Tema 2: Sistema Multicomputador: paso de mensajes en arquitectura con memoria distribuida

Multicomputador de Memoria Distribuida

Asignatura: Arquitectura de Sistemas Paralelos

Profesor: Francisco Javier Gómez Arribas

Departamento de Tecnología Electrónica y de las Comunicaciones

UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

Escuela Politécnica Superior

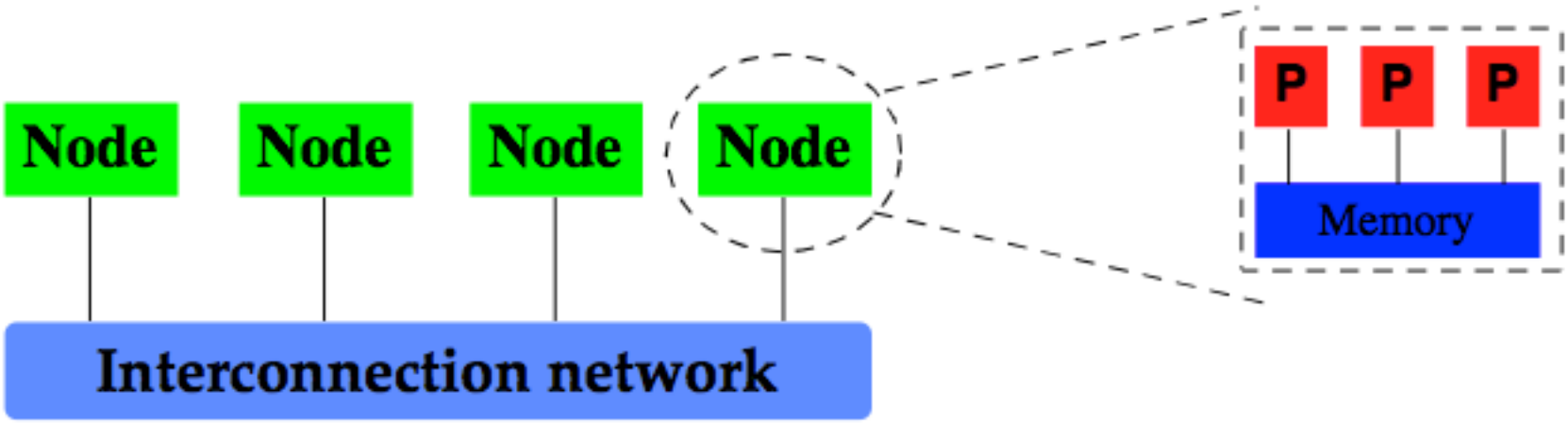
Tema 2: Contenidos

- **Arquitectura multicomputador/multiprocesador**
  - Características del sistema multicomputador
  - Modelo básico de comunicación y computación.
- **Estimación de rendimiento**
  - Solapamiento computación con comunicación.
  - Métricas de rendimiento
  - Ejemplo: variar la estrategia de ejecución de un algoritmo en función de las características de la arquitectura.
- **Programación de multicomputadores por paso de mensajes**
  - Paradigma de programación Maestro/esclavo.
  - Posibilidades de comunicaciones. Definición de datos y Topologías.
  - Planificación de tareas: estática, dinámica bajo demanda.

Multicomputador: Arquitectura idealizada

Multicomputador = Nodos + Red de Interconexión.

- **Nodo = procesador(es) + memoria local**
  - El acceso a memoria local es rápido, porque no involucra conexión de red (acceso a memoria conveccional en sistema uniprocesador)
  - El acceso a memoria remota es lento, involucra conexión de red, con mecanismos de I/O y comunicación (send/receive)



Multicomputadores/multiprocesadores: Arquitecturas

Tipos de arquitecturas:

1. **Memoria distribuida: Distributed Memory MIMD**
   - **Replicar el conjunto procesador/memoria**
   - **Conectar con una red de interconexión**
2. **Memoria compartida: Shared Memory MIMD**
   - **Replicar procesadores y memorias**
   - **Conectar con un bus o una red de interconexión especializada.**
3. **Memoria compartida distribuida: Distributed Shared Memory**
4. **Clusters de Workstation**
5. **SIMD**

