```
/*****************************************
 E8: SOLUCIÓN A LOS DOS PROBLEMAS PROPUESTOS
*****************************************/

//-----------------
// PROBLEMA 1:
//-----------------

// a) Rutina
// He quitado includes y demás, solo dejo lo imprescindible para el problema
// ...

int main( int argc, char *argv[])
{
  double x[k]; // como esta repartido, son suficientes k elementos por proceso, no hace falta
n.
  double suma_local,suma_total;

  MPI_Init(&argc,&argv);

  suma_local=0;
  for (i=0; i<k; i++) {
    suma_local+=x[i];
  }

MPI_Allreduce(&suma_local,&suma_total,1,MPI_DOUBLE,MPI_SUM,MPI_COMM_WORLD
);
  // Otra solución consiste en utilizar Reduce seguido de Broadcast:
  //
MPI_Reduce(&suma_local,&suma_total,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD
);
  // MPI_Bcast(&suma_total,1,MPI_DOUBLE,0,MPI_COMM_WORLD);

  MPI_Finalize();
  return 0;
}

/*
b) Modelo temporal

ta -> Op float,
tcom=tin +m*tc
m -> bytes que se transmiten
tin=10tc
tc -> Comunicar 1 byte
```

```
Vamos a suponer que tanto la opción Allreduce como Reduce seguido de Bcast tienen el
mismo coste
(en un sistema real, Allreduce podría estar optimizado, pero no hay elementos en el
problema que nos
permmitan saberlo).

Reduce -> tcom = log2(p)*(tin+m*tc)
          tcal = log2(p)*op*ta
  m en este caso es 8 bytes (un double)
  op en este caso es 1 pues solo se realiza una operación (una suma pues solo hay un
elemento)
  Por lo tanto:
    tcom = log2(p)*(10tc+8tc) = 18log2(p)tc
    tcal = log2(p)*ta

Bcast -> tcom = log2(p)*(tin+m*tc)
  m es 8 pues solo enviamos un double
  Por lo tanto:
    tcom = log2(p)*(10tc+8tc) = 18log2(p)tc

Calculos en el bucle
    tcal = k*ta

Sumando por tipo de contribución:
  tcom = 18log2(p)tc + 18log2(p)tc = 36log2(p)tc
  tcal = ( log2(p) + k )ta
  Ttotal = tcom + tcal =  36log2(p)tc + (log2(p) + k)ta
*/



//-----------------
// PROBLEMA 2:
//-----------------

// a) Rutina
// He quitado includes y demás, solo dejo lo imprescindible para el problema
// ...

int main( int argc, char *argv[])
{
  double xl[n][n];
  double x[n][n]; // bastaria con que esta matriz estuviera solo en 0 que es el unico que la
utiliza

  MPI_Init(&argc,&argv);
```

```
  // si suponemos que los elementos de la matriz están consecutivos en memoria,
  // basta con la siguiente instruccion:
  MPI_Reduce(&xl[0][0],&x[0][0],n*n,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);

  // si la matriz no se ha definido de forma estática, o no estamos seguros de cómo
  // se ha creado, deberíamos proceder así:
  // for (i=0; i<n; i++) {
  //   MPI_Reduce(xl[i],x[i],n,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
  // }
  // en este segundo caso, los costes temporales aumentan un poco por el tiempo inicial de
cada reduce

  MPI_Finalize();
  return 0;
}

/*
b) Modelo temporal

ta -> Op float,
tcom=m*tc   // en este caso no tenemos tin o tin=0
m -> bytes que se transmiten
tc -> Comunicar 1 byte

Reduce -> tcom = log2(p)*(m*tc)
          tcal = log2(p)*op*ta

En este caso se transmiten n*n doubles (m=8*n*n bytes) y se hacen n*n sumas (op=n*n)
Por lo tanto:
  tcom = log2(p)*8*n*n*tc
  tcal = log2(p)*n*n*ta
Sumamos términos:
  Ttotal= tcom + tcal = log2(p)*n*n*(ta+8tc)

Si se optó por la opción de un reduce por fila, el resultado sería:
  tcom = n*log2(p)*8*n*tc
  tcal = n*log2(p)*n*ta  // se queda igual
Sumamos términos:
  Ttotal= tcom + tcal = log2(p)*n*n*(ta+8tc)

  En ambos casos el resultado coincide, pero si hubiéramos tenido tin, entonces
  el caso con el bucle hubiera tardado más.
 */
```