TEMA-3-CPA.pdf

HuGoCG

Computación paralela

3º Grado en Ingeniería Informática

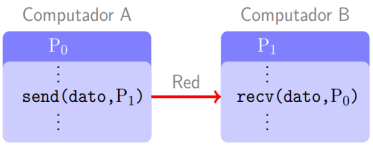
Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

TEMA 3 PASO DE MENSAJES

DISEÑO AVANZADO DE ALGORITMOS PARALELOS

Modelo de paso de mensajes

- Las tareas manejan si espacio de memoria privado.
- Se intercambian datos a través de mensajes.
- La comunicación suele requerir operaciones coordinadas.
- Programación laboriosa / control total de la paralelización.



MPI: Message Passing Interface

Creación de procesos

El programa paralelo se compone de diferentes procesos:

- Suelen corresponderse con procesos del S.O.
- Normalmente un proceso por procesador.
- Cada uno tiene un identificador.

La creación de procesos puede ser:

- Estática: al inicio del programa
  - En línea de comandos (mpiexec).
  - Existen durante toda la ejecución.
  - Es lo más habitual.
- Dinámica: durante la ejecución
  - Primitiva spawn().

Comunicadores

Los procesos se organizan en grupos:

- Para operaciones colectivas, como el envío 1 a todos.
- Se definen mediante índices o con operaciones de conjuntos.

Concepto más general: Comunicador = grupo + contexto.

La comunicación en un comunicador no puede interferir con la de otro. Es útil para aislar la comunicación dentro de una librería. Se definen a partir de grupos u otros comunicadores.

Hay comunicadores predefinidos:

- Mundo (world): formado por todos los procesos creados por mpiexec.
- Propio (self): formado por un solo proceso.

Operaciones básicas de Envío/Recepción

La operación más común es la comunicación punto a punto. Un proceso envía un mensaje (send) y otro lo recibe (recv). Cada send ha de tener un recv emparejado. El mensaje es el contenido de una o más variables.

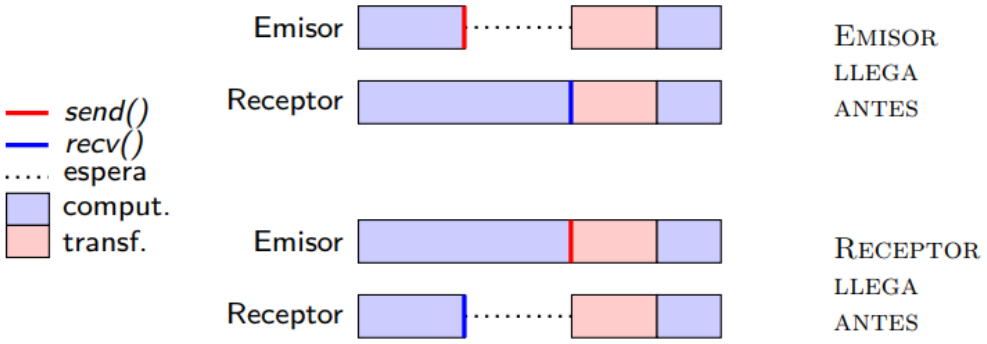
La operación send es segura desde el punto de vista semántico si se garantiza que el proceso 1 recibe el valor que tenía x antes del envío (10).

```
/* Proceso 0 */
x = 10;
send(x,1);
x = 0;
```

```
/* Proceso 1 */
recv(y,0);
```

Envío con Sincronización

En el modo síncrono, la operación send no termina hasta que el proceso ha efectuado el recv correspondiente. Además de la transferencia de datos, los procesos se sincronizan. Requiere un protocolo para que emisor y receptor sepan que puede comenzar la transmisión.



Modalidades de Envío/Recepción

Envío con buffer/envío síncrono:

- Un buffer almacena una copia temporal del mensaje.
- El **send con buffer** finaliza cuando el mensaje se ha copiado de memoria del programa a un buffer del sistema.
- El **send síncrono** no finaliza hasta que se inicia el recv correspondiente en el otro proceso.

