

Sistemas Distribuidos y Paralelos

Ingeniería en Computación



El problema de los N cuerpos

Universidad Nacional de La Plata



Facultad de Informática



Agenda

2

I. El problema de los N Cuerpos

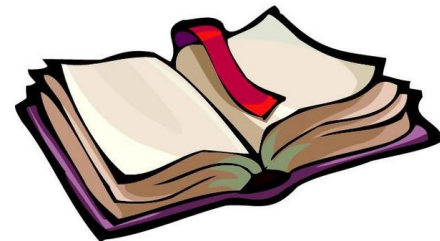
II. Algoritmo secuencial

- i. Calcular fuerzas
- ii. Mover cuerpos
- iii. Algoritmo

III. Algoritmos Paralelos

- i. Diseño
- ii. Solución con Memoria Compartida - SPMD
- iii. Solución con Memoria Distribuida
 - i. Master/Worker
 - ii. Algoritmo Sistólico
 - iii. Resumen

IV. Otros algoritmos



Agenda

3

I. El problema de los N Cuerpos

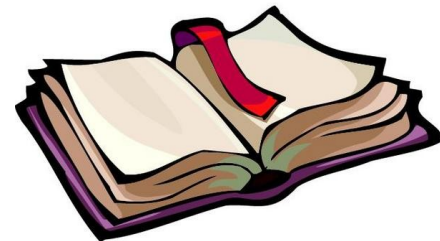
II. Algoritmo secuencial

- i. Calcular fuerzas
- ii. Mover cuerpos
- iii. Algoritmo

III. Algoritmos Paralelos

- i. Diseño
- ii. Solución con Memoria Compartida - SPMD
- iii. Solución con Memoria Distribuida
 - i. Master/Worker
 - ii. Algoritmo Sistólico
 - iii. Resumen

IV. Otros algoritmos

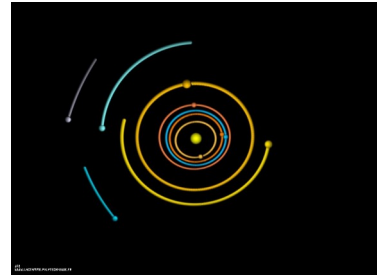


El problema de los N cuerpos

4



- El problema se conoce como Problema Gravitacional de los N Cuerpos.
- Simula la evolución en los movimientos de partículas (cuerpos) que interactúan según las leyes de la gravitación universal de Newton.
- Condiciones:
 - Cada cuerpo tiene *masa*, una posición (x,y) y una *velocidad*.
 - La gravedad causa que los cuerpos aceleren y se muevan.
 - El movimiento de un sistema de N Cuerpos se simula mediante pasos discretos de tiempo.
 - En cada paso se calcula la fuerza que actúa sobre cada cuerpo y se actualiza su velocidad y posición.



Agenda

5

I. El problema de los N Cuerpos

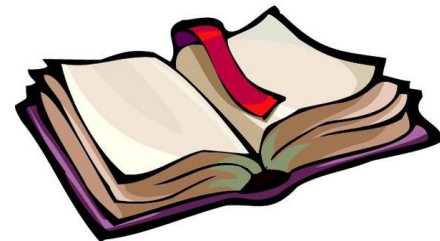
II. Algoritmo secuencial

- i. Calcular fuerzas
- ii. Mover cuerpos
- iii. Algoritmo

III. Algoritmos Paralelos

- i. Diseño
- ii. Solución con Memoria Compartida - SPMD
- iii. Solución con Memoria Distribuida
 - i. Master/Worker
 - ii. Algoritmo Sistólico
 - iii. Resumen

IV. Otros algoritmos



Agenda

6

I. El problema de los N Cuerpos

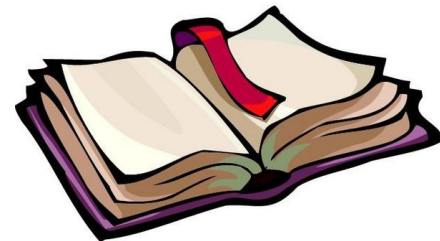
II. Algoritmo secuencial

- i. Calcular fuerzas
- ii. Mover cuerpos
- iii. Algoritmo

III. Algoritmos Paralelos

- i. Diseño
- ii. Solución con Memoria Compartida - SPMD
- iii. Solución con Memoria Distribuida
 - i. Master/Worker
 - ii. Algoritmo Sistólico
 - iii. Resumen

IV. Otros algoritmos



Algoritmo secuencial

7

- La solución secuencial a la simulación del problema de los N Cuerpos tiene la siguiente estructura:

```
for(paso = inicio to fin en pasos  $DT$ ) {  
    Calcular fuerzas  
    Mover los cuerpos  
}
```

- Donde DT es la longitud de cada paso de tiempo

Agenda

8

I. El problema de los N Cuerpos

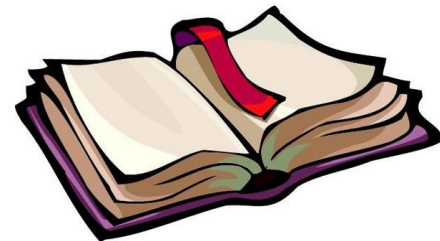
II. Algoritmo secuencial

- i. Calcular fuerzas
- ii. Mover cuerpos
- iii. Algoritmo

III. Algoritmos Paralelos

- i. Diseño
- ii. Solución con Memoria Compartida - SPMD
- iii. Solución con Memoria Distribuida
 - i. Master/Worker
 - ii. Algoritmo Sistólico
 - iii. Resumen

IV. Otros algoritmos



Algoritmo secuencial

Calcular fuerzas

9

- En física Newtoniana, la magnitud de la *fuerza* gravitacional entre dos cuerpos con masa m_1 y m_2 es:

$$F = G \cdot \frac{m_1 \cdot m_2}{r^2}$$

- F : módulo de la *fuerza* ejercida entre ambos cuerpos
- G : constante de gravitación universal $6,67 \cdot 10^{-11} \text{ Nm}^2 \text{ kg}^{-2}$
- r : Distancia entre los dos cuerpos.
 - Asumiendo un espacio bidimensional. Si las posiciones en el espacio de dos cuerpos m_1 y m_2 son (x_1, y_1) y (x_2, y_2) , respectivamente, consideramos la distancia Euclidiana:

$$r = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Ignoramos el problema en el cual los cuerpos están muy próximos (Colisión!!!) y el cálculo de la *fuerza* conduce a una inestabilidad numérica

(r^2 tiende a 0, F tiende a ∞)

$$\lim_{r \rightarrow 0} F = \lim_{r \rightarrow 0} G \cdot \frac{m_1 \cdot m_2}{r^2} = \infty$$

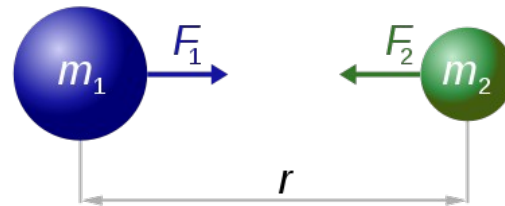


Algoritmo secuencial

Calcular fuerzas

10

- La dirección de la *fuerza* que el cuerpo m_1 ejerce sobre el cuerpo m_2 está dada por un vector en el sentido m_1 a m_2
- La dirección de la *fuerza* que el cuerpo m_2 ejerce sobre el cuerpo m_1 está dada por un vector en el sentido m_2 a m_1
- Las magnitudes de las *fuerzas* que ejerce cada cuerpo sobre el otro son las mismas, pero de sentido opuesto:



$$F_1 = F_2 = G \frac{m_1 \times m_2}{r^2}$$

Agenda

11

I. El problema de los N Cuerpos

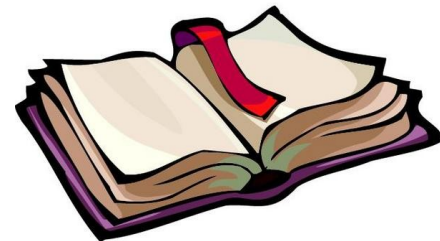
II. Algoritmo secuencial

- i. Calcular fuerzas
- ii. Mover cuerpos
- iii. Algoritmo

III. Algoritmos Paralelos

- i. Diseño
- ii. Solución con Memoria Compartida - SPMD
- iii. Solución con Memoria Distribuida
 - i. Master/Worker
 - ii. Algoritmo Sistólico
 - iii. Resumen

