FARMER en C y C++ bajo MPICH

Antonio Carrillo Ledesma
http://www.mmc.igeofcu.unam.mx/acl

Objetivo

El presente proyecto es sobre la implementación de un Maestro-Esclavo (farmer) en el lenguaje de programación C y C++ bajo MPICH trabajando en un cluster Linux Debian. Donde tomando en cuenta la implementación en estrella del cluster y el modelo de paralelismo de MPICH, el nodo maestro tendrá comunicación sólo con cada nodo esclavo y no existirá comunicación entre los nodos esclavos. Esto reducirá las comunicaciones y optimizará el paso de mensajes.

El problema a tratar, será el cálculo sobre un plano en 2D delimitado por (0,0) y (1,1) esquina inferior y superior respectivamente, en el cual se propone tomar una malla uniforme de puntos espaciados en Δx y Δy respectivamente. En el plano se calcula un diagrama de Lenguas de Arnold (http://www.mmc.igeofcu.unam.mx/acl/circle/).

Estructura Maestro-Esclavo

El Maestro manda a procesar a cada nodo esclavo una línea horizontal, cuando algún nodo esclavo termine la tarea asignada, avisa al maestro para que se le asigne otra tarea. No se mandan puntos independientes de la malla, ya que la comunicación crecería y la inactividad se incrementaría, por ello es mejor mandar una línea de la partición a cada trabajador.

La estructura básica del Maestro-Esclavo en C es codificada como

```
main(int argc, char *argv[]) {
     MPI_Init(&argc, &argv);
     MPI_Comm_rank(MPI_COMM_WORLD, &id);
     if (id == 0) {
          // Operaciones del Maestro
     }
     else {
          // Operaciones del esclavo
     }
     .
     .
     MPI_Finalize();
}
```

La estructura básica del Maestro-Esclavo en C++ es codificada como

```
main(int argc, char *argv[]) {
      MPI::Init(argc,argv);
      id = MPI::COMM_WORLD.Get_rank();
      if (id == 0) {
            // Operaciones del Maestro
      }
      else {
            // Operaciones del esclavo
      }
      .
      .
      MPI::Finalize();
}
```

En ella hay que implementar las operaciones para que el maestro asigne la primera tarea a cada nodo esclavo y posteriormente espere a que algún nodo esclavo termine para mandarle a ese mismo nodo otra tarea. Al termino de las tareas asignadas hay que esperar a que los nodos ocupados actualmente terminen las tareas pendientes y hasta que todas ellas sean concluidas se mandará el aviso a cada nodo esclavo que concluye el programa.

Nodo maestro en C:

```
printf("Maestro-Esclavo, Numero de Esclavos %d\n",ME_numprocs-1);

ME_P = 1;
ME_sw = 1;
time_t inicio,final;
inicio=time(NULL);
double T;
for (T = 0.0; T < 1.0; T+=0.0009765625)
{
        // Llena de trabajo a los trabajadores
        if (ME_P < ME_numprocs)
        {
                // Aviso de envio de una nueva tarea al nodo P
                MPI_Send(&ME_sw, 1, MPI_INT, ME_P,1, MPI_COMM_WORLD);
                // Preparacion para envio de una nueva tarea

                // Envio de una nueva tarea al nodo P
                MPI_Send(&T, 1, MPI_DOUBLE, ME_P,0, MPI_COMM_WORLD);
                ME_P++;
        } else {
                // Recibo de terminación de tarea del nodo L
                MPI_Recv(&ME_L, 1, MPI_INT, MPI_ANY_SOURCE,0, MPI_COMM_WORLD,&ME_St);
                // Aviso de envio de una nueva tarea al nodo L
                MPI_Send(&ME_sw, 1, MPI_INT, ME_L,1, MPI_COMM_WORLD);
                // Preparacion para envio de una nueva tarea

                // Envio de una nueva tarea al nodo L
                MPI_Send(&T, 1, MPI_DOUBLE, ME_L,0, MPI_COMM_WORLD);
        }
}
ME_P--;
while(ME_P> 0)
{
        // Recibo de terminación de tarea del nodo L
        MPI_Recv(&ME_L, 1, MPI_INT, MPI_ANY_SOURCE,0, MPI_COMM_WORLD, &ME_St);
        ME_P--;
}
```

```
ME_sw = 0;
// Aviso de terminación de tareas al nodo P
for (ME_P = 1; ME_P < ME_numprocs; ME_P++) MPI_Send(&ME_sw, 1, MPI_INT, ME_P,1, MPI_COMM_WORLD);
final=time(NULL);
printf("El tiempo empleado fue %lf segundos\n",difftime(final,inicio));
```