Las 4 primeras convergen y se ajustan al modelo MIMD

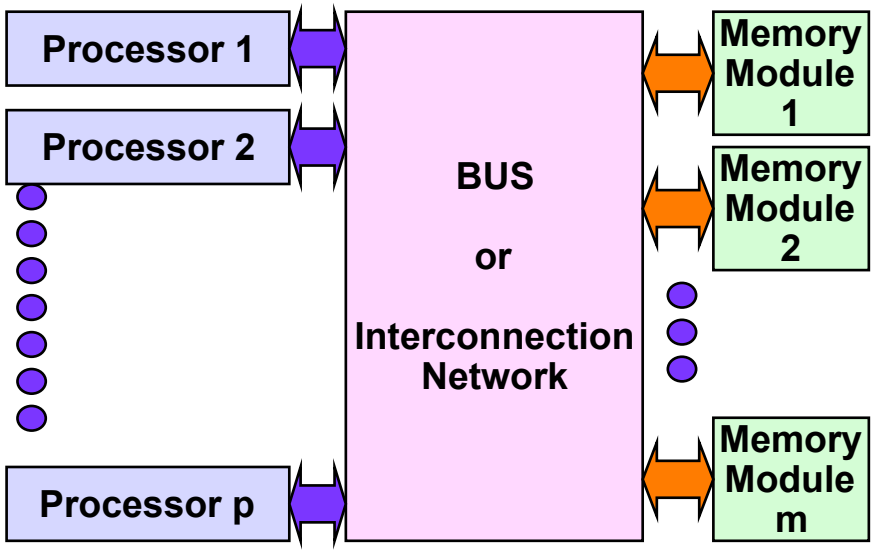
Memoria compartida: Shared Memory Architecture

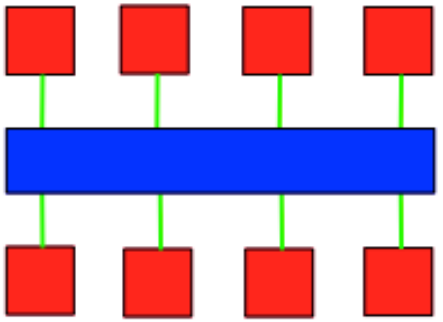
El acceso a memoria compartida es igual para todos los procesadores.

- **Baja latencia**
- **Alto Ancho de banda**
- **La contención del bus limita las escalabilidad**

Para buscar escalabilidad se introduce localidad con:

- **Conexiones jerárquicas multi-etapa**
- **Caches locales reducen:**
  - **Tráfico de Memoria**
  - **Tráfico de Red**
  - **Tiempo de acceso a memoria**



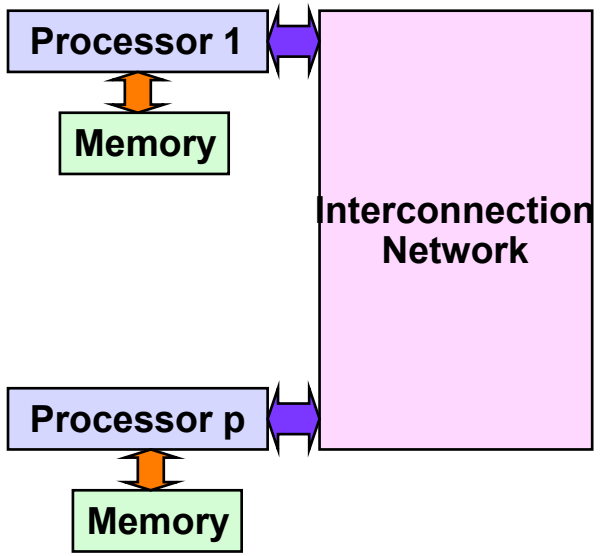


Memoria distribuida: Distributed Memory Machine

MIMD

Procesadores con memoria local conectados por " High-speed interconnection network"

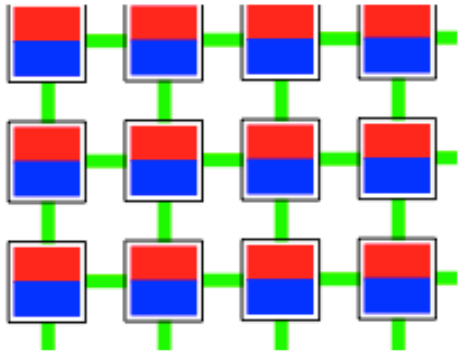
- **Acceso a memoria local mucho más rápido que a la memoria remota**
- **Buen Ancho de banda**
- **Media latencia**



Soporte Hardware para acceso remoto vía:

