In this assignment, you will write your own global summation program (equivalent to \texttt{MPI\_Allreduce}) using \texttt{MPI\_Send} and \texttt{MPI\_Recv}. Your program should run on \( P = 2^l \) processors \((l = 0, 1, \ldots)\). Each process contributes a partial value, and at the end, all the processes will have the globally summed value of these partial contributions.

Your program will use a communication structure called butterfly, which is structured as a series of pairwise exchanges (see the figure below where messages are denoted by arrows). This structure allows a global reduction among \( P \) processes to be performed in \( \log_2 P \) steps.

\[
a_{000} + a_{001} + a_{010} + a_{011} + a_{100} + a_{101} + a_{110} + a_{111} = ((a_{000} + a_{001}) + (a_{010} + a_{011})) + ((a_{100} + a_{101}) + (a_{110} + a_{111}))
\]

At each level \( l \), a process exchanges messages with a partner whose rank differs only at the \( l \)-th bit position in the binary representation.

**Hypercube Template**

We can use the following template to perform a global reduction using any associative operator \( \texttt{OP} \) (such as multiplication or maximum), \((a \texttt{ OP } b) \texttt{ OP } c = a \texttt{ OP } (b \texttt{ OP } c)\).

\[
\begin{align*}
\text{procedure hypercube}(\text{myid}, \text{input}, \log P, \text{output}) & \\
& \begin{aligned}
& \text{mydone := input;} \\
& \text{for } l := 0 \text{ to } \log P - 1 \text{ do} \\
& \begin{aligned}
& \text{partner := myid XOR } 2^l; \\
& \text{send mydone to partner;} \\
& \text{receive hisdone from partner;} \\
& \text{mydone = mydone } \texttt{ OP } \text{ hisdone}
\end{aligned}
\end{aligned}
\text{end}
\end{aligned}
\]

\[
\text{output := mydone}
\]

**Use of Bitwise Logical XOR**

Note that
\[
0 \texttt{ XOR } 0 = 1 \texttt{ XOR } 1 = 0;
0 \texttt{ XOR } 1 = 1 \texttt{ XOR } 0 = 1.
\]

so that \( a \texttt{ XOR } 1 \) flips the bit \( a \), \textit{i.e.},
\[ a \text{ XOR } 1 = \bar{a} \]
\[ a \text{ XOR } 0 = a \]

where \( \bar{a} \) is the complement of \( a \) (\( \bar{a} = 1 \) for \( a = 0 \)). In particular, \( \text{myid XOR 2^l} \) reverses the \( l \)-th bit of the rank of this process, \( \text{myid} \):

\[ \text{abcdefg XOR 0000100 = abcd \bar{efg}} \]

Note that the XOR operator is \(^\wedge\) (caret symbol) in the C programming language.

**ASSIGNMENT**

Complete the following program by implementing the function, \( \text{global_sum} \), using \( \text{MPI_Send} \) and \( \text{MPI_Recv} \) functions and the hypercube template given above.

*Submit the source code as well as the printout from a test run on 4 processors and that on 8 processors.*

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

int nprocs; /* Number of processors */
int myid; /* My rank */

double global_sum(double partial) {
    /* Implement your own global summation here */
}

int main(int argc, char *argv[]) {
    double partial, sum, avg;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    partial = (double) myid;
    printf("Node %d has %le\n", myid, partial);
    sum = global_sum(partial);
    if (myid == 0) {
        avg = sum/nprocs;
        printf("Global average = %le\n", avg);
    }
    MPI_Finalize();
    return 0;
}
```