Operaciones bloqueantes/no bloqueantes:

- Al finalizar la llamada a send bloqueante es seguro modificar la variable que se envía.
- Al finalizar la llamada a recv bloqueante se garantiza que la variable contiene el mensaje.
- Las no bloqueante simplemente inician la operación.

Finalización de la operación

En las operaciones no bloqueantes hay que poder determinar la finalización tanto en el recv (para poder leer el mensaje), como en el send (para poder sobrescribir la variable).

El send y recv no bloqueantes nos dan un número de operación req:

- wait(req): el proceso se bloquea hasta que ha terminado la operación req.
- test(req): indica si ha finalizado o no.
- waitany y waitall: cuando hay varias operaciones pendientes.

Se puede usar para solapar comunicación y cálculo.

Selección de mensajes

La operación recv requiere un identificador de proceso id:

- No concluye hasta que llega un mensaje de id.
- Se ignoran los mensajes procedentes de otros procesos.

Se utiliza una etiqueta (tag) para distinguir entre mensajes.

Recv(z, any_src, any_tag, status) aceptará el primer mensaje que entre. En status aparecen el emisor y la etiqueta. Los mensajes no seleccionados pasan a la cola de mensajes.

msi

Si tu ordenador tiene más años que los que dura tu carrera: toca cambio.

¿Necesitas un portátil para estudiar pero eres de los que se echa una partidita para desconectar?
Con el Katana GF66 de MSI lo tienes todo; movilidad, autonomía y rendimiento óptimo para compaginar tus dos facetas

Problema: Interbloqueo

Un mal uso de send y recv puede producir interbloqueo.

Caso de comunicación síncrona:

```
/* Proceso 0 */
send(x,1);
recv(y,1);
```

```
/* Proceso 1 */
send(y,0);
recv(x,0);
```

- Ambos quedan bloqueados en el envío

Caso de envío con buffer:

- El ejemplo anterior no causaría interbloqueo
- Puede haber otras situaciones con interbloqueo

```
/* Proceso 0 */
recv(y,1);
send(x,1);
```

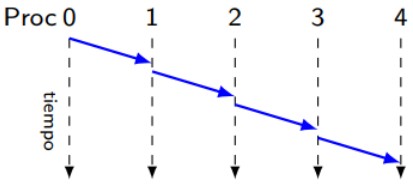
```
/* Proceso 1 */
recv(x,0);
send(y,0);
```

Posible solución: intercambiar el orden de uno de ellos

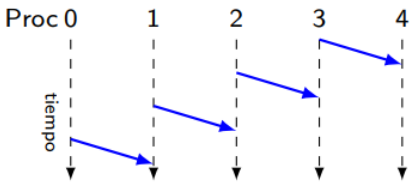
Problema: Serialización

Cada proceso tiene que enviar un dato a su vecino derecho

```
/* Proceso i */
recv(y,i-1);
send(x,i+1);
```




```
/* Proceso i */
send(x,i+1);
recv(y,i-1);
```



Posibles soluciones:

- Protocolo pares-impares: los procesos pares hacen una variante, los impares la otra
- `send` o `recv` no bloqueante
- Operaciones combinadas: `sendrecv`

Comunicación colectiva

Las operaciones colectivas involucran a todos los procesos de un comunicador (suele haber un proceso raíz).

- Sincronización (barries): todos los procesos esperan a que lleguen los demás.
- Movimiento de datos: uno o varios envían a uno o varios.
- Reducciones: además de comunicar se realiza un cálculo sobre los datos.

Estas operaciones pueden realizarse con comunicación punto a punto, pero es recomendable utilizar las primitivas correspondientes.

Tipos:

- Difusión uno a todos:
  - Todos reciben lo que tiene el proceso raíz.
- Reducción todos a uno:
  - Operación dual a la difusión.
  - Los datos se combinan mediante un operador asociativo.
- Reparto (scatter):
  - El raíz envía un mensaje individualizado a cada uno.
- Recogida o concatenación (gather):
  - Operación dual al reparto.
  - Similar a la reducción pero sin operar.
- Difusión todos a todos:
  - p difusiones simultáneas, con proc. raíz distintos.
  - Al final, todos almacenan todos los datos.
- Reducción todos a todos:
  - Operación dual a la difusión todos a todos.

