



Escuela  
Superior  
de Informática

UNIVERSIDAD DE CASTILLA-LA MANCHA  
ESCUELA SUPERIOR DE INFORMÁTICA

**Computadores Avanzados.**

4º GRADO EN INGENIERÍA INFORMÁTICA.

---

## **Práctica NCuerpos.**

---

*Autor:*

**Marcos López Sobrino y  
Alberto Salas Segúin.**

*Fecha:*

29 de diciembre de 2018

## Índice

1. Instrucciones para la ejecución de los programas.	2
2. Explicaciones de diseño.	4
3. Solución a la dificultad de la pagina 43.	5

## 1. Instrucciones para la ejecución de los programas.

Para la compilación y ejecución de los programas, se ha decidido realizar un *Makefile*, donde tenemos las siguientes instrucciones:

```
all: compile-par-rapido run-par-rapido
```

Mediante esta instrucción, primero se compila el programa paralelo rápido y seguidamente, una vez compilado, se ejecuta dicho programa.

```
compile-sec:
    gcc NCuerposSecuencial.c -o NCuerposSecuencial -lm -Wall
```

Con esta instrucción, se compila el programa secuencial.

```
run-sec:
    ./NCuerposSecuencial
```

A través de esta instrucción, el programa secuencial es ejecutado.

```
compile-par:
    mpicc NCuerposParalelo.c -o NCuerposParalelo -lm -Wall
```

Mediante instrucción se compila el programa paralelo básico.

Siguiendo la transparencia 30 del documento, si solo interesa el tiempo de ejecución, se ha incluido la directiva de compilación condicional *# ifndef NO\_SAL* en el programa, de modo que para únicamente obtener el tiempo de ejecución, en el terminal tenemos que introducir la instrucción *make compile-par-nosal*. Esta instrucción es:

```
compile-par-nosal:
    mpicc NCuerposParalelo.c -o NCuerposParalelo -lm -Wall -D NO_SAL
```

La ejecución del programa paralelo básico se hace con la siguiente instrucción.

```
run-par:
    mpirun -np 2 NCuerposParalelo
```

```
compile-par-rapido:
    mpicc NCuerposParalelo_AlgoritmoRapido.c -o
        NCuerposParalelo_AlgoritmoRapido -lm -Wall
```

Esta instrucción es utilizada para la compilación del programa paralelo rápido.

Al igual que antes, si únicamente nos interesa el tiempo de ejecución del programa, la instrucción a utilizar es la siguiente:

```
compile-par-rapido-nosal:
    mpicc NCuerposParalelo_AlgoritmoRapido.c -o
        NCuerposParalelo_AlgoritmoRapido -lm -Wall -D NO_SAL
```

Con la instrucción *run-par-rapido* se ejecuta el programa paralelo rápido.

```
run-par-rapido:
    mpirun -np 2 NCuerposParalelo_AlgoritmoRapido
```

## 2. Explicaciones de diseño.

En primer lugar, se han implementado 3 estructuras:

- **struct Datos.** Dicha estructura contiene las variables **n**, **tp**, **k**, **delta**, **u** necesarias para el algoritmo.
- **struct Masas.** En esta estructura se encuentra el **id** y la masa (**m**) de cada cuerpo.
- **struct Coord.** Donde almacenamos el **id**, la componente **x** y la componente **y** de cada uno de los cuerpos.

Para poder trabajar con estas estructuras se ha utilizado ***MPI\_Datatype***.

```
MPI_Datatype MPI_Datos;  
MPI_Datatype MPI_Masas;  
MPI_Datatype MPI_Coord;  
MPI_Datatype MPI_CNCR;
```

***MPI\_Datatype MPI\_CNCR*** se ha utilizado para la implementación del algoritmo con distribución cíclica como se dice en el documento a partir de la diapositiva 44. De este modo, el proceso 0 va a almacenar las posiciones de todos los cuerpos y se las va a enviar a todos los procesos mediante la primitiva ***MPI\_Scatter***.

Las operaciones que se han utilizado han sido las mencionadas en el enunciado y explicadas en el documento *Introducción a MPI*.

- Para el envío y recepción de las masas se ha utilizado la operación ***MPI\_Bcast***.
- Para el envío y recepción de las posiciones y velocidades desde el rank 0 a los demás, se ha optado por ***MPI\_Scatter*** como hemos mencionado anteriormente.
- Para sincronizar a todos los procesos ***MPI\_Barrier***.
- Para que el proceso 0, pueda imprimir los valores de los cuerpos y que se produzca una salida ordenada, primero debe recibir todos los datos, que son las posiciones, velocidades y aceleraciones de los cuerpos asignados a otros procesos. Para este fin, se ha usado ***MPI\_Gather***.

### 3. Solución a la dificultad de la pagina 43.

```

1  for(int fase = 1; fase < npr; fase++){
2
3      /* ---- Enviamos a destino/Recibimos desde fuente: p_anillo y
4         a_anillo ----*\
5
6         MPI_Sendrecv_replace(p_anillo, ncu, MPI_Coord, dst, 1, src, 1,
7                               MPI_COMM_WORLD, MPI_STATUS_IGNORE);
8         MPI_Sendrecv_replace(a_anillo, ncu, MPI_Coord, dst, 1, src, 1,
9                               MPI_COMM_WORLD, MPI_STATUS_IGNORE);
10
11     /* ---- Calculamos aceleraciones ---- *\
12
13     calcularAceleracion(fase);
14
15 }
16
17     /* ---- Enviamos a_anillo a destino y recibimos a_anillo de
18        fuente ----*\
19
20 MPI_Sendrecv_replace(a_anillo, ncu, MPI_Coord, dst, 1, src, 1,
21                       MPI_COMM_WORLD, MPI_STATUS_IGNORE);
22
23     /* ---- Sumamos aceleraciones ----*\
24
25 for(int i = 0; i < ncu; i++){
26     a_local[i].x += a_anillo[i].x;
27     a_local[i].y += a_anillo[i].y;
28 }

```

En el bucle de la línea 1 del código anterior, lo que se hace es que en cada fase, enviamos y recibimos *p\_anillo* y *a\_anillo*, es decir, cada proceso en su variable *p\_anillo* y *a\_anillo* tiene almacenadas las posiciones y velocidades de todos los cuerpos; en todos los procesos las mismas.

Cuando termina el bucle, volvemos a obtener todas las aceleraciones (calculadas en la última iteración) haciendo uso de ***MPI\_Sendrecv\_replace*** en la línea 16. Así, todos los procesos vuelven a tener todas las aceleraciones.

Por último, nos quedaría sumar las aceleraciones guardadas en *a\_anillo* a las locales. Esto se hace en el bucle de la línea 20. Al terminar, ya tendríamos las aceleraciones correctas de los cuerpos en *a\_local*.