Nota al nodo maestro, el nodo esclavo no retorna el producto de su trabajo al nodo maestro, ya que en este caso se generan archivos de varios gigabytes, lo cual no hace practico enviárselos al nodo maestro, por ello cada nodo esclavo genera un colector de trabajo, este es un archivo en el cual graba todos los datos generados y al termino cierra este archivo.

Para el nodo esclavo en C:

```
 // Recolector de trabajo
FILE* datos;
char xcad[100];
sprintf(xcad,"DatS%d.dat",ME_id);
datos = fopen(xcad,"wt");


int sw;
while (1)
{
        double A;
        // Recibo aviso de envio de una nueva tarea
         MPI_Recv(&sw, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, &ME_St);
         if (!sw) {
                break;
        }
        // Recibo una nueva tarea
        MPI_Recv(&A, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &ME_St);

        // Procesamiento de la tarea
        int Pr,Qr;
        double B;
        for (B = 0.0; B < 1.0; B+=0.0009765625) {
                if (Resonancias(100, 100, 50, A, B, 0.1,&Pr,&Qr) ) fprintf(datos,"%lf %lf %d %d\n",A,B,Pr,Qr);
        }

         // Avisa la terminacion del trabajo
        MPI_Send(&ME_id, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
}
fclose(datos);
```

Nodo maestro en C++:

```
// Maestro
printf("Maestro-Esclavo, Numero de Esclavos %d\n",MP_np-1);

ME_P = 1;
ME_sw = 1;
time_t inicio,final;
inicio=time(NULL);

double T;
for (T = 0.0; T < 1.0; T+=0.0009765625)
{
        // Llena de trabajo a los trabajadores
        if (ME_P < MP_np)
        {
                // Aviso de envio de una nueva tarea al nodo P
                MPI::COMM_WORLD.Send(&ME_sw, 1, MPI::INT, ME_P,1);
                // Preparacion para envio de una nueva tarea

                // Envio de una nueva tarea al nodo P
                MPI::COMM_WORLD.Send(&T, 1, MPI::DOUBLE, ME_P,0);
```

```
                ME_P++;
        } else {
                // Recibo de terminación de tarea del nodo L
                MPI::COMM_WORLD.Recv(&ME_L, 1, MPI::INT, MPI_ANY_SOURCE,0);

                // Aviso de envio de una nueva tarea al nodo L
                MPI::COMM_WORLD.Send(&ME_sw, 1, MPI::INT, ME_L,1);
                // Preparacion para envio de una nueva tarea

                // Envio de una nueva tarea al nodo L
                MPI::COMM_WORLD.Send(&T, 1, MPI::DOUBLE, ME_L,0);
        }
}
ME_P--;
while(ME_P> 0)
{
         // Recibo de terminación de tarea del nodo L
         MPI::COMM_WORLD.Recv(&ME_L, 1, MPI::INT, MPI_ANY_SOURCE,0);
        ME_P--;
}
ME_sw = 0;
// Aviso de terminación de tareas al nodo P
for (ME_P = 1; ME_P < MP_np; ME_P++) MPI::COMM_WORLD.Send(&ME_sw, 1, MPI::INT, ME_P,1);
final=time(NULL);
printf("El tiempo empleado fue %lf segundos\n",difftime(final,inicio));
```