IV. Otros algoritmos

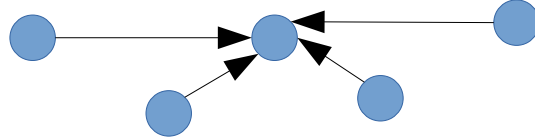


Algoritmo secuencial

Mover cuerpos

12

- La *fuerza* total que experimenta un cuerpo es la suma de las *fuerzas* ejercidas por cada uno de los otros cuerpos.



- Las *fuerzas* son vectores. Las sumas de vectores son conmutativas, los vectores de *fuerza* pueden sumarse en cualquier orden.
- Las *fuerzas* gravitacionales sobre un cuerpo causan que este acelere y se mueva.
- La relación entre *fuerza*, *masa* y *aceleración* se describe en la ecuación (**2^{da} Ley de Newton**):

$$F = m \cdot a$$

- La *aceleración* de un cuerpo m es la *fuerza* total ejercida sobre el cuerpo dividido su *masa*.

$$a = \frac{F}{m}$$

Algoritmo secuencial

Mover cuerpos

13

- Durante un intervalo pequeño dt , la *aceleración* (a) sobre un cuerpo m es aproximadamente constante, así que el cambio de *velocidad* (dv) es:

$$dv = a \cdot dt$$

- El cambio en la *posición* (dp) del cuerpo es la integral de su *velocidad* y la *aceleración* sobre el intervalo de tiempo dt , el cual es aproximadamente:

$$dp = v \cdot dt + \frac{a}{2} \cdot dt^2 = \left(v + \frac{dv}{2} \right) \cdot dt$$

- Esta fórmula emplea la estrategia leapfrog: la mitad del cambio en la *posición* se debe a la antigua *velocidad*, la otra mitad se debe a la nueva *velocidad*.

Agenda

14

I. El problema de los N Cuerpos

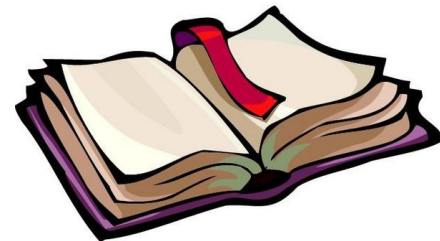
II. Algoritmo secuencial

- i. Calcular fuerzas
- ii. Mover cuerpos
- iii. Algoritmo

III. Algoritmos Paralelos

- i. Diseño
- ii. Solución con Memoria Compartida - SPMD
- iii. Solución con Memoria Distribuida
 - i. Master/Worker
 - ii. Algoritmo Sistólico
 - iii. Resumen

IV. Otros algoritmos



Algoritmo secuencial

Programa principal

15

- El algoritmo secuencial resuelve el problema con un orden $O(n^2)$:

```
#define G 6.673e-11
const unsigned bodies = ...;
const unsigned DT = ...;

typedef struct T3Dpoint{
    double x;
    double y;
} T3Dpoint_t

//Definiciones de variables compartidas
double m[bodies]; //Masa
T3Dpoint_t p[bodies], v[bodies], f[bodies]; //Posición, velocidad y fuerza, respectivamente
...
int main(int argc, char *argv[]){
    ...
    for(timeSteps = start; timeSteps <= finish; timeSteps += DT){
        calculateForces();
        moveBodies();
    }
}
```



Algoritmo secuencial

Función calculateForces y moveBodies

16

```
void calculateForces(){
double r, F; T3Dpoint_t direction;
for(i = 0; i < bodies - 1; i++){
    for(j = i+1; j < bodies; j++){
```

$r = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

$r = \text{sqrt}((p[j].x - p[i].x)^2 + (p[j].y - p[i].y)^2);$

$F = G \cdot (m[i] \cdot m[j]) / r^2;$

direction.x = p[j].x - p[i].x;
direction.y = p[j].y - p[i].y;

$F = G \cdot \frac{m_1 \cdot m_2}{r^2}$

$dp = (v + \frac{dv}{2}) \cdot dt$

f[i].x += F * direction.x/r;
f[j].x -= F * direction.x/r;
f[i].y += F * direction.y/r;
f[j].y -= F * direction.y/r;

```
    }
}
```

$$dv = a \cdot dt = \frac{F}{m} \cdot dt$$

$$r = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$F = G \cdot (m[i] \cdot m[j]) / r^2;$$

$$F = G \cdot \frac{m_1 \cdot m_2}{r^2}$$

$$dp = (v + \frac{dv}{2}) \cdot dt$$

```
void moveBodies(){
T3Dpoint dv, dp;
for(i = 0; i < bodies; i++){
    dv.x = (f[i].x / m[i]) * DT;
    dv.y = (f[i].y / m[i]) * DT;

    dp.x = (v[i].x + dv.x/2) * DT;
    dp.y = (v[i].y + dv.y/2) * DT;

    v[i].x += dv.x;
    v[i].y += dv.y;
    p[i].x += dp.x;
    p[i].y += dp.y;

    //Re-inicializa el vector de fuerza
    f[i].x = f[i].y = 0.0
}
```

Par (i,j) y (j,i) a la vez.

Las magnitudes de las fuerzas que ejerce cada cuerpo sobre el otro son las mismas, pero de sentido opuesto.



Leapfrog: la mitad del cambio en la posición se debe a la antigua velocidad y la otra mitad a la nueva velocidad.

Agenda

17

I. El problema de los N Cuerpos

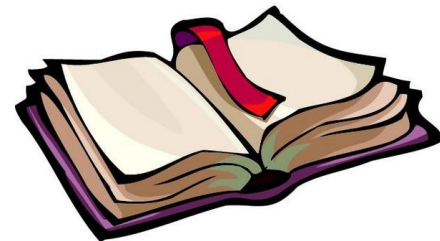
II. Algoritmo secuencial

- i. Calcular fuerzas
- ii. Mover cuerpos
- iii. Algoritmo

III. Algoritmos Paralelos

- i. Diseño
- ii. Solución con Memoria Compartida - SPMD
- iii. Solución con Memoria Distribuida
 - i. Master/Worker
 - ii. Algoritmo Sistólico
 - iii. Resumen

IV. Otros algoritmos



Agenda

18

I. El problema de los N Cuerpos

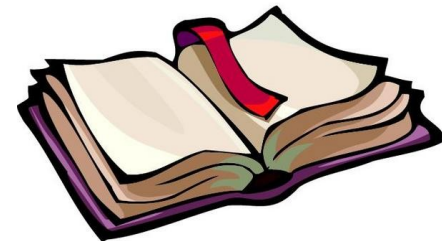
II. Algoritmo secuencial

- i. Calcular fuerzas
- ii. Mover cuerpos
- iii. Algoritmo

III. Algoritmos Paralelos

- i. Diseño
- ii. Solución con Memoria Compartida - SPMD
- iii. Solución con Memoria Distribuida
 - i. Master/Worker
 - ii. Algoritmo Sistólico
 - iii. Resumen

IV. Otros algoritmos



Algoritmos paralelos

19

- Paralelizamos el algoritmo secuencial a partir de las siguientes condiciones:
 - P unidades de procesamiento, un proceso o hilo por unidad de procesamiento
 - El número de cuerpos (**bodies**) es **mayor a P y múltiplo de P**

Paralelizar la función **moveBodies** es simple.

