,
compute the
  // total average of all numbers. Since we are assuming each process
computed
  // an average across an equal amount of elements, this computation
will
  // produce the correct answer.
  if (world_rank == 0) {
      float avg = compute_avg(sub_avgs, world_size);
      printf("Avg of all elements is %f\n", avg);
      // Compute the average across the original data for comparison
      float original_data_avg =
      compute_avg(rand_nums, num_elements_per_proc * world_size);
      printf("Avg computed across original data is %f\n",
original_data_avg);
  }

  // Clean up
  if (world_rank == 0) {
      free(rand_nums);
      free(sub_avgs);
  }
  free(sub_rand_nums);

  MPI_Barrier(MPI_COMM_WORLD);
  MPI_Finalize();
}
```

Output:

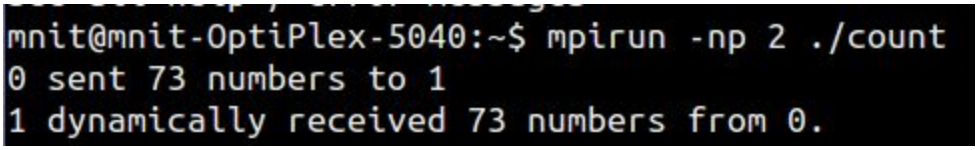
```
mnit@mnit-OptiPlex-5040:~$ mpirun -np 4 ./scaga 3
Avg of all elements is 0.519614
Avg computed across original data is 0.519614
mnit@mnit-OptiPlex-5040:~$ mpirun -np 4 ./scaga 5
Avg of all elements is 0.435156
Avg computed across original data is 0.435156
mnit@mnit-OptiPlex-5040:~$ gedit scaga.c
```

2) **MPI_Get_count()**

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

```
int main(int argc, char** argv) {
  MPI_Init(NULL, NULL);

  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);
  if (world_size != 2) {
    fprintf(stderr, "Must use two processes for this example\n");
    MPI_Abort(MPI_COMM_WORLD, 1);
  }
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

  int number_amount;
  if (world_rank == 0) {
    const int MAX_NUMBERS = 100;
    int numbers[MAX_NUMBERS];
    // Pick a random amont of integers to send to process one
    srand(time(NULL));
    number_amount = (rand() / (float)RAND_MAX) * MAX_NUMBERS;
    // Send the amount of integers to process one
    MPI_Send(numbers, number_amount, MPI_INT, 1, 0, MPI_COMM_WORLD);
    printf("0 sent %d numbers to 1\n", number_amount);
  } else if (world_rank == 1) {
    MPI_Status status;
    // Probe for an incoming message from process zero
    MPI_Probe(0, 0, MPI_COMM_WORLD, &status);
    // When probe returns, the status object has the size and other
    // attributes of the incoming message. Get the size of the
message.
    MPI_Get_count(&status, MPI_INT, &number_amount);
    // Allocate a buffer just big enough to hold the incoming numbers
    int* number_buf = (int*)malloc(sizeof(int) * number_amount);
    // Now receive the message with the allocated buffer
    MPI_Recv(number_buf, number_amount, MPI_INT, 0, 0,
MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
    printf("1 dynamically received %d numbers from 0.\n",
           number_amount);
    free(number_buf);
  }
  MPI_Finalize();
}
```

Output:



3) **MPI_Issend() & MPI_Irecv()**

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[])
{
      int myid, numprocs, left, right,i;
      int buffer[10], buffer2[10];
      MPI_Request request, request2;
      MPI_Status status;

      MPI_Init(&argc,&argv);
      MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
      MPI_Comm_rank(MPI_COMM_WORLD, &myid);
      right = (myid + 1) % numprocs;
      left = myid - 1;
      if (left < 0)
      left = numprocs - 1;
      MPI_Irecv(buffer, 10, MPI_INT, left, 123, MPI_COMM_WORLD,
&request);
      MPI_Issend(buffer2, 10, MPI_INT, right, 123, MPI_COMM_WORLD,
&request2);
      for(i=0;i<10;i++)
      printf("Value is:%d\n",buffer[i]);
      MPI_Wait(&request, &status);
      MPI_Wait(&request2, &status);
```

```
    MPI_Finalize();
    return 0;
}
```

Output:

```
mnit@mnit-OptiPlex-5040:~/n6$ mpirun -np 4 5
Value is:1254359544
Value is:32574
Value is:1266391760
Value is:32574
Value is:0
Value is:0
Value is:-1276493800
Value is:32764
Value is:1
Value is:0
Value is:2105172472
Value is:32753
Value is:2117204688
Value is:32753
Value is:0
Value is:0
Value is:-731666664
Value is:32766
Value is:1
Value is:0
Value is:1408020984
Value is:32661
Value is:1420053200
Value is:32661
Value is:0
Value is:0
Value is:2143762648
Value is:32766
Value is:1
Value is:0
Value is:1605657080
Value is:32563
Value is:1617689296
Value is:32563
Value is:0
Value is:0
```