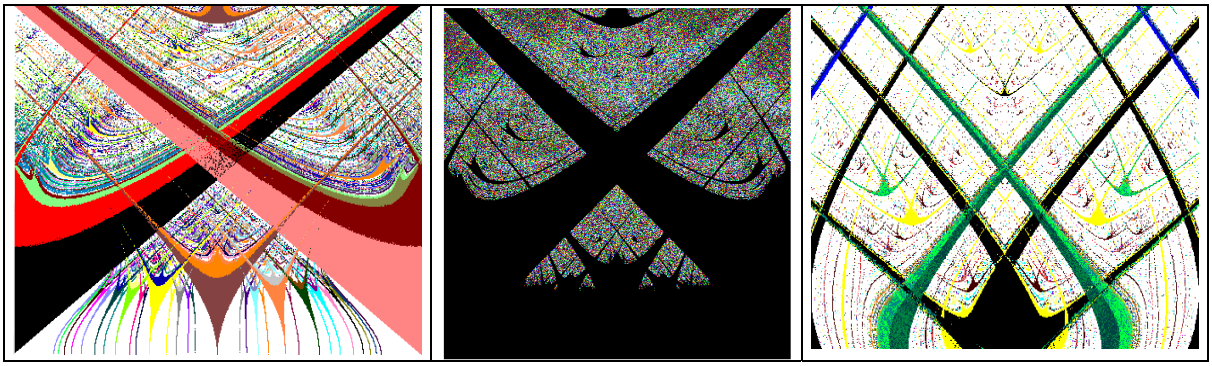
Para el nodo esclavo en C++:

```
// Recolector de trabajo
FILE* datos;
char xcad[100];
sprintf(xcad,"DatS%d.dat",ME_id);
datos = fopen(xcad,"wt");

int sw;
while (1)
{
        double A;
        // Recibo aviso de envio de una nueva tarea
        MPI::COMM_WORLD.Recv(&sw, 1, MPI::INT, 0,1);
        if (!sw)
        {
                break;
        }
        // Recibo una nueva tarea
        MPI::COMM_WORLD.Recv(&A, 1, MPI::DOUBLE, 0,0);
        // Procesamiento de la tarea

        int Pr,Qr;
        double B;
        for (B = 0.0; B < 1.0; B+=0.0009765625)
        {
                if (Resonancias(100, 100, 50, A, B, 0.1,&Pr,&Qr) ) fprintf(datos,"%lf %lf %d %d\n",A,B,Pr,Qr);
        }

        // Avisa la terminacion del trabajo
         MPI::COMM_WORLD.Send(&ME_id, 1, MPI::INT, 0,0);
}
fclose(datos);
```

El problema del cálculo de lenguas de Arnold, es un problema de búsqueda de regiones de sincronización en el espacio de parámetros de la función de Arnold (o cualquiera de sus variantes):

$F(t) = t + A + B*sin(2.0*M\_PI*t)$.

A continuación muestro de alguno de estos gráficos calculados por el programa a modo de prueba usando el esquema maestro-esclavo objeto de este proyecto.



El cálculo de cada una de estas gráficas consume muchos recursos de cómputo y pese a que los archivos que se generan son grandes, no se compara con el consumo de tiempo de procesamiento, por ejemplo el tiempo de cálculo empleando una PC a 3.4 GHtz para una de estas imágenes fue del orden (por la definición solicitada) de 18 hrs.

El programa completo está detallado en el apéndice, conjuntamente con la codificación de el cálculo de lenguas de Arnold las cuales son un par de rutinas en C para los programas Maestro-Esclavo en C y C++.

