



Programación Paralela - CC332

2021-I

José Fiestas

21 de mayo de 2021

Universidad Nacional de Ingeniería
jose.fiestas@uni.edu.pe

Unidad 3. Descomposición en paralelo

Objetivos:

1. Paralelismo directo
2. Uso de random en paralelo
3. Particionamiento, Divide y Vencerás

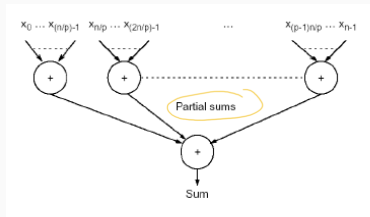
Particionamiento, Divide y Vencerás

Particionamiento

Las tareas se dividen
y se ejecutan independientemente.

Esto se aplica:

- A la data (data partitioning, domain decomposition)
- A las funciones del programa (functional decomposition). Aquí también la comunicación es lo mas costoso



Ejemplo: suma de elementos de un array

El cálculo secuencial necesita $n-1$ adiciones, con complejidad $O(n)$.

Con p procesos se obtiene:

- Envío de data $t_{comm_1} = p(t_{startup} + (n/p)t_{data})$
- Cálculo de cada proceso $t_{comp_1} = n/(p - 1)$
- Recopilación de sumas parciales $t_{comm_2} = t_{startup} + pt_{data}$
- Suma total $t_{comp_2} = p - 1$

Complejidad es $t_p = O(n)$

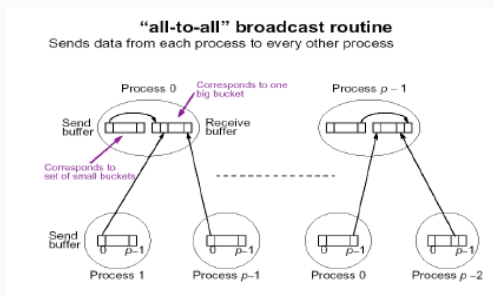
Ejemplo: Bucket Sort

Utiliza un método de **particionamiento**. Funciona mejor con conjuntos homogéneos. Un intervalo a se separa en m regiones (buckets), con n/m valores en cada una. Luego son los grupos en cada bucket ordenados (con quicksort o mergesort), en **$O((n/m) \log(n/m))$** y los resultados concatenados en una lista ordenada.

Tiempo secuencial: $t_s = n + m(\frac{n}{m} \log(\frac{n}{m})) = O(n \log(\frac{n}{m}))$

Ejemplo: Bucket Sort en paralelo

Cada bucket pertenece a un proceso, con lo que se logra $O((n/p) \log(n/p))$, con $p=m$ procesos. Pero cada proceso debe controlar todos los números no-ordenados para su selección. Si se separa la lista en p regiones, cada una para cada proceso, esta se separa en cada bucket y se envía luego a cada otro proceso, para ser ordenada finalmente ahí. Esto implica una comunicación all-to-all



Ejemplo: Bucket Sort en paralelo

Primero se envía la data a los procesos y estos la clasifican. De ahí, los pasos son:

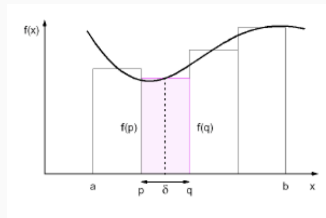
- Comunicación, $t_{comm_1} = t_{startup} + nt_{data}$
- 1.cálculo, n/p números en p buckets pequeños, $t_{comp_2} = \frac{n}{p}$
- Comunicación, $p-1$ buckets son enviados a $p-1$ procesos,
 $t_{comm_3} = p(p-1)(t_{startup} + \frac{n}{p^2} t_{data})$
- 2.cálculo, los n/p números en los buckets grandes se ordenan,
 $t_{comp_4} = \frac{n}{p} \log(\frac{n}{p})$

Ejemplo: Integración numérica

Para el cálculo de una integral de la forma $I = \int_a^b f(x)dx$ se puede dividir el intervalo a-b en partes, tal que cada una se aproxima con un rectángulo, asignado a cada proceso.