Esquemas algorítmicos

Paralelismo de datos / Particionado de datos

En algoritmos con muchos datos que se tratan de forma similar (algoritmos matriciales):

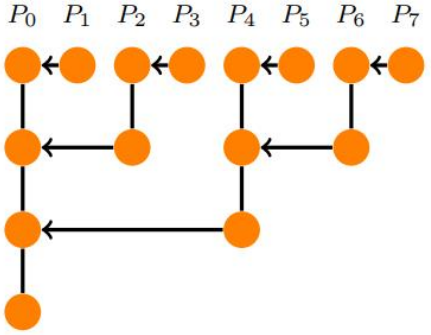
- En memoria compartida se paralelizan los bucles.
- En paso de mensajes se realiza un particionado de datos explícito.

En paso de mensajes puede ser desaconsejable paralelizar. El volumen de computación debe ser al menos un orden de magnitud mayor que el de comunicaciones.

Esquema en árbol

Reducción mediante Recursive Doubling. Se establece log2(p) etapas de comunicación. Tras cada etapa intervienen la mitad de procesos. El que recibe acumula sobre su s local.

```
EN CADA P(pr), pr=0 HASTA p-1
  s = sl;
  PARA j=1 HASTA log2(p)
    SI resto(pr,2^j)==0
      recibir(aux,pr+2^(j-1))
      s = s + aux
    SI NO
      SI resto(pr,2^(j-1))==0
        enviar(s,pr-2^(j-1))
      FSI
    FSI
  FPARA
```



Paralelismo de tareas

En caso de que se generen más tareas que procesos, o la resolución de una tarea genere nuevas tareas, la asignación estática no es viable o tiene problemas de carga desequilibrada. La mejor opción es la asignación dinámica, se van asignando a medida que los procesos quedan ociosos.

Suele implementarse mediante un esquema asimétrico maestro-trabajadores:

- El maestro lleva cuenta de las tareas hechas/por hacer.
- Los trabajadores reciben las tareas y notifican al maestro cuando terminan.

Evaluación de las prestaciones

Tiempo que tarda un algoritmo paralelo en p procesadores, desde que empieza el primero hasta que acaba el último. Se descompone en tiempo aritmético y tiempo de comunicaciones:

$$t(n,p) = t_a(n,p) + t_c(n,p)$$

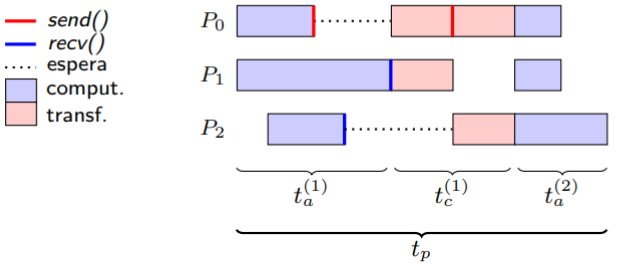
T_a corresponde a todos los tiempos de cálculo:

- Todos los procesadores calculan concurrentemente.
- Es como mínimo igual al máximo del tiempo aritmético.

T_c corresponde a tiempos asociados a transferencia de datos:

- En memoria distribuida t_c = tiempo de envío de mensajes.
- En memoria compartida t_c = tiempo de sincronización.

Ej.: paso de mensajes con tres procesos, P_0 envía a P_1 y P_2



En la práctica:

- No hay separación clara entre fases de cálculo y comunicación (P_1 no tiene que esperar)
- A veces se puede solapar comunicación y cálculo (con operaciones no bloqueantes, por ejemplo P_2)

$$t_p = t_a + t_c - t_{\text{overlap}} \qquad t_{\text{overlap}}\text{: tiempo de solapamiento}$$

Modelado del tiempo de comunicación

Tiempo necesario para enviar un mensaje de n bytes: $t_s + t_w n$

- Tiempo de establecimiento de la comunicación, t_s.
- Ancho de banda, w (máx numero de bytes por seg.).
- Tiempo de envío de 1 byte, $t_w = 1/w$.