Distribuimos proporcionalmente el número de cuerpos entre P procesos.



Embarrassing Parallel Problem

```
void moveBodies() {  
    T3Dpoint dv, dp;  
    for(i = 0; i < bodies; i++) {  
        ...  
    }  
}
```

No podemos hacer lo mismo con la función **calculateForces**. Distribuir proporcionalmente los cuerpos lleva a **desbalance de carga**:

- El for anidado no es regular
- El primer proceso calcula la fuerza del primer cuerpo con los demás.
- El último proceso calcula la fuerza de sólo los dos últimos cuerpos.



```
void calculateForces() {  
    double r, F; T3Dpoint_t direction;  
    for(i = 0; i < bodies - 1; i++) {  
        for(j = i+1; j < bodies; j++) {  
            ...  
        }  
    }  
}
```

Agenda

20

I. El problema de los N Cuerpos

II. Algoritmo secuencial

- i. Calcular fuerzas
- ii. Mover cuerpos
- iii. Algoritmo

III. Algoritmos Paralelos

- i. Diseño
- ii. Solución con Memoria Compartida - SPMD
- iii. Solución con Memoria Distribuida
 - i. Master/Worker
 - ii. Algoritmo Sistólico
 - iii. Resumen

IV. Otros algoritmos



Algoritmos paralelos

Diseño

21



Dependencia entre ambas operaciones

Calculo de fuerzas

Movimiento de cuerpos (Embarrassing Parallel Problem)

Descomposición

Por datos de entrada.
Granularidad mas fina: una tarea es el cálculo de la fuerza entre un par de cuerpos.
Suponiendo 4 cuerpos (numerados de 0 a 3), el total de tareas es la combinación de 4 tomados de a 2:
 $(0,1) (0,2) (0,3) (1,2) (1,3) (2,3)$

Por datos de entrada.
Granularidad mas fina: una tarea es el movimiento de un cuerpo.

Comunicación

La analizaremos en cada solución propuesta.

Comunicación inexistente.

Aglomeración

La analizaremos en cada solución propuesta dependiendo de la granularidad elegida.



Suponiendo P unidades de procesamiento y C cuerpos, cada tarea será el movimiento de C/P cuerpos.

Mapeo

Podemos plantear un **mapeo estático** por partición de **tareas**, pero dado que hay desbalance de carga podemos utilizar un **mapeo dinámico**. De acuerdo a la solución propuesta emplearemos uno u otro según convenga.

Estático por partición de datos.

Dependencia entre ambas operaciones

El esfuerzo de diseño está en el **cálculo de fuerzas** ya que el movimiento es simple de diseñar y resolver.

Agenda

22

I. El problema de los N Cuerpos

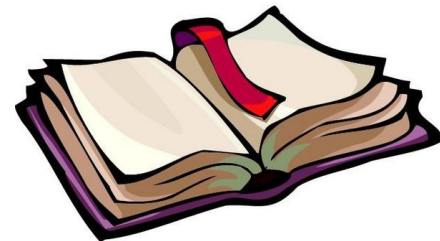
II. Algoritmo secuencial

- i. Calcular fuerzas
- ii. Mover cuerpos
- iii. Algoritmo

III. Algoritmos Paralelos

- i. Diseño
- ii. Solución con Memoria Compartida - SPMD
- iii. Solución con Memoria Distribuida
 - i. Master/Worker
 - ii. Algoritmo Sistólico
 - iii. Resumen

IV. Otros algoritmos



- **Descomposición:**

- Suponiendo el problema con 8 cuerpos (numerados de 0 a 7)
- Granularidad elegida: una tarea es el cómputo de los pares de un cuerpo

Granularidad mínima – Pares de fuerzas: combinación 8 tomados de a 2

(0,1) (0,2) (0,3) (0,4) (0,5) (0,6) (0,7) (1,2) (1,3) (1,4) (1,5) (1,6) (1,7) (2,3) (2,4) (2,5) (2,6) (2,7) (3,4) (3,5) (3,6) (3,7) (4,5) (4,6) (4,7) (5,6) (5,7) (6,7)



Granularidad elegida (Tarea: cálculo de un cuerpo)		
Tarea	Cuerpo	Pares de fuerza
0	0	{(0,1) (0,2) (0,3) (0,4) (0,5) (0,6) (0,7)}
1	1	{(1,2) (1,3) (1,4) (1,5) (1,6) (1,7)}
2	2	{(2,3) (2,4) (2,5) (2,6) (2,7)}
3	3	{(3,4) (3,5) (3,6) (3,7)}
4	4	{(4,5) (4,6) (4,7)}
5	5	{(5,6) (5,7)}
6	6	{(6,7)}
7	7	{}

- Debemos analizar como distribuir los cuerpos (tareas) entre los hilos o procesos de manera de balancear la carga de trabajo.
- Suponemos 2 unidades de procesamiento. Por lo tanto, tendremos 2 procesos o hilos P_0 y P_1 . Existen tres formas de asignar los cuerpos:

	Cuerpos							
	0	1	2	3	4	5	6	7
Proporcional	P_0	P_0	P_0	P_0	P_1	P_1	P_1	P_1
Stripes	P_0	P_1	P_0	P_1	P_0	P_1	P_0	P_1
Stripes reverso (competencias)	P_0	P_1	P_1	P_0	P_0	P_1	P_1	P_0

- Para comparar estas estrategias obtenemos la cantidad de pares de fuerzas que debe calcular cada proceso (**Aglomeración**).

Algoritmos paralelos

Solución con Memoria Compartida – SPMD - Estrategias

25



	P ₀			P ₁		
	Cuerpo	Pares	Nro Pares	Cuerpo	Pares	Nro Pares
Proporcional	0	{(0,1) (0,2) (0,3) (0,4) (0,5) (0,6) (0,7)}	7	4	{(4,5) (4,6) (4,7)}	3
	1	{(1,2) (1,3) (1,4) (1,5) (1,6) (1,7)}	6	5	{(5,6) (5,7)}	2
	2	{(2,3) (2,4) (2,5) (2,6) (2,7)}	5	6	{(6,7)}	1
	3	{(3,4) (3,5) (3,6) (3,7)}	4	7	{ }	0
	Total a calcular P ₀		22	Total a calcular P ₁		6
Stripes	0	{(0,1) (0,2) (0,3) (0,4) (0,5) (0,6) (0,7)}	7	1	{(1,2) (1,3) (1,4) (1,5) (1,6) (1,7)}	6
	2	{(2,3) (2,4) (2,5) (2,6) (2,7)}	5	5	{(5,6) (5,7)}	2
	4	{(4,5) (4,6) (4,7)}	3	3	{(3,4) (3,5) (3,6) (3,7)}	4
	6	{(6,7)}	1	7	{ }	0
	Total a calcular P ₀		16	Total a calcular P ₁		12
Stripes Reverso	0	{(0,1) (0,2) (0,3) (0,4) (0,5) (0,6) (0,7)}	7	1	{(1,2) (1,3) (1,4) (1,5) (1,6) (1,7)}	6
	3	{(3,4) (3,5) (3,6) (3,7)}	4	2	{(2,3) (2,4) (2,5) (2,6) (2,7)}	5
	4	{(4,5) (4,6) (4,7)}	3	5	{(5,6) (5,7)}	2
	7	{ }	0	6	{(6,7)}	1
	Total a calcular P ₀		14	Total a calcular P ₁		14

Algoritmos paralelos

Solución con Memoria Compartida - SPMD - Resumen

- Resumen:




	0	1	2	3	4	5	6	7	Carga P_0	Carga P_1
Proporcional	P_0	P_0	P_0	P_0	P_1	P_1	P_1	P_1	22	6
Stripes	P_0	P_1	P_0	P_1	P_0	P_1	P_0	P_1	16	12
Stripes reverso	P_0	P_1	P_1	P_0	P_0	P_1	P_1	P_0	14	14

- La estrategia Proporcional obtiene el peor balance de carga, la estrategia por Stripes lo mejora y la estrategia por Stripes Reverso es el mejor caso.
- La implementación por Stripes Reverso es muy costosa. Por lo tanto, implementamos la **estrategia de Stripes** que proporciona un balance de carga **aceptable**.
- La solución para 2 procesos o hilos es escalable a P procesos o hilos.
- A partir de la granularidad elegida obtenemos automáticamente un **mapeo estático de tareas** que asigna a cada proceso o hilo el conjunto de pares a calcular.