Conclusiones

El esquema de paralelización Maestro-Esclavo (Farmer), permite sincronizar por parte del nodo maestro las tareas que se realizan en paralelo usando varios nodos esclavos, este modelo puede ser explotado de manera eficiente si existe poca comunicación entre maestro-esclavo y los tiempos consumidos en realizar las tareas asignadas son mayores que los periodos involucrados en las comunicaciones para la asignación de dichas tareas. Ya que de esta manera se garantiza que la mayoría de los procesadores estarán trabajando de manera continua y existirán pocos tiempos muertos.

También es notoria la mayor eficiencia del código generado a partir del programa en C versus el correspondiente a C++, pero pese a las diferencias en tiempos de ejecución, muestran cualitativamente el mismo desempeño.

Apéndice

A continuación se detallan los códigos para realizar el esquema maestro-esclavo y las rutinas necesarias para el cálculo de las lenguas de Arnold. La parte de la programación del maestro-esclavo tiene como opción trazar lo que se esta ejecutando tanto en el nodo maestro como en el nodo esclavo al activar la directiva de pre-procesamiento VIS.

Código en C del programa Maestro-Esclavo

```
// Programa Maestro - Esclavo

// Compilar usando
//   mpicc me.c -o me -lm

// Correr usando 8 procesadores por ejemplo
//   mpirun -np 8 me



#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "/usr/lib/mpich/include/mpi.h"
#include <time.h>

// Funcion de Arnold
double F(double T,double A,double B);
//  Calcula  la  resonancia  de  algun  punto del espacio  de parametros
int Resonancias(int MaxPer,int Ciclos,int Trans,double A,double B, double Eps,int *Pr,int *Qr);



//#define VIS


// Programa Maestro-Esclavo
int main(int argc, char *argv[])
{
   int ME_id,MP_np;
   int ME_P, ME_L, ME_sw;
   MPI_Status ME_St;
         MPI_Init(&argc,&argv);
         MPI_Comm_rank(MPI_COMM_WORLD,&ME_id);
         MPI_Comm_size(MPI_COMM_WORLD,&MP_np);
         // Revisa que pueda arrancar el esquema M-E
         if (MP_np < 2)
   {
      printf("Se necesitan almenos dos procesadores para el esquema M-E\n");
      return 1;
   }


   // Controlador del esquema M-E
      if (ME_id == 0)
   {
      // Maestro
      printf("Maestro-Esclavo, Numero de Esclavos %d\n",MP_np-1);

      ME_P = 1;
                  ME_sw = 1;
      time_t inicio,final;
      inicio=time(NULL);


                  double T;
      for (T = 0.0; T < 1.0; T+=0.0009765625)
      {
         // Llena de trabajo a los trabajadores
                if (ME_P < MP_np)
         {
            // Aviso de envio de una nueva tarea al nodo P
            MPI_Send(&ME_sw, 1, MPI_INT, ME_P,1, MPI_COMM_WORLD);
            // Preparacion para envio de una nueva tarea