Método estático:

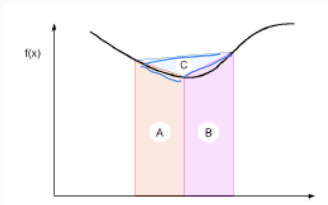
En el método del trapecio, cada área se calcula con la fórmula $\frac{1}{2}(f(p) + f(q))\delta$. Si se conoce δ de antemano, se comunica a cada proceso. Con p procesos, es $(b-a)/p$



Ejemplo: Integración numérica

Cuadratura adaptativa:

Si no se conoce δ de antemano, se calcula hasta que se logre la convergencia deseada. δ se divide en 3 zonas, y se divide hasta que C sea lo suficientemente pequeño

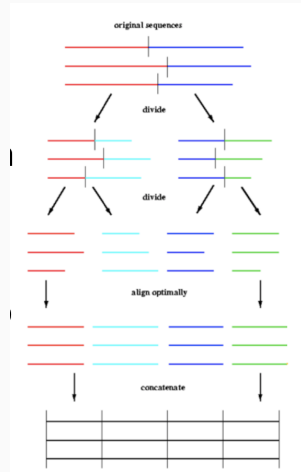


Paradigma Divide y vencerás (divide and conquer)

Algoritmo heurístico

acelerado para solución de secuencias múltiples y homólogas. Funciona:

- Separando las secuencias en partes reduciendo su longitud. Esto es, dividiendo el problema en sub-problemas
- Optimizar su distribución o resolver los sub-problemas (recursiva, directamente)
- Concatenar resultados o combinar soluciones de sub-problemas en solución general



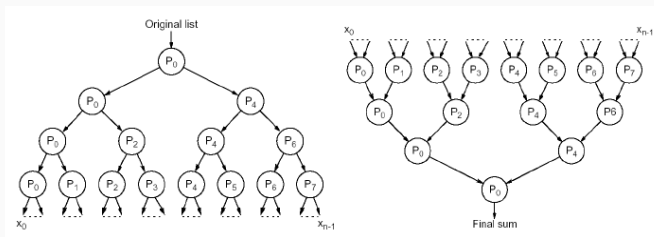
Paradigma Divide y vencerás (divide and conquer)

Se utiliza para operaciones globales en listas (ordenamiento), cálculo de máximo/mínimo, o búsqueda

En su versión recursiva, se generan dos sub-problemas en cada separación. Esto se representa con un árbol, que se divide para abajo y se recopila para arriba.

Paradigma Divide y vencerás (divide and conquer)

Paralelismo: Partes distintas del árbol pueden ser ejecutadas a la vez. Una solución sería asignar un proceso a cada nodo del árbol. Pero sería ineficiente, ya que para 2^m sub-problemas se necesitan $2^{m+1} - 1$ procesos



Paradigma Divide y vencerás (divide and conquer)

Paralelismo:

Sea la data n una potencia de 2 y p procesos, para la primera etapa (división) se necesitan $\log(p)$ pasos

$$t_{comm_1} = \frac{n}{2}t_{data} + \frac{n}{4}t_{data} + \dots + \frac{n}{p}t_{data} = \frac{n(p-1)}{p}t_{data}$$

Mientras que en la combinación de resultados

$$t_{comm_2} = (\log(p))t_{data}$$

Es decir, $t_{comm} = O(n)$

El cálculo suma n/p números, $t_{comp} = \frac{n}{p} + \log(p) = O(n)$

La eficiencia máxima es p , si todos los procesos calculan su suma parcial.

Ejemplo: Ordenamiento

Paralelismo:

Sea la data n una potencia de 2 y p procesos, para la primera etapa (división) se necesitan $\log(p)$ pasos

$$t_{comm_1} = \frac{n}{2}t_{data} + \frac{n}{4}t_{data} + \dots + \frac{n}{p}t_{data} = \frac{n(p-1)}{p}t_{data}$$

Mientras que en la combinación de resultados

$$t_{comm_2} = (\log(p))t_{data}$$

Es decir, $t_{comm} = O(n)$

El cálculo suma n/p números, $t_{comp} = \frac{n}{p} + \log(p) = O(n)$

La eficiencia máxima es p , si todos los procesos calculan su suma parcial.

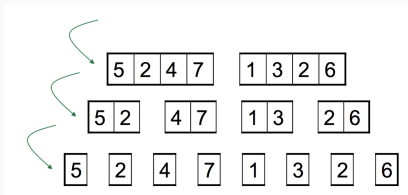
Ejemplo: Ordenamiento

Dado un array, **objetivo** es ordenar el array

5	2	4	7	1	3	2	6
---	---	---	---	---	---	---	---

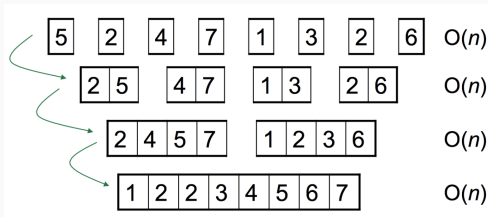
1	2	2	3	4	5	6	7
---	---	---	---	---	---	---	---

Paso 1: dividir el array. Necesita $\log(n)$ divisiones para lograr arrays de un elemento



Ejemplo: Ordenamiento

Paso 2: Conquista, requiere $\log(n)$ iteraciones, cada una tomando tiempo $O(n)$. $T(n) = O(n \log n)$