Algoritmos paralelos

Solución con Memoria Compartida – SPMD - Algoritmo

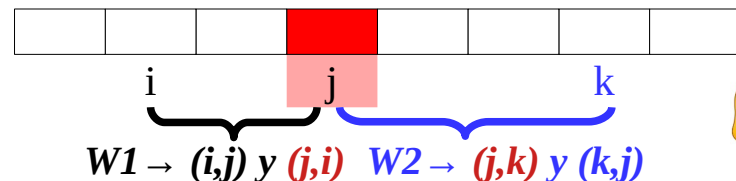
- El algoritmo paralelo sobre memoria compartida:

- Estrategia **por Stripes**
- P workers** (procesos o hilos)
- Paradigma SPMD 

```
Process worker [idW: 0 to P-1]{  
  for(timeSteps = start, timeSteps <= finish; timeSteps + DT){  
    calculateForces(idW);  
    moveBodies(idW);  
  }  
}
```

Cambios respecto al secuencial

- En la solución secuencial, `calculateForces()` resuelve los pares (i,j) y (j,i) a la vez.
- Si lo aplicamos a la solución paralela, se produce una **condición de carrera** cuando un worker calcula el par (i,j) y otro worker el par (j,k) .



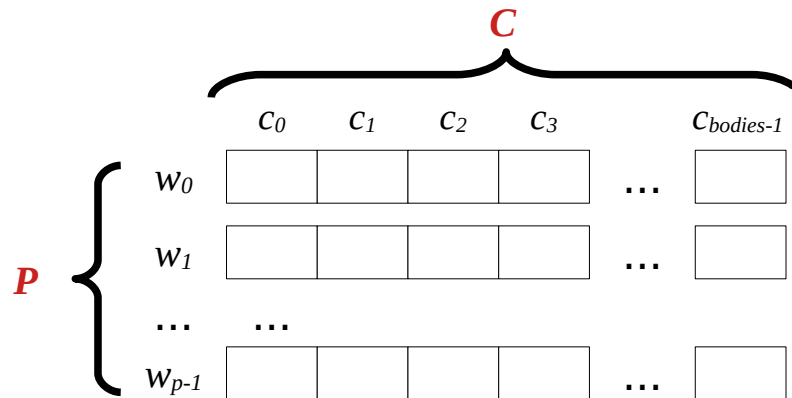
Soluciones:

- Sincronizar regiones críticas:** una región crítica por cuerpo (**costoso e ineficiente!!!**)
- Duplicar cómputo:** $w1$ calcula sólo (i,j) . $w2$ calcula (j,k) y (j,i) (**menor rendimiento!!!**)
- Utilizar una matriz de fuerzas:** usar **más memoria**

Algoritmos paralelos

Solución con Memoria Compartida – SPMD - Matriz de fuerzas

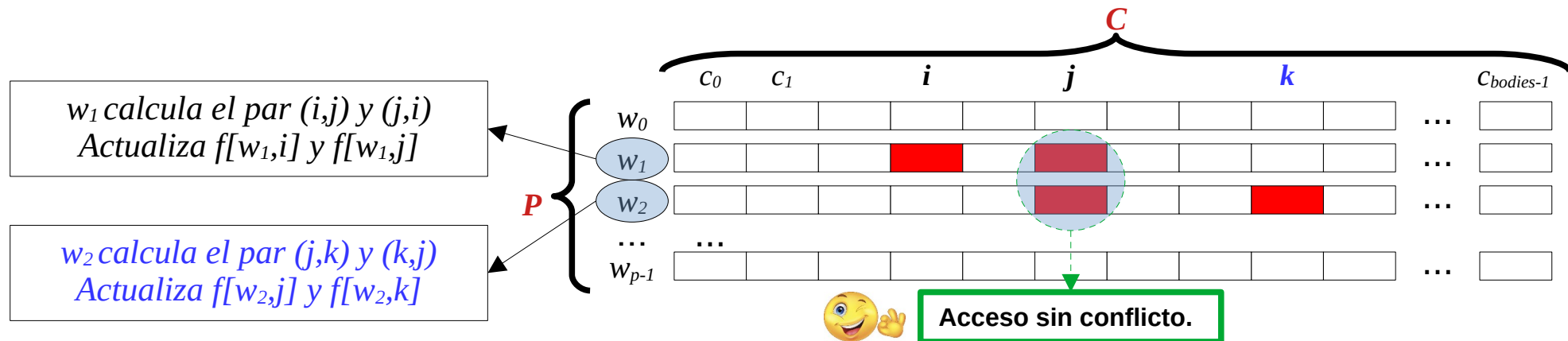
- Como buscamos mayor rendimiento implementamos el algoritmo paralelo utilizando una **matriz de fuerzas** y calculando los **pares (i,j) y (j,i) a la vez:**
- Las dimensiones de la matriz son $P \times C$:
 - Una fila por worker (P filas)
 - Cada fila del tamaño del número de cuerpos ($bodies=C$ columnas)



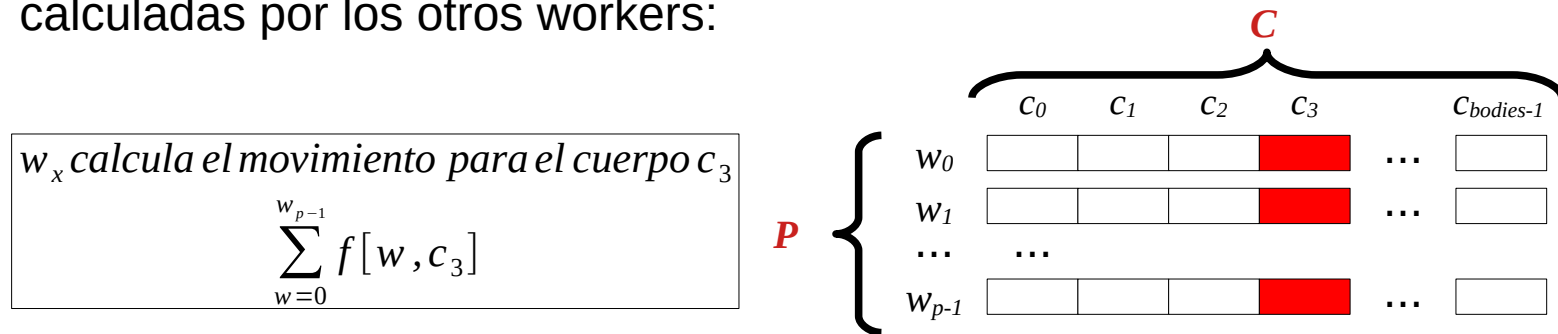
Algoritmos paralelos

Solución con Memoria Compartida – SPMD - Matriz de fuerzas

- Un worker **calcula la fuerza** entre (i,j) y (j,i) y actualiza las posiciones i y j de su fila:



- Otro worker **moverá** el cuerpo i . Para ello requiere sumar las fuerzas sobre el cuerpo i calculadas por los otros workers:



- **Dependencia de datos entre los workers (Comunicación):**



- Para mover los cuerpos es necesario que todos los workers hayan calculado las fuerzas.
- Para volver a calcular las fuerzas es necesario la nueva posición.