            // Envio de una nueva tarea al nodo P
            MPI_Send(&T, 1, MPI_DOUBLE, ME_P,0, MPI_COMM_WORLD);
#ifdef VIS
            printf("Maestro envio S %d\n",ME_P);
#endif
            ME_P++;
         } else {
            // Recibo de terminación de tarea del nodo L
```

```
            MPI_Recv(&ME_L, 1, MPI_INT, MPI_ANY_SOURCE,0, MPI_COMM_WORLD, &ME_St);
#ifdef VIS
            printf("Tarea Terminada S%d\n",ME_L);
#endif

            // Aviso de envio de una nueva tarea al nodo L
            MPI_Send(&ME_sw, 1, MPI_INT, ME_L,1, MPI_COMM_WORLD);
            // Preparacion para envio de una nueva tarea

            // Envio de una nueva tarea al nodo L
            MPI_Send(&T, 1, MPI_DOUBLE, ME_L,0, MPI_COMM_WORLD);
#ifdef VIS
            printf("Maestro envio S %d\n",ME_L);
#endif
         }
      }
      ME_P--;
#ifdef VIS
      printf("Tareas Pendientes %d\n",ME_P);
#endif
      while(ME_P> 0)
      {
          // Recibo de terminación de tarea del nodo L
          MPI_Recv(&ME_L, 1, MPI_INT, MPI_ANY_SOURCE,0, MPI_COMM_WORLD, &ME_St);
#ifdef VIS
          printf("Tarea Terminada S%d\n",ME_L);
#endif
          ME_P--;
#ifdef VIS
          printf("Tareas Pendientes %d\n",ME_P);
#endif
      }
      ME_sw = 0;
      // Aviso de terminación de tareas al nodo P
      for (ME_P = 1; ME_P < MP_np; ME_P++) MPI_Send(&ME_sw, 1, MPI_INT, ME_P,1, MPI_COMM_WORLD);
           final=time(NULL);
           printf("El tiempo empleado fue %lf segundos\n",difftime(final,inicio));

        } else {
      // Esclavos

      // Recolector de trabajo
      FILE* datos;
      char xcad[100];
         sprintf(xcad,"DatS%d.dat",ME_id);
            datos = fopen(xcad,"wt");


      int sw;
                while (1)
      {
                    double A;
              // Recibo aviso de envio de una nueva tarea
              MPI_Recv(&sw, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, &ME_St);
              if (!sw)
         {
#ifdef VIS
            printf("Termina esclavo: %d\n",ME_id);
#endif
            break;
          }
          // Recibo una nueva tarea
          MPI_Recv(&A, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &ME_St);
#ifdef VIS
          printf("Esclavo: %d ---> tarea: %lf\n",ME_id,A);
#endif
          // Procesamiento de la tarea


               int Pr,Qr;
               double B;
          for (B = 0.0; B < 1.0; B+=0.0009765625)
          {
             if   (Resonancias(100,   100,   50,   A,   B,   0.1,&Pr,&Qr)   )   fprintf(datos,"%lf   %lf   %d
%d\n",A,B,Pr,Qr);
          }

          // Avisa la terminacion del trabajo
          MPI_Send(&ME_id, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
      }
      fclose(datos);
         }

        MPI_Finalize();
        return 0;
}
```