Diapos 30 a 32 ejemplos

Parámetros relativos

Speedup: indica la ganancia de velocidad que consigue el algoritmo paralelo con respecto a un algoritmo secuencial:

$$S(n,p) = \frac{t(n)}{t(n,p)}$$

T(n) puede ser el mejor algoritmo secuencial conocido o puede ser el algoritmo paralelo ejecutado en un procesador.

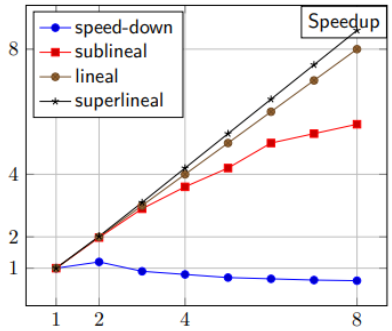
Eficiencia: mide el grado de aprovechamiento que un algoritmo paralelo hace de un computador paralelo:

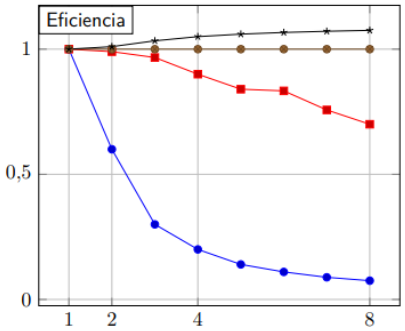
$$E(n,p) = \frac{S(n,p)}{p}$$

Suele expresarse en tanto por cien (o tanto por 1).

Speedup: Casos Posibles

$S(n,p) < 1$	$1 < S(n,p) < p$	$S(n,p) = p$	$S(n,p) > p$
"Speed-down"	Caso sublineal	Caso lineal	Caso superlineal
El algoritmo paralelo es más lento que el algoritmo secuencial	El algoritmo paralelo es más rápido que el secuencial, pero no aprovecha toda la capacidad de los proc.	El algoritmo paralelo es lo más rápido posible, aprovechamiento de los procesadores al 100 %	Situación anómala, el algoritmo paralelo tiene menor coste que el secuencial





Diapo 36 ejemplo

Normalmente la eficiencia disminuye a medida que se incrementa el número de procesadores. El efecto es menos acusado para tamaños de problema grandes.

Diapo 37 gráficas

Ley de Amdahl

Muchas veces, una parte del problema no se puede paralelizar → la ley de Amdahl mide el speedup máximo alcanzable.

Dado un algoritmo secuencial, descomponemos $t(n) = t_s + t_p$

- T_s es el tiempo de la parte intrínsecamente secuencial.
- T_p es el tiempo de la parte perfectamente paralelizable.

El tiempo mínimo paralelo alcanzable será $t(n,p) = t_s + t_p/p$

Speedup máximo:

$$\lim_{p\to\infty} S(n,p) = \lim_{p\to\infty} \frac{t(n)}{t(n,p)} = \lim_{p\to\infty} \frac{t_s + t_p}{t_s + \frac{t_p}{p}} = 1 + \frac{t_p}{t_s}$$

Diseño de algoritmos: Asignación de Tareas

La asignación de tareas o planificación de tareas consiste en determinar en qué unidades de procesamiento y en qué orden se ejecutará cada tarea.

Procesos y procesadores

Proceso: unidad lógica de cómputo capaz de ejecutar tareas computacionales.

Procesador: Unidad hardware que realiza cálculos.

Un algoritmo paralelo se compone de procesos que ejecutan tareas.

La asignación establece correspondencia entre tareas y procesos en la fase de diseño. La correspondencia entre procesos y procesadores se hace al final y probablemente en ejecución.

Ejemplos diapos 42 y 43

Objetivos de la asignación

Minimizar el tiempo de ejecución.

- Tiempo de computación: maximizar la concurrencia signando tareas independientes a procesos distintos.
- Tiempo de comunicación: asignar tareas que se comuniquen mucho al mismo proceso.
- Tiempo de ocio: minimizar las dos causas de ociosidad:
    - Desequilibrios de carga.
    - Espera.