- El algoritmo requiere **barreras de sincronización** en cada etapa.

```
T3Dpoint m[bodies] p[bodies], v[bodies], f[P,bodies];  
  
Process worker [idW: 0 to P-1]{  
    for(timeSteps = start, timeSteps <= finish; timeSteps + DT){  
        calculateForces(idW);  
        barrier();  
        moveBodies(idW);  
        barrier();  
    }  
}
```

Cambios respecto al secuencial



Algoritmos paralelos

Solución con Memoria Compartida – SPMD - Funciones

31

```
void calculateForces(int idW){
double r, F; T3Dpoint direction;
    for(i = idW; i < bodies - 1; i+=P){
        for(j = i+1; j < bodies, j++){
            r = sqrt( (p[j].x - p[i].x)2 + (p[j].y - p[i].y)2 );
            F = G*(m[i]*m[j])/r2 ;

            direction.x = p[j].x - p[i].x;
            direction.y = p[j].y - p[i].y;

            f[idW,i].x += F * direction.x/r;
            f[idW,j].x -= F * direction.x/r;
            f[idW,i].y += F * direction.y/r;
            f[idW,j].y -= F * direction.y/r;

        }
    }
}
```

```
void moveBodies(int idW){
T3Dpoint dv, dp, force;
    force.x = force.y = 0.0;
    for(i = idW; i < bodies; i+=P){
        for(k=0; k<P; k++){
            force.x += f[k,i].x;
            force.y += f[k,i].y;

            f[k,i].x = f[k,i].y = 0.0;
        }
        dv.x = (force.x / m[i]) * DT;
        dv.y = (force.y / m[i]) * DT;

        dp.x = (v[i].x + dv.x/2) * DT;
        dp.y = (v[i].y + dv.y/2) * DT;

        v[i].x+= dv.x;
        v[i].y+= dv.y;

        p[i].x+= dp.x;
        p[i].y+= dp.y;

        force.x = force.y = 0.0
    }
}
```



Uso de la matriz de fuerzas

Cambios respecto al secuencial

Agenda

32

I. El problema de los N Cuerpos

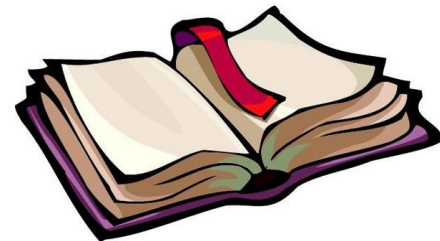
II. Algoritmo secuencial

- i. Calcular fuerzas
- ii. Mover cuerpos
- iii. Algoritmo

III. Algoritmos Paralelos

- i. Diseño
- ii. Solución con Memoria Compartida - SPMD
- iii. Solución con Memoria Distribuida
 - i. Master/Worker
 - ii. Algoritmo Sistólico
 - iii. Resumen

IV. Otros algoritmos



I. El problema de los N Cuerpos

II. Algoritmo secuencial

- i. Calcular fuerzas
- ii. Mover cuerpos
- iii. Algoritmo

III. Algoritmos Paralelos

- i. Diseño
- ii. Solución con Memoria Compartida - SPMD
- iii. Solución con Memoria Distribuida
 - i. Master/Worker
 - ii. Algoritmo Sistólico
 - iii. Resumen

IV. Otros algoritmos



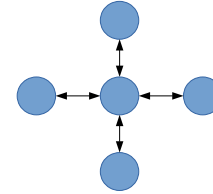
Algoritmos paralelos

Solución con Memoria Distribuida

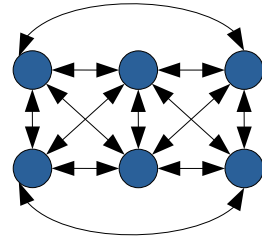
34

- Implementamos el algoritmo paralelo distribuido con **dos paradigmas**:

- **Master/Worker**: utilizando un **Mapeo dinámico centralizado**



- **Algoritmo sistólico**: utilizando un **Mapeo estático** basado en partición de **tareas**



- Dado que estamos en un modelo de memoria distribuida (envío y recepción de mensajes) debemos elegir la solución que minimice el overhead de comunicación.



Agenda

35

I. El problema de los N Cuerpos

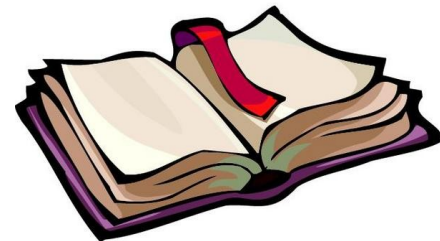
II. Algoritmo secuencial

- i. Calcular fuerzas
- ii. Mover cuerpos
- iii. Algoritmo

III. Algoritmos Paralelos

- i. Diseño
- ii. Solución con Memoria Compartida - SPMD
- iii. Solución con Memoria Distribuida
 - i. Master/Worker
 - ii. Algoritmo Sistólico
 - iii. Resumen

IV. Otros algoritmos

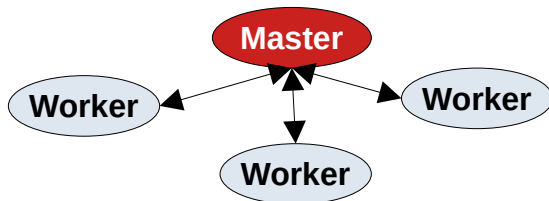


- Suponemos:

- Un proceso **Master** que no realiza trabajo, identifica las tareas (**descomposición**) determina la granularidad (**descomposición + aglomeración**) y las distribuye.
- **P workers** que inicialmente conocen todos los cuerpos, su posición inicial y su velocidad inicial.

- Dos etapas bien diferenciadas:

- **Calcular las fuerzas:** sensible al desbalance de carga. Para balancear la carga utilizamos un **mapeo dinámico centralizado**.
- **Mover los cuerpos:** no hay problemas de balance de carga. Realizamos un **mapeo estático de tareas**.



Algoritmos paralelos

Solución con Memoria Distribuida - Master/Worker

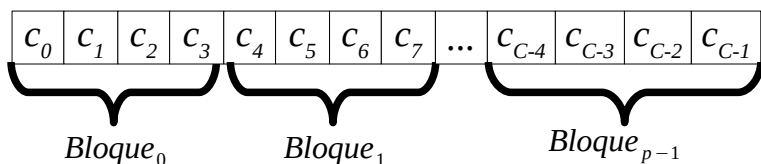
37

- Para el **cálculo de fuerzas**, podemos considerar dos tipos de **granularidad**:



- **Fina** (un par de cuerpos): **mucha comunicación** de mensajes pequeños.
- **Gruesa** (pares de bloques de cuerpos):

Dividir C cuerpos en P bloques de tamaño C/P



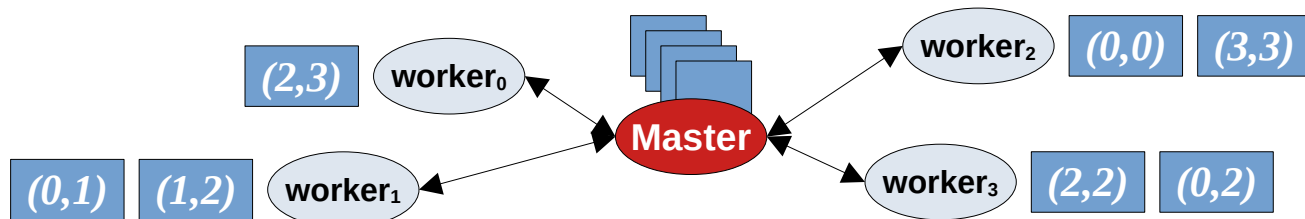
Armar pares (i,j) de combinaciones de bloques

