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

#define MAXSIZE 10000000


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

    double* data = NULL;

    int i, x, low, high;
    int myid, numprocs;
    int dest, source;
    double myresult, result, result_temp;
    double starttime, endtime;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);

   // ogni processo avrà un proprio vettore (duplicazione struttura dati)
    data = new double[MAXSIZE];

    result = 0;
    myresult = 0;


    // Init...(each process will see its own portion!)
    // ogni processo inzializza TUTTO il vettore, anche se lavorerà sulla propria porzione di
    for (i=0; i<MAXSIZE;i++)
        data[i] = i;

    MPI_Barrier(MPI_COMM_WORLD);
    starttime = MPI_Wtime();

    // la mia porzione di competenza
    x = MAXSIZE/numprocs;
    low = myid * x;
    high = low + x;

    // Compute my result (even Process 0 will do it) - Master Slave Democratico
    for (i=low; i<high; i++)
        myresult = myresult + data[i];

     if (myid == 0){
        result = myresult;
        for (source=1; source<numprocs; source++){
            MPI_Recv(&myresult, 1, MPI_DOUBLE, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &status);
        result = result + myresult;
        }
     }
        else
            MPI_Send(&myresult, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);

    MPI_Barrier(MPI_COMM_WORLD);
    endtime = MPI_Wtime();

    if (myid == 0){
        printf("Sum is %e.\n", result);
        printf("Elapsed time: %f\n", 1000*(endtime - starttime));
    }
    delete[] data;
    MPI_Finalize();
    exit(0);
}

```