Ejemplos diapo 45

Estrategias generales de asignación

Asignación estática: las decisiones de asignación se toman antes de la ejecución:

1. Se estima el número de tareas, su tiempo de ejecución y los costes de comunicación.
2. Se agrupan tareas en otras mayores para reducir costes de comunicación.
3. Se asocian tareas a procesos.

El problema de asignación estática óptima es NP-completo para el caso general.

Las ventajas son que no añade ninguna sobrecarga en tiempo de ejecución y que el diseño e implementación son más sencillos.

Asignación dinámica: el reparto de trabajo se realiza en tiempo de ejecución.

Este tipo de asignación se emplea cuando las tareas se generan dinámicamente y/o el tamaño de las tareas no se conoce a priori.

Las técnicas dinámicas son más complejas. La principal desventaja es la sobrecarga inducida debida a la transferencia de información de carga y de trabajo computacional entre los procesos y la toma de decisiones para mover carga entre procesos.

La ventaja es que no es necesario conocer el comportamiento a priori, son flexibles y apropiadas para arquitecturas heterogéneas.

Agrupamiento

Se utiliza para reducir el número de tareas con el objetivo de:

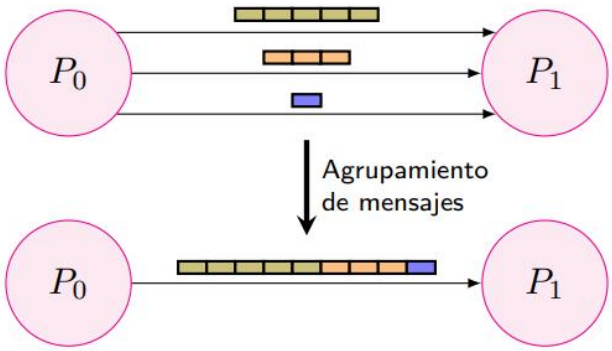
- Limitar costes de creación y destrucción de tareas.
- Minimizar los retardos debidos a la interacción entre tareas.

Estrategias de agrupamiento:

- Minimización del volumen de datos transferidos. Distribución de tareas en bloques de datos, agrupamiento de tareas no concurrentes, almacenamiento temporal de resultados.
- Reducción de la frecuencia de interacciones. Minimizar el número de transferencias y aumentar el volumen de los datos a transferir en cada una.

Reducción de la frecuencia de interacciones

- En *paso de mensajes* significa reducir el número de mensajes (latencia) y aumentar el tamaño de éstos



- En *memoria compartida* significa reducir el número de fallos de caché

Replicación

Implica que parte de los cálculos o datos del problema no se reparten, sino que son realizados o manejados por todos.

- Replicación de datos: copiar datos de acceso común en los distintos procesos con objetivo de reducir las comunicaciones.

- Replicación de cómputo y comunicación: repetir un cálculo en cada uno de los procesos en caso de que este cálculo sea menor que el coste de transferencia y cálculo, tanto secuencial como paralelo.

Diapo 51 ejemplo

Esquemas de asignación

Esquemas de asignación estática

Esquemas estáticos para descomposición de dominio:

- Se centran en estructuras de datos de gran tamaño.
- La asignación de tareas a procesos consiste en repartir los datos entre los procesos.
- Dos tipos:
  - Distribución de matrices por bloques.
  - División estática de grafos.

Esquemas sobre grafos de dependencias estáticos:

- Se suelen obtener mediante descomposición funcional del flujo de datos o descomposición recursiva.

Distribuciones de matrices por bloques

En computación matricial suele suceder que el cálculo de un elemento depende de elementos contiguos (localidad espacial). La asignación hace corresponder porciones contiguas (bloques) del dominio de datos (matriz).

Distribuciones más usuales:

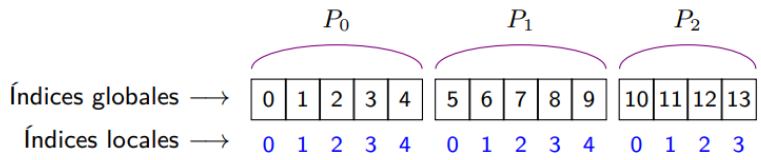
- Unidimensional por bloques de un vector.
- Unidimensional por bloques de filas de una matriz.
- Unidimensional por bloques de columnas de una matriz.
- Distribución bidimensional por bloques de una matriz.