$(B_0, B_0)(B_0, B_1)(B_0, B_2)(B_0, B_3)(B_1, B_1)(B_1, B_2)(B_1, B_3)(B_2, B_2) \dots$

Tareas de pares $(\text{Bloque}_i, \text{Bloque}_j) \rightarrow \text{Cantidad de tareas} = \text{Numero de pares} = \sum_{p=1}^P p = \frac{P \cdot (P+1)}{2}$

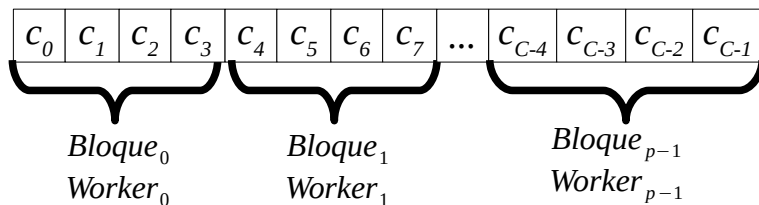
Una tarea es el **par (i,j)** que representa el cálculo de las **fuerzas** del **bloque i** con el **bloque j**

- Luego, los workers piden al Master tareas repetidamente, es decir piden pares (B_i, B_j) y calculan las fuerzas.

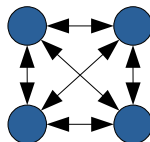


- Terminadas las tareas (cálculo de fuerzas), se hace un **mapeo estático**, uno a uno, entre los bloques y los Workers para realizar el movimiento de los cuerpos:

- Worker₀ mueve el Bloque₀
- Worker₁ mueve el Bloque₁
- ... Worker_{p-1} mueve el Bloque_{p-1}



- Cada worker necesita el vector de fuerzas de su bloque. Dos formas de intercambiarlos:
 - Centralizada mediante el Master:** **comunicación muy costosa**
 - Distribuida:** cada worker envía su vector de fuerzas a los otros workers
- Una vez que los workers realizaron los movimientos, intercambian las nuevas posiciones y velocidades con los otros workers para un nuevo ciclo (**comunicación**).



Algoritmos paralelos

Solución con Memoria Distribuida - Master/Worker

39

```
Process Master{  
  Declaración e inicialización de variables  
  for(timeSteps = start, timeSteps <= finish; timeSteps + DT){  
    Inicializar las tareas  
    for(i=0; i < numTasks + P; i++){  
      receive getTask(idW);  
      (block1,block2)=nextTask(tareas);  
      send workerTask[worker] (block1,block2);  
    }  
  }  
}
```

- (block1,block2) representa una tarea compuesta por ese par
- El límite del for es numTasks + P: es el número de tareas mas P veces indicando que no hay mas tareas
- Para indicar que no hay más tareas se puede enviar (-1,-1)



Algoritmos paralelos

Solución con Memoria Distribuida - Master/Worker

40

```
Process worker[idW:0 to P-1]{  
  T3Dpoint m[bodies] p[bodies], v[bodies], f[bodies]; // y otras variables;  
  for(timeSteps = start, timeSteps <= finish; timeSteps + DT){  
    1 solicitar tarea al master y calcular fuerzas  
    2 intercambiar fuerzas con otros workers y calcular movimientos de mi  
bloque  
    3 intercambiar cuerpos, velocidades y posiciones  
    4 reinicializar f a cero  
  }  
}
```



Algoritmos paralelos

Solución con Memoria Distribuida - Master/Worker

41

```
Process worker[idW:0 to P-1]{  
  T3Dpoint m[bodies] p[bodies], v[bodies], f[bodies]; // y otras variables;  
  for(timeSteps = start, timeSteps <= finish; timeSteps + DT){  
    //1 solicitar tarea al master y calcular fuerzas  
    while (hayaTareas){  
      send getTask(idW);  
      receive workerTask(block_i,block_j);  
      If (hay tarea) Calcular fuerzas entre los cuerpos de block_i y block_j  
    }  
    2 intercambiar fuerzas con otros workers y calcular movimientos de mi bloque  
    3 intercambiar cuerpos, velocidades y posiciones  
    4 reinicializar f a cero  
  }  
}
```



Algoritmos paralelos

Solución con Memoria Distribuida - Master/Worker

42

```
Process worker[idW:0 to P-1]{
  T3Dpoint m[bodies] p[bodies], v[bodies], f[bodies]; // y otras variables;
  for(timeSteps = start, timeSteps <= finish; timeSteps + DT){
    1 solicitar tarea al master y calcular fuerzas
    //2 intercambiar fuerzas con otros workers y calcular movimientos de mi bloque
    for(i=0, i<P, i++){
      if (i != idW) send forces[i](f);
    }
    for(i=0, i<P, i++){
      if (i != idW) receive forces[i](tf);
      sumar fuerzas de tf a f
    }
    actualizar p y v para mi bloque de cuerpos
    3 intercambiar cuerpos, velocidades y posiciones
    4 reinicializar f a cero
  }
}
```



```
Process worker[idW:0 to P-1]{  
  T3Dpoint m[bodies] p[bodies], v[bodies], f[bodies]; // y otras variables;  
  for(timeSteps = start, timeSteps <= finish; timeSteps + DT){  
    1 solicitar tarea al master y calcular fuerzas  
    2 intercambiar fuerzas con otros workers y calcular movimientos de mi bloque  
    //3 intercambiar cuerpos, velocidades y posiciones  
    for(i=0, i<P, i++){  
      if (i != idW) send bodies[i] (idW,p,v);  
    }  
    for(i=0, i<P, i++){  
      if (i != idW) receive bodies[i] (idW,tp,tv);  
      mover los cuerpos de tp y tv a p y v  
    }  
    4 reinicializar f a cero  
  }  
}
```



```
Process worker[idW:0 to P-1]{  
  T3Dpoint m[bodies] p[bodies], v[bodies], f[bodies]; // y otras variables;  
  for(timeSteps = start, timeSteps <= finish; timeSteps + DT){  
    1 solicitar tarea al master y calcular fuerzas  
    2 intercambiar fuerzas con otros workers y calcular movimientos de mi bloque  
    3 intercambiar cuerpos, velocidades y posiciones  
    //4 reinicializar f a cero  
    for(i = 0; i < bodies; i++){  
      f[i].x = f[i].y = 0.0;  
    }  
  }  
}
```



I. El problema de los N Cuerpos

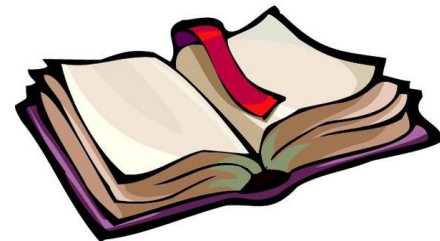
II. Algoritmo secuencial

- i. Calcular fuerzas
- ii. Mover cuerpos
- iii. Algoritmo

III. Algoritmos Paralelos

- i. Diseño
- ii. Solución con Memoria Compartida - SPMD
- iii. Solución con Memoria Distribuida
 - i. Master/Worker
 - ii. Algoritmo Sistólico
 - iii. Resumen

IV. Otros algoritmos



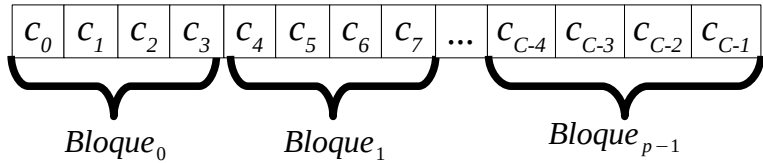
Algoritmos paralelos

Solución con Memoria Distribuida - Sistólico

46

- Descomponemos el problema siguiendo la estrategia del paradigma master/worker.
- Suponiendo P workers y C cuerpos:

Dividir C cuerpos en P bloques de tamaño C/P



Armar pares (i,j) de combinaciones de bloques

$(B_0, B_0)(B_0, B_1)(B_0, B_2)(B_0, B_3)(B_1, B_1)(B_1, B_2)(B_1, B_3)(B_2, B_2)...$

Tareas de pares $(Bloque_i, Bloque_j) \rightarrow$ Cantidad de tareas = Numero de pares = $\sum_{p=1}^P p = \frac{P \cdot (P+1)}{2}$

- Luego, realizamos un mapeo estático de pares de bloques en base a las estrategias de la implementación sobre memoria compartida:
 - Proporcional
 - Stripes
 - Stripes Reversos



$(B_0, B_0)(B_0, B_1)(B_0, B_2)(B_0, B_3)...$
 $(B_1, B_1)(B_1, B_2)(B_1, B_3)...$
 $(B_2, B_2)(B_2, B_3)...$
 $(B_3, B_3)...$
...

Algoritmos paralelos

Solución con Memoria Distribuida - Sistólico

47

- Si analizamos cada una de las estrategias nos encontramos con lo siguiente:
 - **Proporcional:**
 - Carga menos balanceada
 - Menos espacio de almacenamiento local por cada worker
 - Menos comunicación: mensajes mas cortos y la mitad que otras estrategias
 - **Stripes o Stripes reversos:**
 - Carga más balanceada
 - Más espacio de almacenamiento: cada worker necesita tener su propia copia de los vectores de posición, velocidad, fuerza y masa
 - Más comunicación: cada worker debe intercambiar vectores completos, esto es un gran número de mensajes muy grandes



Seguimos una estrategia **Proporcional**.
El desbalance puede verse compensado al minimizar las comunicaciones.

Algoritmos paralelos

Solución con Memoria Distribuida - Sistólico

48

- Suponiendo 4 Workers ($P=4$) y C cuerpos:
 - 4 bloques de $C/4$ cuerpos cada uno.
 - Granularidad elegida (Aglomeración): una tarea es el cómputo de todos los pares de un bloque i
 - $((B_i, B_i), (B_i, B_{i+1}), (B_i, B_{i+2}) \dots (B_i, B_{P-1}))$
 - Cada worker se encarga de una tarea (Mapeo):
- Inicialmente, cada worker posee solo los cuerpos que corresponden a su bloque.



Tarea	Bloque	Pares de bloques	Worker
0	B_0	$\{ (B_0, B_0)(B_0, B_1)(B_0, B_2)(B_0, B_3) \}$	$Worker_0$
1	B_1	$\{ (B_1, B_1)(B_1, B_2)(B_1, B_3) \}$	$Worker_1$
2	B_2	$\{ (B_2, B_2)(B_2, B_3) \}$	$Worker_2$
3	B_3	$\{ (B_3, B_3) \}$	$Worker_3$

Algoritmos paralelos

Solución con Memoria Distribuida - Sistólico

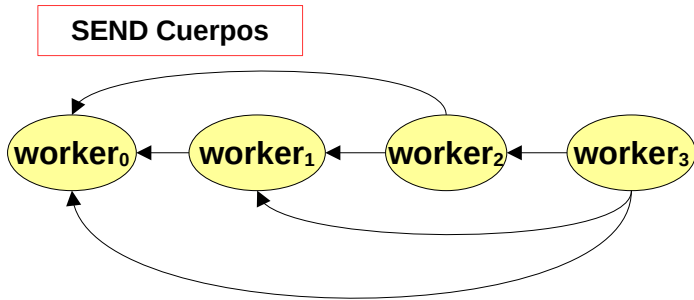
49

- Comunicación en 3 etapas:



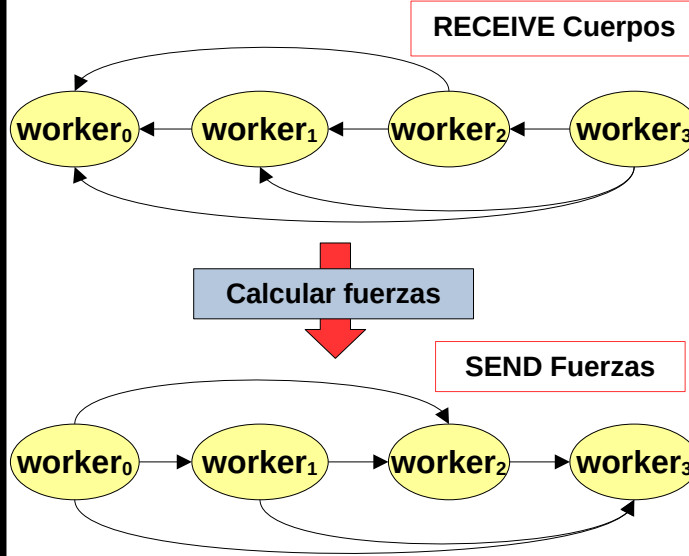
Etapas 1

Los workers envían sus cuerpos a los workers con menor idW.



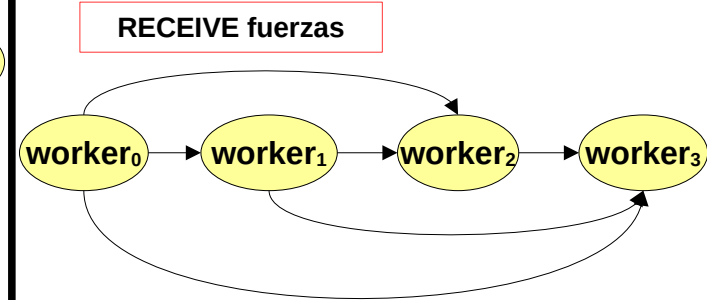
Etapas 2

Reciben los cuerpos de los workers con mayor idW, calculan las fuerzas de su bloque con lo recibido y la envían.



Etapas 3

Reciben las fuerzas de los workers con menor idW y actualizan la velocidad y posición.



Algoritmos paralelos

Solución con Memoria Distribuida - Sistólico



- **El worker₀ :**

- Se encarga de los 4 primeros pares de bloques $\{ (B_0, B_0)(B_0, B_1)(B_0, B_2)(B_0, B_3) \}$
- Necesita las posiciones y velocidades para los cuerpos en los bloques B_1 , B_2 y B_3 , y necesita enviar a otros workers las fuerzas calculadas de estos bloques
- Pero no necesita enviar a otros workers la posición o velocidad de sus propios cuerpos y tampoco necesita recibir información de fuerza de ningún otro worker

- **El worker₃ :**

- Se encarga sólo del par (B_3, B_3)
- No necesita enviar las fuerzas a ningún otro worker y tampoco recibir la posición y velocidad de ningún otro cuerpo

- **Workers intermedios:**

- Deben enviar las fuerzas a otros workers y recibir las nuevas posiciones y velocidades.



- A tener en cuenta:
 - Si los bloques tienen todos el mismo tamaño la carga estará desbalanceada pero podría compensar cualquier aumento en el número y tamaño de mensajes.
 - Podemos utilizar bloques de tamaño variable y de esa forma lograr mayor balance.
 - Las partes del código donde hay pasaje de mensajes podrían no ser simétricas: workers diferentes envían diferente números de mensajes y lo hacen en diferentes momentos:
 - Ventaja: podemos superponer cómputo con comunicación (**overlapping**). En particular, cada worker envía mensajes y hace cómputo local antes de recibir mensajes y procesarlos.

Algoritmos paralelos

Solución con Memoria Distribuida - Sistólico

52

```
Process worker[idW:0 to P-1]{  
  int blockSize // Tamaño de mi bloque de cuerpos  
  int tempSize = número máximo de otros cuerpos en mensajes  
  T3Dpoint m[bodies] p[blockSize], v[blockSize], f[blockSize];  
  T3Dpoint tp[tempSize], tv[tempSize], tf[tempSize];  
  for(timeSteps = start, timeSteps <= finish; timeSteps + DT){  
    1 enviar mis cuerpos a workers con menor idW y calcular fuerzas para mi bloque  
    2 recibir cuerpos de workers con mayor idW y calcular fuerzas entre  
    lo recibido y mi bloque. Luego, enviárselas a estos nuevamente  
    3 recibir las fuerzas de los workers con menor idW. Actualizar p y v.  
    4 reinicializar f a cero  
  }  
}
```



Algoritmos paralelos

Solución con Memoria Distribuida - Sistólico

53

```
Process worker[idW:0 to P-1]{  
  int blockSize // Tamaño de mi bloque de cuerpos  
  int tempSize = número máximo de otros cuerpos en mensajes  
  T3Dpoint m[bodies] p[blockSize], v[blockSize], f[blockSize];  
  T3Dpoint tp[tempSize], tv[tempSize], tf[tempSize];  
  for(timeSteps = start, timeSteps <= finish; timeSteps + DT){  
    1 enviar mis cuerpos a workers con menor idW y calcular fuerzas para mi bloque  
    for(i=0; i<idW-1, i++){  
      send bodies[i] (w,p,v);  
      calculo f para mi bloque de cuerpos  
    }  
    2 recibir cuerpos de workers con mayor idW y calcular fuerzas entre  
    lo recibido y mi bloque. Luego, enviárselas a estos nuevamente  
    3 recibir las fuerzas de los workers con menor idW. Actualizar p y v.  
    4 reinicializar f a cero  
  }  
}
```



Algoritmos paralelos

Solución con Memoria Distribuida - Sistólico

54

```
Process worker[idW:0 to P-1]{
  int blockSize // Tamaño de mi bloque de cuerpos
  int tempSize = número máximo de otros cuerpos en mensajes
  T3Dpoint m[bodies] p[blockSize], v[blockSize], f[blockSize];
  T3Dpoint tp[tempSize], tv[tempSize], tf[tempSize];
  for(timeSteps = start, timeSteps <= finish; timeSteps + DT){
    1 enviar mis cuerpos a workers con menor idW y calcular fuerzas para mi bloque
    2 recibir cuerpos de workers con mayor idW, calcular fuerzas entre
    lo recibido y mi bloque. Luego, enviárselas a estos nuevamente
    for(i=idW+1, i<P, i++){
      receive bodies[i] (otherWorker, tp, tv);
      calculo fuerzas entre mi bloque y el bloque de otherWorker. Las guardo en tf
      send forces[otherWorker] (tf)
    }
    3 recibir las fuerzas de los workers con menor idW. Actualizar p y v.
    4 reinicializar f a cero
  }
}
```



Algoritmos paralelos

Solución con Memoria Distribuida - Sistólico

55

```
Process worker[idW:0 to P-1]{
  int blockSize // Tamaño de mi bloque de cuerpos
  int tempSize = número máximo de otros cuerpos en mensajes
  T3Dpoint m[bodies] p[blockSize], v[blockSize], f[blockSize];
  T3Dpoint tp[tempSize], tv[tempSize], tf[tempSize];
  for(timeSteps = start, timeSteps <= finish; timeSteps + DT){
    1 enviar mis cuerpos a workers menor idW y calcular fuerzas para mi bloque
    2 recibir cuerpos de workers con mayor idW, calcular fuerzas entre
    lo recibido y mi bloque. Luego, enviárselas a estos nuevamente
    3 recibir las fuerzas de los workers con menor idW. Actualizar p y v
    for(i=0; i<idW-1, i++){
      receive forces[i](tf);
      combinar tf con f
    }
    actualizar p y v para mis cuerpos
    4 reinicializar f a cero
  }
}
```



```
Process worker[idW:0 to P-1]{
  int blockSize // Tamaño de mi bloque de cuerpos
  int tempSize = número máximo de otros cuerpos en mensajes
  T3Dpoint m[bodies] p[blockSize], v[blockSize], f[blockSize];
  T3Dpoint tp[tempSize], tv[tempSize], tf[tempSize];
  for(timeSteps = start, timeSteps <= finish; timeSteps + DT){
    1 enviar mis cuerpos a workers con menor idW y calcular fuerzas para mi bloque
    2 recibir cuerpos de workers con mayor idW, calcular fuerzas entre
    lo recibido y mi bloque. Luego, enviárselas a estos nuevamente
    3 recibir las fuerzas de los workers con menor < idW. Actualizar p y v
    4 reinicializar f a cero
    for(i = 0; i < blockSize; i++)
      f[i].x = f[i].y = 0.0;
  }
}
```



Agenda

57

I. El problema de los N Cuerpos

II. Algoritmo secuencial

- i. Calcular fuerzas
- ii. Mover cuerpos
- iii. Algoritmo

III. Algoritmos Paralelos

- i. Diseño
- ii. Solución con Memoria Compartida - SPMD
- iii. Solución con Memoria Distribuida
 - i. Master/Worker
 - ii. Algoritmo Sistólico
 - iii. Resumen

IV. Otros algoritmos



Algoritmos paralelos

Solución con Memoria Distribuida - Resumen

58

Paradigma	Mensajes	Número de Mensajes	Cuerpos por mensaje
Master/Workers	Solicitar tasks	$2 * (numTask + P)$	2
	Intercambiar fuerzas	$P * (P - 1)$	n
	Intercambiar cuerpos	$P * (P - 1)$	$2*n$
Algoritmo sistólico	Intercambiar cuerpos	$P * (P - 1)/2$	$2*n$
	Intercambiar fuerzas	$P * (P - 1)/2$	n



La elección de una u otra estrategia dependerá de las arquitecturas y la red de comunicación utilizada.
Sólo podemos determinar cuál es la solución más adecuada a través de la **experimentación**.

I. El problema de los N Cuerpos

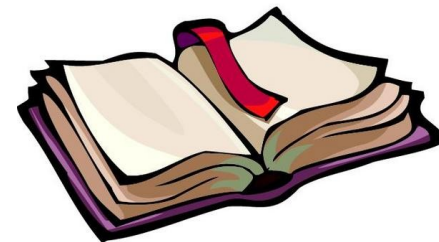
II. Algoritmo secuencial

- i. Calcular fuerzas
- ii. Mover cuerpos
- iii. Algoritmo

III. Algoritmos Paralelos

- i. Diseño
- ii. Solución con Memoria Compartida - SPMD
- iii. Solución con Memoria Distribuida
 - i. Master/Worker
 - ii. Algoritmo Sistólico
 - iii. Resumen

IV. Otros algoritmos



Otros algoritmos

60

- La parte dominante del algoritmo es el cálculo de fuerzas de orden $O(n^2)$ en cada paso. Otras soluciones intentan reducir ese orden mediante la siguiente condición:
 - La fuerza entre dos cuerpos suficientemente lejos es insignificante
 - Un grupo de cuerpos muy cercanos puede considerarse como un único cuerpo con respecto a otros muy lejanos
 - Algoritmos complejos que siguen esta idea, descartan las estructuras lineales organizando los cuerpos por proximidad en árboles:
 - **Barnes Hut** $O(n \log n)$
 - **Faster Multipole Method** $O(n)$