Código en C++ del programa Maestro-Esclavo

```
// Programa Maestro - Esclavo

// Compilar usando
//   mpiCC me.cpp -o me -lm

// Correr usando 8 procesadores por ejemplo
//   mpirun -np 8 me



#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "/usr/lib/mpich/include/mpi.h"
#include <time.h>

// Funcion de Arnold
double F(double T,double A,double B);
//  Calcula  la  resonancia  de  algun  punto del espacio  de parametros
int Resonancias(int MaxPer,int Ciclos,int Trans,double A,double B, double Eps,int *Pr,int *Qr);



//#define VIS


// Programa Maestro-Esclavo
int main(int argc, char *argv[])
{
   int ME_id,MP_np;
   int ME_P, ME_L, ME_sw;

   MPI::Init(argc,argv);
   ME_id = MPI::COMM_WORLD.Get_rank();
   MP_np = MPI::COMM_WORLD.Get_size();

         // Revisa que pueda arrancar el esquema M-E
         if (MP_np < 2)
   {
      printf("Se necesitan almenos dos procesadores para el esquema M-E\n");
      return 1;
   }


   // Controlador del esquema M-E
         if (ME_id == 0)
   {
      // Maestro
                  printf("Maestro-Esclavo, Numero de Esclavos %d\n",MP_np-1);

      ME_P = 1;
                  ME_sw = 1;
      time_t inicio,final;
      inicio=time(NULL);


                  double T;
      for (T = 0.0; T < 1.0; T+=0.0009765625)
      {
         // Llena de trabajo a los trabajadores
                  if (ME_P < MP_np)
         {
            // Aviso de envio de una nueva tarea al nodo P
            MPI::COMM_WORLD.Send(&ME_sw, 1, MPI::INT, ME_P,1);
            // Preparacion para envio de una nueva tarea

            // Envio de una nueva tarea al nodo P
            MPI::COMM_WORLD.Send(&T, 1, MPI::DOUBLE, ME_P,0);
#ifdef VIS
            printf("Maestro envio S %d\n",ME_P);
#endif
            ME_P++;
         } else {
            // Recibo de terminación de tarea del nodo L
            MPI::COMM_WORLD.Recv(&ME_L, 1, MPI::INT, MPI_ANY_SOURCE,0);
#ifdef VIS
            printf("Tarea Terminada S%d\n",ME_L);
#endif

            // Aviso de envio de una nueva tarea al nodo L
            MPI::COMM_WORLD.Send(&ME_sw, 1, MPI::INT, ME_L,1);
            // Preparacion para envio de una nueva tarea

            // Envio de una nueva tarea al nodo L
            MPI::COMM_WORLD.Send(&T, 1, MPI::DOUBLE, ME_L,0);
#ifdef VIS
            printf("Maestro envio S %d\n",ME_L);
```

```
#endif
        }
      }
      ME_P--;
#ifdef VIS
      printf("Tareas Pendientes %d\n",ME_P);
#endif
      while(ME_P> 0)
      {
          // Recibo de terminación de tarea del nodo L
          MPI::COMM_WORLD.Recv(&ME_L, 1, MPI::INT, MPI_ANY_SOURCE,0);
#ifdef VIS
          printf("Tarea Terminada S%d\n",ME_L);
#endif
          ME_P--;
#ifdef VIS
          printf("Tareas Pendientes %d\n",ME_P);
#endif
      }
      ME_sw = 0;
      // Aviso de terminación de tareas al nodo P
      for (ME_P = 1; ME_P < MP_np; ME_P++) MPI::COMM_WORLD.Send(&ME_sw, 1, MPI::INT, ME_P,1);
           final=time(NULL);
           printf("El tiempo empleado fue %lf segundos\n",difftime(final,inicio));

        } else {
      // Esclavos

      // Recolector de trabajo
      FILE* datos;
      char xcad[100];
        sprintf(xcad,"DatS%d.dat",ME_id);
           datos = fopen(xcad,"wt");


      int sw;
                 while (1)
      {
                    double A;
              // Recibo aviso de envio de una nueva tarea
              MPI::COMM_WORLD.Recv(&sw, 1, MPI::INT, 0,1);
              if (!sw)
         {
#ifdef VIS
           printf("Termina esclavo: %d\n",ME_id);
#endif
           break;
         }
         // Recibo una nueva tarea
         MPI::COMM_WORLD.Recv(&A, 1, MPI::DOUBLE, 0,0);
#ifdef VIS
         printf("Esclavo: %d ---> tarea: %lf\n",ME_id,A);
#endif
         // Procesamiento de la tarea


              int Pr,Qr;
              double B;
         for (B = 0.0; B < 1.0; B+=0.0009765625)
         {
            if   (Resonancias(100,   100,   50,   A,   B,   0.1,&Pr,&Qr)   )   fprintf(datos,"%lf   %lf   %d
%d\n",A,B,Pr,Qr);
         }

         // Avisa la terminacion del trabajo
         MPI::COMM_WORLD.Send(&ME_id, 1, MPI::INT, 0,0);
      }
      fclose(datos);
        }

        MPI::Finalize();
        return 0;
}
```

