Lab#8: Reduction in MPI

Reduction

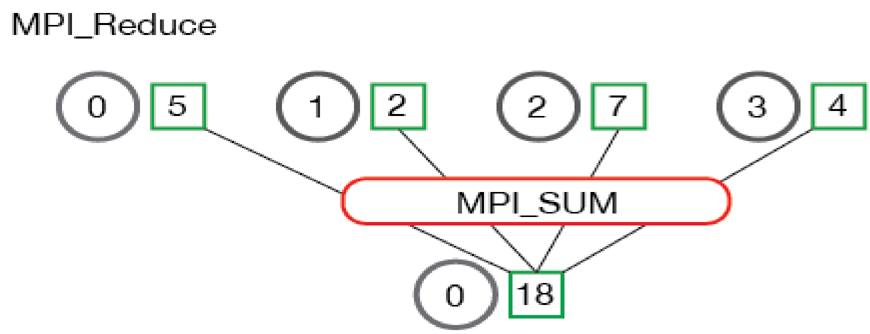
Similar to MPI_Gather, MPI_Reduce takes an array of input elements on each process and returns an array of output elements to the root process. The output elements contain the reduced result. The prototype for MPI_Reduce looks like this:

```
MPI_Reduce(void* send_data, void* recv_data, int count,MPI_Datatype datatype, MPI_
Op op, int root,MPI_Comm communicator)
```

The send_data parameter is an array of elements of type datatype that each process wants to reduce. The recv_data is only relevant on the process with a rank of root. The recv_data array contains the reduced result and has a size of sizeof(datatype) * count. The op parameter is the operation that you wish to apply to your data. MPI contains a set of common reduction operations that can be used. Although custom reduction operations can be defined, it is beyond the scope of this lesson. The reduction operations defined by MPI include:

- MPI_MAX - Returns the maximum element.
- MPI_MIN - Returns the minimum element.
- MPI_SUM - Sums the elements.
- MPI_PROD - Multiplies all elements.
- MPI_LAND - Performs a logical *and* across the elements.
- MPI_LOR - Performs a logical *or* across the elements.
- MPI_BAND - Performs a bitwise *and* across the bits of the elements.
- MPI_BOR - Performs a bitwise *or* across the bits of the elements.
- MPI_LXOR - Logical XOR
- MPI_BXOR -Botwise XOR
- MPI_MAXLOC - Returns the maximum value and the rank of the process that owns it.
- MPI_MINLOC - Returns the minimum value and the rank of the process that owns it.

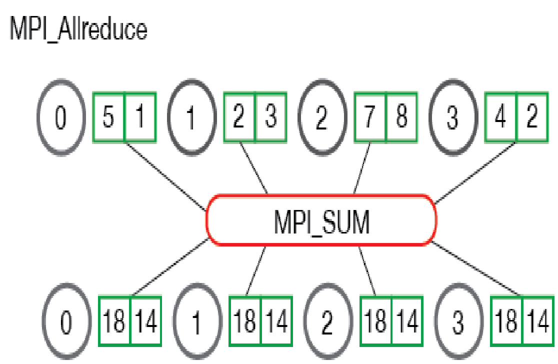
Below is an illustration of the communication pattern of MPI_Reduce.



In the above, each process contains one integer. MPI_Reduce is called with a root process of 0 and using MPI_SUM as the reduction operation. The four numbers are summed to the result and stored on the root process. It is also useful to see what happens when processes contain multiple elements. The illustration below shows reduction of multiple numbers per process.

MPI_Allreduce: If the result of the reduction operation is needed by all processes, MPI provides:

int MPI_Allreduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)



The MPI Timer

The elapsed (wall-clock) time between two points in an MPI program can be computed using MPI_Wtime:

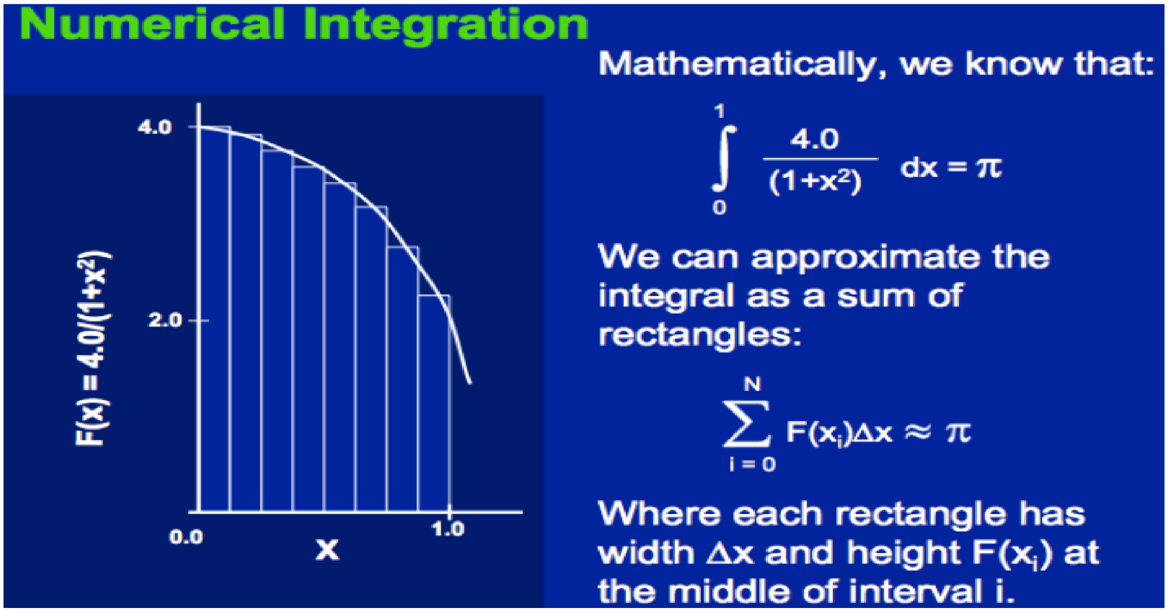
```
double t1, t2;
t1 = MPI_Wtime();
...
t2 = MPI_Wtime();
printf( "Elapsed time is %f\n", t2 - t1 );
```

Example#1: Sum of Array Elements using Reduce

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

int main(int argc, char *argv[]) {
   int rank, size;
   int array_size = 100;
   int local_array[array_size];
   int global_sum = 0;
   int local_sum = 0;

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

   // Check if the number of processes is less than the array size
   if (size > array_size) {
      if (rank == 0) {
         printf("The number of processes should be less than or equal to the array size.\n");
      }
      MPI_Finalize();
      return 1;
   }

   // Initialize the array with values
   for (int i = 0; i < array_size; i++) {
      local_array[i] = i + 1; // Initialize array with values from 1 to 100
   }

   // Scatter the array among processes
   MPI_Scatter(local_array, array_size / size, MPI_INT, local_array, array_size / size,
MPI_INT, 0, MPI_COMM_WORLD);

   // Compute local sum
   for (int i = 0; i < array_size / size; i++) {
      local_sum += local_array[i];
   }

   // Reduce local sums to get the global sum
   MPI_Reduce(&local_sum, &global_sum, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);
```

```
    // Print the global sum (only process 0)
    if (rank == 0) {
        printf("Global Sum: %d\n", global_sum);
    }

    // Finalize MPI
    MPI_Finalize();

    return 0;
}
```

Task#1: Write a program in MPI C to estimate the value of Pi using Reduction on 4 processes.



Task#2: Write a program in MPI C that find the maximum value in an array of 10,000 random integers using MPI_Reduce clause with four processes.