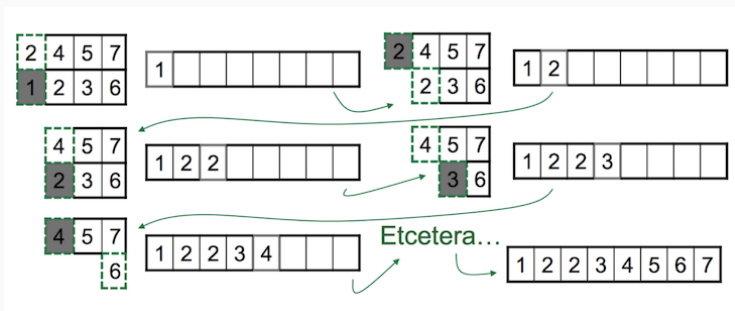


Paso 3: Combinar. Para 2 arrays de tamaño 1 la union es directa. En general, 2 arrays ordenadas de tamaño n y m pueden ser combinadas en un tiempo $O(n+m)$ para formar un array $n+m$ ordenada.

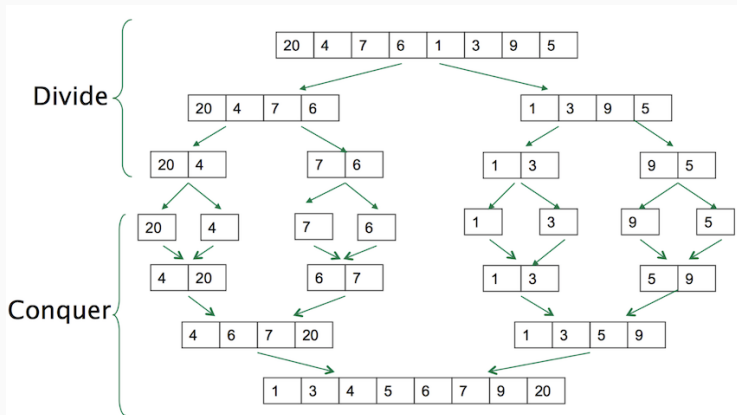


Ejemplo: Ordenamiento

Caso: Combinar dos arrays de tamaño 4



Ordenamiento: Mergesort



Algoritmo:

mergesort(arr[],l,r)

If $r > l$

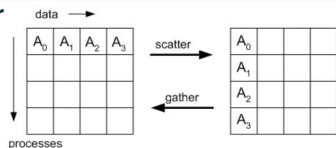
1. Encuentre el punto de división del array en dos mitades: **middle m = $(l+r)/2$**
2. Llama rmergeSort para la primera mitad: **call mergesort(arr, l, m)**
3. Llamar mergeSort para la segunda mitad: **call mergeSort(arr, m+1, r)**
4. Combinar las dos mitades ordenadas en 2 y 3: **call merge(arr, l, m, r)**

Complejidad:

- Para la combinación la complejidad es $O(n)$
- Numero de iteraciones es $O(\log n)$
- Tiempo total es $O(n \log n)$
- Se usa para ordenar listas concatenadas (linked lists) en tiempo $O(n \log n)$

Multiplicacion Matriz-Vector

- Matriz se distribuye en filas
- producto escalar se realiza en cada proceso
- MPI_Gather se usa para recolectar los productos



$$\begin{matrix} & A & * & b & = & c \\ \left[\begin{array}{c} \text{Process 0} \\ \text{-----} \\ \text{Process 1} \\ \text{-----} \\ \text{Process 2} \\ \text{-----} \\ \text{Process 3} \end{array} \right] & \left[\begin{array}{c} \\ \\ \\ \end{array} \right] & & = & \left[\begin{array}{c} 0 \\ \text{-----} \\ 1 \\ \text{-----} \\ 2 \\ \text{-----} \\ 3 \end{array} \right] \end{matrix}$$

MPI_Gather recolecta resultados de cada proceso y las une en el vector resultante

```
double a[25,100], b[100], cpart[25], ctotat[100];
int root;
root=0;
for(i=0; i<25; i++)
{
    cpart[i]=0;
    for(k=0; k<100; k++)
        cpart[i]=cpart[i]+a[i,k]*b[k];
}
MPI_Gather(cpart, 25, MPI_REAL, ctotat, 25, MPI_REAL, root, MPI_COMM_WORLD);
```

Cálculo de fuerzas entre cuerpos, aplicado a problemas de astrofísica o dinámica molecular.

Gravitación en N-cuerpos

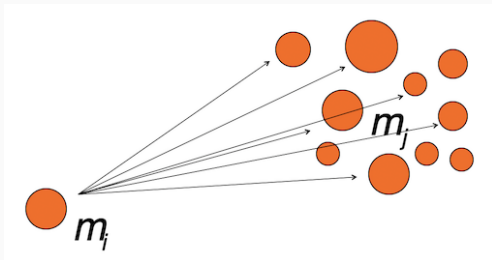
Se trata de calcular las fuerzas gravitatorias de cuerpos en el espacio (planetas, estrellas, galaxias), con lo que se obtendrán posiciones y velocidades de los cuerpos en un tiempo determinado.

Problema de N-cuerpos

No hay solución analítica desde $N=3$

El cálculo es de la fuerza ejercida sobre el cuerpo i

$$F_i = -Gm_i \sum_{1 \leq j \leq N, j \neq i} \frac{m_j r_{ij}}{|r_{ij}|^3}$$



Número de operaciones de cálculo de fuerza es $N(N-1)/2$

Problema de N-cuerpos

Para una simulación se puede solo aproximar el movimiento en intervalos pequeños. Sea el intervalo Δt , es para $t+1$

$F = \frac{mv_{t+1} - v_t}{\Delta t}$, y con ello, para la velocidad y la posición:

$$v_{t+1} = v_t + \frac{F\Delta t}{m}$$

$$x_{t+1} = x_t + v\Delta t$$

Ya que el cuerpo se mueve a una nueva posición, se modifican su aceleración y fuerza, que deberá volver a calcularse.

Nota: Ya que la velocidad no es constante en Δt es solo una aproximación. Por ello se utilizan métodos de interpolación en el tiempo como 'leap-frog'

Problema de N-cuerpos: código secuencial

```
class Nbody{
public:
float pos[3][N];
float vel[3][N];
float m[N];
}
int main(int arg, char**argv){
...
// define class galaxy
Nbody galaxy;
...
// initialize properties
galaxy.init();
// integrate forces

galaxy.integr();
return 0;
}

void integr() {
...
// medir CPU time
start=clock();
force(n,pos,vel,m,dt);
// medir CPU time
end=clock();
cpuTime=difftime(end,start)/(CLOCKS\ _PER
...
}
```

Problema de N-cuerpos: código secuencial

```
void force(int n, float pos[][],float vel[][], float m[], float dt) {
    ...
    // suma en i
    for (int i=0;i<n;i++)
    {
        float my_r_x=r_x[i];
        // suma en j
        for (int j=0;j<n;j++)
        {
            if(j!=i) // evitar i=j
            {
                //calcular aceleracion
                float d=r_x[j]-my_r_x; // 1 FLOP
                a_x+= G*m[j]/(d*d); // 4 FLOPS
                ...
            }
        }
    }
    // actualizar velocidades
    v_x[i] += a_x*dt; // 2 FLOPS
    // actualizar posiciones
    r_x[i]+=v_x[i]*dt; // 2 FLOPS
}
}
```

Problema de N-cuerpos en paralelo

Se puede realizar con partición directa, donde cada proceso se encarga de un grupo de cuerpos y las fuerzas son comunicadas con los otros procesos. La complejidad es $O(n^2)$, ya que cada cuerpo es afectado por los $n-1$ restantes. Esto no es eficiente.

Por ello, se considera que un cluster de cuerpos distantes afecta aproximadamente como un solo cuerpo. El algoritmo de Barnes-Hut es de divide-and-conquer, que funciona dividiendo el dominio en cubos (en 3D) y subdividiendolos en grupos de 8.






Problema de N-cuerpos en paralelo

El problema de N-cuerpos se puede paralelizar utilizando **memoria compartida** (OpenMP/CUDA) o **memoria distribuída** (MPI).

En esta clase se hará una descripción del algoritmo y los métodos usados, así como las ventajas/desventajas de cada uno de ellos

Usando memoria compartida se evita el tiempo de comunicación entre procesos, pero tiene el límite dado por la cantidad de núcleos de un procesador.

Usando memoria distribuída, se paga el precio de la comunicación entre procesos pero se puede siempre escalar el cluster para obtener mayor eficiencia.

-  David B. Kirk and Wen-mei W. Hwu *Programming Massively Parallel Processors: A Hands-on Approach*. 2nd. Morgan Kaufmann, 2013. isbn: 978-0-12-415992-1.
-  Norm Matloff. *Programming on Parallel Machines*. University of California, Davis, 2014.
-  Peter S. Pacheco. *An Introduction to Parallel Programming*. 1st. Morgan Kaufmann, 2011. isbn: 978-0-12-374260- 5.
-  Michael J. Quinn. *Parallel Programming in C with MPI and OpenMP*. 1st. McGraw-Hill Education Group, 2003. isbn: 0071232656.
-  Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Program- ming*. 1st. Addison-Wesley Professional, 2010. isbn: 0131387685, 9780131387683.