Distribución unidimensional por bloques

El índice global i se asigna al proceso $\lfloor i/m_b \rfloor$ donde $m_b = \lceil n/p \rceil$ es el tamaño de bloque

El índice local es $i \mod m_b$ (resto de la división entera)

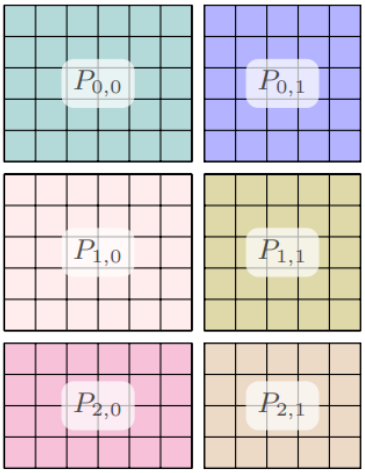
Ejemplo: para un vector de 14 elementos entre 3 procesos



$m_b = \lceil 14/3 \rceil = 5$
Cada proceso tiene m_b elementos (excepto el último)

Distribución bidimensional por bloques

Ejemplo de distribución bidimensional por bloques para una matriz de $m \times n = 14 \times 11$ entre 6 procesos organizados en una malla lógica de 3×2



Cada proceso posee un bloque de $m_b \times n_b = \lceil m/p_m \rceil \times \lceil n/p_n \rceil$, donde p_m y p_n son la primera y segunda dimensión de la malla de procesos, respectivamente (3 y 2 en el ejemplo)

Diapo 58 a 61 ejemplo diferencias finitas

Efecto Volumen-Superficie

El agrupamiento mejora la localidad. Reduce el volumen de comunicaciones. Interesa un agrupamiento con más computación y menos comunicaciones.

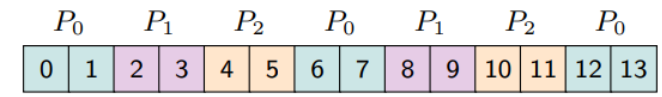
El efecto volumen-superficie consiste en que la carga de computación aumenta proporcionalmente al número de elementos asignados a la tarea. Y el coste de comunicaciones aumenta proporcionalmente al perímetro de la tarea.

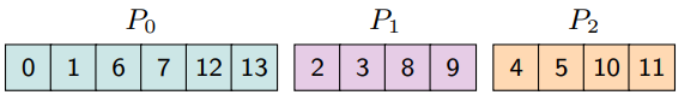
Distribuciones cíclicas

Objetivo: equilibrar la carga durante todo el tiempo de ejecución

- Mayor coste de comunicación al reducirse la localidad
- Se combina con los esquemas por bloques
- Debe existir un equilibrio entre reparto de carga y costes de comunicación: tamaño adecuado de bloque

Distribución cíclica unidimensional (tamaño de bloque 2):





Se aplica igualmente a matrices (por filas o columnas)

Asignación basada en grafos de dependencias estáticos

Caso de descomposición funcional. Suponemos un grafo de dependencias estático y coste de tareas conocido.

El problema de encontrar asignaciones óptimas es NP-completo. Existen situaciones para las que se conocen algoritmos óptimos o enfoques heurísticos.

Ejemplos diapos 66 a 71

Esquemas de Equilibrado Dinámico de la Carga

Cuando la asignación estática no es factible

- Las tareas obtenidas en la descomposición pasan a ser estructuras de datos que representan subproblemas
- Los subproblemas se mantienen en una colección desde la cual se van asignando a los procesos
- La solución de un subproblema puede conducir a la creación dinámica de otros subproblemas
- Necesitan un mecanismo de detección de fin para terminar

Tipos:

- Esquema centralizado, un proceso maestro gestiona la colección de subproblemas y los va asignando
- Esquema distribuido, sin maestro; equilibrado de la carga no trivial