Rutinas para el cálculo de Lenguas de Arnold

```
// Funcion de Arnold
double F(double T,double A,double B)
{
        return(T + A + B*sinl(2.0*M_PI*T));
}

//  Calcula  la  resonancia  de  algun  punto del espacio  de parametros
int Resonancias(int MaxPer,int Ciclos,int Trans,double A,double B, double Eps,int *Pr,int *Qr)
{
/*************************************************************************
 * Resonancias  calcula  la  resonancia  de  algun  punto del espacio  de  *
 * parametros                                                              *
 *                                                                         *
 * Entradas  :                                                             *
 *   MaxPer  : Maximo periodo para buscar                                  *
 *   Ciclos  : Numero de ciclos en la busqueda                             *
 *   Trans   : Longitud del transitorio                                    *
 *   A       : Valor del parametro A                                       *
 *   B       : Valor del parametro B                                       *
 *   Eps     : Tolerancia deseada para considerar que una orbita es cerrada*
 *                                                                         *
 * Salidas   : La  funcion  retorna  un  valor  verdadero  si  fue  posible*
 *             calcular la resonancia en algun punto, y en Pr y  Qr retorna *
 *             el numero de vueltas y el  periodo, respectivamente,  de  la *
 *             resonancia.                                                 *
 *************************************************************************/
  int Found=0;            /*** Variable auxiliar para determinar si se encontro alguna resonancia ***/
  int I=0;                /*** Contador de iteraciones ***/
  int J;                  /*** Contador de iteraciones ***/
  double aux;        /*** Variable auxiliar para obtener partes fraccionarias ***/
  double aux2;       /*** Variable auxiliar para obtener partes fraccionarias ***/
  double X;          /*** Auxiliar para almacenar las iteraciones de la funcion de disparos ***/
  double X0;         /*** Almacena el primer valor de las iteraciones para buscar las resonancias ***/
  int K;                  /*** Contador de ciclos ***/

  X0=random()/2.e09;   // Condicion inicial aleatoria
  do {                      // Transitorios
         X0=modf(F(X0,A,B),&aux);
         I++;
  } while (I<Trans);
  K=0;
  X=X0;
  do {       // Comienza otro ciclo
         do {     // Busca una primera aproximacion
                I=0;
                while ((!Found) && (I<MaxPer))   // busca la resonancia
                  {                                          // y halla una primera orbita
         X=modf(X,&aux);                                                    // periodica
                        X=F(X,A,B);
                        Found=((fabs(X0-modf(X,&aux))<Eps) || (fabs(fabs(X0-modf(X,&aux))-1.0)<Eps));
                        I++;     // Almacena el periodo de la orbita en la variable I
                  }
                if ((I>=MaxPer) && (!Found)) {
                  K++;
                  X0=X;   //Transitorio variable si no encontro una orbita periodica
                }
         } while ((!Found) && (K<Ciclos));
         if (Found) { // Comienza comprobacion de la resonancia hallada
                Found=0;
                if ((I>0) && (I<MaxPer)) {
                  X=modf(X,&aux);
                  X0=X;
                  aux=0.0;
                  for (J=0; J<I; J++) {                 //Ciclo de check
                        X=modf(X,&aux);
                        X=F(X,A,B)+aux;
                  }
                  Found=((fabs(X0-modf(X,&aux2))<=Eps) || (fabs(fabsl(X0-modf(X,&aux2))-1.0)<=Eps));
                  if (Found) { // Se encontro resonancia y paso el check
                        *Pr= (int) aux;               // Asigna numero de vueltas (envolvencia).
                        *Qr=I;                 // Asigna periodo almacenado (resonancia).
                        return(Found);
                  }
                }
         }
         X0=modf(X,&aux);        // Si no detecto periodicidad toma otro punto
         X=X0;                   // mas avanzado de una probable orbita y lo
                                 // utiliza al volver a entrar al loop.
         K++;
  } while ((!Found) && (K<Ciclos) && (I<MaxPer) && (J<MaxPer));

  if (!Found)   {            // No encontro resonancia en el punto
         *Pr=-1;
         *Qr=-1;
  }
  return(Found);
}
```