5) **MPI_Test()**

```
#include <stdio.h>
#include <mpi.h>
#include <sys/time.h>

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

int rank, size;
MPI_Status status;

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

if (rank != 0) { // Slaves
      int buf;

      if (rank == 1) {
      buf = 1;
      MPI_Send(&buf, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
      }
      if (rank == 2) {
      buf = 2;
      MPI_Send(&buf, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
      }

}
else { // Master
      int sum = 0;
      int flag = -1, res;
      MPI_Request request;
      MPI_Status status;
      while (1) {
      if(flag != 0)
      {
      MPI_Irecv(&res, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
MPI_COMM_WORLD, &request);
```

```
	flag = 0;
	}
	MPI_Test(&request, &flag, &status);

	if (flag != 0) {
		printf("recv : %d, slave : %d\n", res, status.MPI_SOURCE);
		if (status.MPI_SOURCE != -1)
			sum += res;
	flag = -1;
	}


	if (sum == 3)
		break;
	}

	printf("sum : %d\n", sum);
}

MPI_Finalize();
return 0;

}
```

Output:

```
mnit@mnit-OptiPlex-5040:~/n6$ mpirun -np 4 6
recv : 2, slave : 2
recv : 1, slave : 1
sum : 3
```

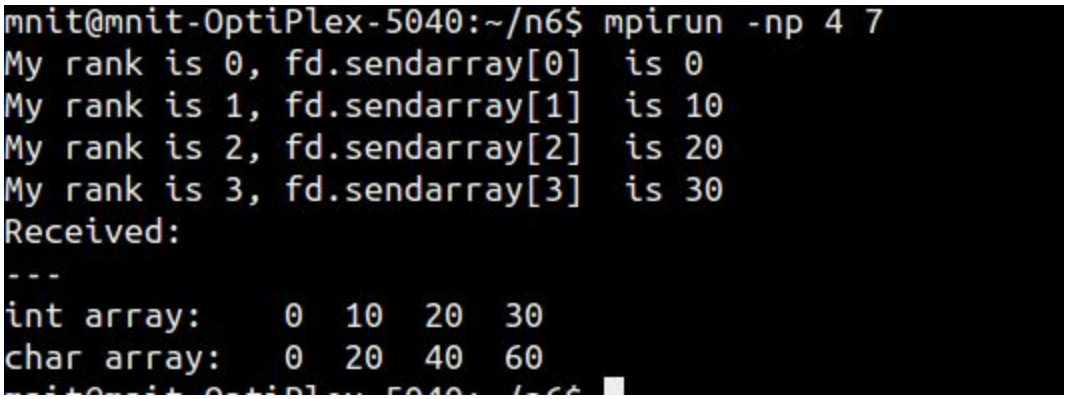
6) **MPI_Allgather()**

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

struct mystruct{
	int sendarray[4];
	int a;
	char array2[4];
};

typedef struct mystruct struct_t;
```

```
int main (int argc, char ** argv)
{
     int rank, size;
     struct_t fd;
     struct_t recv;
     int i, j;
     MPI_Init(&argc, &argv);
     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
     MPI_Comm_size(MPI_COMM_WORLD, &size);
     // init
     for (i=0;i<4;i++){
     fd.sendarray[i] = 0;
     fd.array2[i] = 0;

     recv.sendarray[i] =999;
     recv.array2[i]  = 99;
     }
     recv.a =999;

     // put some stuff in your array
     fd.sendarray[rank] = rank*10;
     fd.array2[rank] = (char)(rank*20);
     fd.a = rank;
     printf("My rank is %d, fd.sendarray[%d]  is %d\n", rank, rank,
fd.sendarray[rank]);

     // gather data from all now.. send the int:
     MPI_Allgather (&(fd.sendarray[rank]), 1, MPI_INT,
recv.sendarray, 1, MPI_INT,  MPI_COMM_WORLD);
     // then the char
     MPI_Allgather (&(fd.array2[rank]),   1, MPI_CHAR, recv.array2,
1, MPI_CHAR, MPI_COMM_WORLD);

     // check if correct data has been received
     if (rank == 0) {
     printf("Received:\n");
     printf("---\n");
     printf("int array:  ");
     for (j=0; j<4; j++) printf("%3d ", recv.sendarray[j]);
     printf("\nchar array: ");
     for (j=0; j<4; j++) printf("%3d ", (int)(recv.array2[j]));
     printf("\n");
```

```
	}
	MPI_Finalize();

	return 0;
}
```

Output:



7) **MPI_Alltoall()**

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

int main( int argc, char **argv )
{
	int send[4], recv[3];
	int rank, size, k;

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

	if (size != 4) {
	printf("Error!:# of processors must be equal to 4");
	printf("Programm aborting....");
	MPI_Abort(MPI_COMM_WORLD, 1);
	}
	for (k=0;k<size;k++)
	send[k] = (k+1) + rank*size;

	printf("%d : send = %d %d %d %d\n", rank, send[0], send[1],
send[2], send[3]);
```

```
	MPI_Alltoall(&send, 1, MPI_FLOAT, &recv, 1, MPI_INT,
MPI_COMM_WORLD);

	printf("%d : recv = %d %d %d %d\n", rank, recv[0], recv[1],
recv[2], recv[3]);

	MPI_Finalize();
	return 0;
}
```

Output: