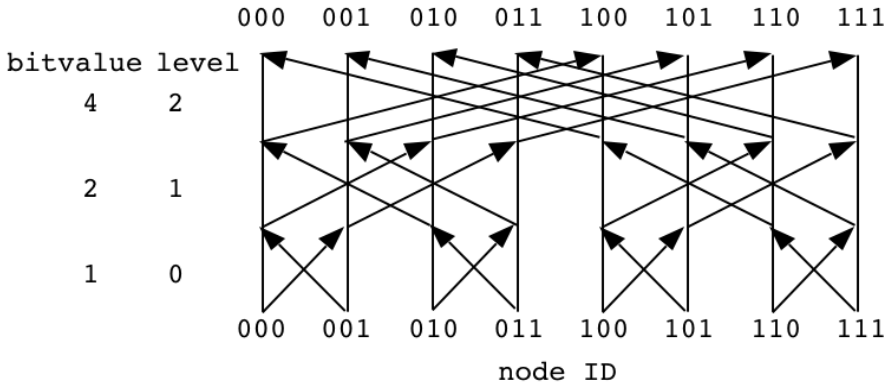
CSCI596 Assignment 2—Message Passing Interface

Due: September 18 (Mon), 2017, 11:59 pm

In this assignment, you will write your own global summation program (equivalent to `MPI_Allreduce`) using `MPI_Send and MPI_Recv`. Your program should run on $P = 2^l$ processors ($l = 0, 1,...$). 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.




```
   a000 + a001 + a010 + a011 + a100 + a101 + a110 + a111
=  ((a000 + a001) + (a010 + a011)) + ((a100 + a101) + (a110 + a111))
```

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 `OP` (such as multiplication or maximum), `(a OP b) OP c = a OP (b OP c)`.

```
procedure hypercube(myid, input, logP, output)
begin
  mydone := input;
  for l := 0 to logP-1 do
  begin
    partner := myid XOR 2^l;
    send mydone to partner;
    receive hisdone from partner;
    mydone = mydone OP hisdone
  end
  output := mydone
end
```

USE OF BITWISE LOGICAL XOR

Note that

```
0 XOR 0 = 1 XOR 1 = 0;
0 XOR 1 = 1 XOR 0 = 1.
```

so that `a XOR 1` flips the bit `a`, *i.e.*,

```
a XOR 1 = ā
a XOR 0 = a
```

where ā is the complement of a (ā = 1|0 for a = 0|1). In particular, `myid XOR` 2^l reverses the l-th bit of the rank of this process, `myid`:

```
abcdefg XOR 0000100 = abcd ēfg
```

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

ASSIGNMENT

Complete the following program by implementing the function, `global_sum`, using `MPI_Send` and `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.

```
#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;
}
```
