In class we looked at two possible “parallelizations” of the following serial pseudocode:

```
int n; // Initialized
double x[n]; // Initialized
double sum = 0;
for (int i = 0; i < n; i++)
    sum += x[i];
output sum;
```

The first parallelization was for a shared memory MIMD system:

```
shared int n, p; // Initialized
shared double x[n]; // Initialized
shared double sum; // Initialized to 0
shared double my_sum[p],
shared int Im_done[p];
private int my_rank = Get_my_rank();
my_sum[my_rank] = 0;
Im_done[my_rank] = False;
private int my_first_elt, my_last_elt, my_count;
my_count = n/p; // p evenly divides n
my_first_elt = my_rank*my_count;
my_last_elt = my_first_elt + my_count;

for (private int i = my_first_elt; i < my_last_elt; i++)
    my_sum[my_rank] += x[i];
Im_done[my_rank] = True;
if (my_rank == 0) {
    for (private int i = 0; i < p; i++) {
        while (!Im_done[i]); // Busy-wait loop
        sum += my_sum[i];
    }
    Output sum;
}
```
The second parallelization was for a distributed memory MIMD system:

```c
// All variables are private, but some are replicated across the nodes
int n, p; // Initialized and replicated
int my_count = n/p; // p evenly divides n
double my_x[my_count]; // Initialized; not replicated
int my_rank = Get_my_rank();
double temp, sum, my_sum = 0;
for (int i = 0; i < my_count; i++)
    my_sum += my_x[i];
if (my_rank == 0) {
    sum = my_sum;
    for (int i = 1; i < p; i++) {
        Receive my_sum from node i and store in temp;
        sum += temp;
    }
    Output sum;
} else { // my_rank > 0
    Send my_sum to node 0;
}
```

In both programs, only node/core 0 has access to the solution. Describe how to modify both programs so that when the program is completed, all the nodes/cores will have access to the solution.