- **Load/Store**
- **Message passing layer**

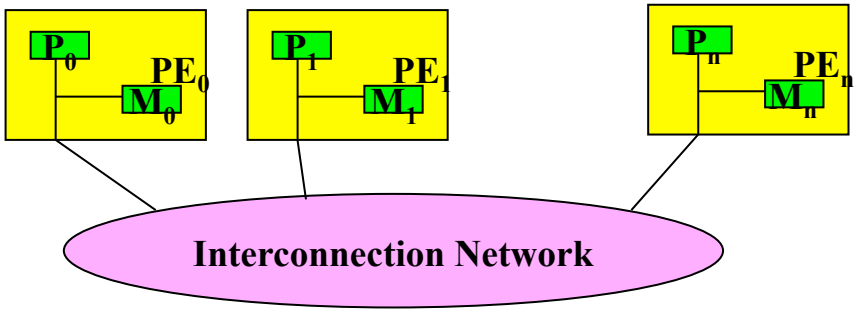
Memoria cache para minimizar accesos, y comunicaciones:



Variantes de Sistemas con memoria distribuida

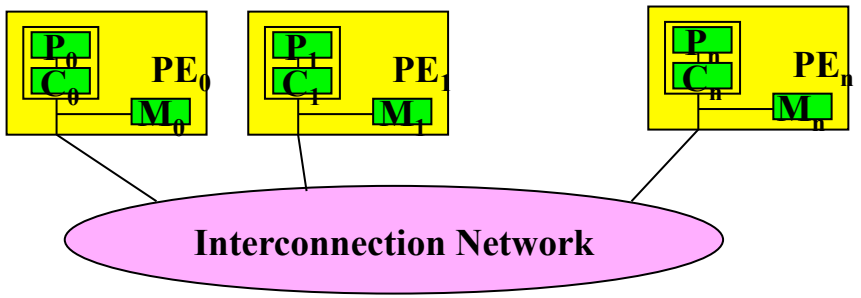
NUMA Acceso a memoria no uniforme: Non-uniform memory access (NUMA)

- El tiempo de accesos a memoria de un procesador en un sistema multiprocesador son diferentes dependiendo de la localización física de la memoria.
- Solo un procesador puede acceder a la memoria a la vez. NUMA permitirían gestionarlo ofreciendo memoria distribuida para cada procesador.
- Difícil de programar



NUMA con coherencia cache : Cache coherent non-uniforms memory access (CC-NUMA)

- Las referencias a memoria de todos los procesadores devolverán automáticamente el último valor actualizado en cualquiera de las caches del sistema .
- Hoy NUMA y cc-NUMA son sinónimos
- Es escalable

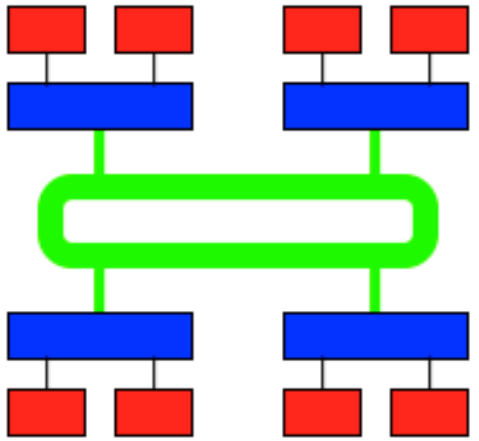


Distributed Shared Memory & Cluster

DSM (Distributed Shared Memory)

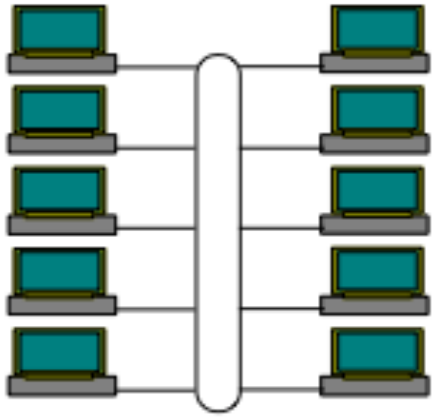
Hibrido entre memoria compartida y distribuida:

- **Un grupo pequeño de procesadores comparte memoria y otros acceden pon una red de inter-conexión escalable.**
  - **Latencia media/baja**
  - **Ancho de banda alto.**



Cluster de workstation:

- **Workstation conectadas por red.**
  - **Coste bajo.**
  - **Alta latencia**
  - **Ancho de banda moderado.**

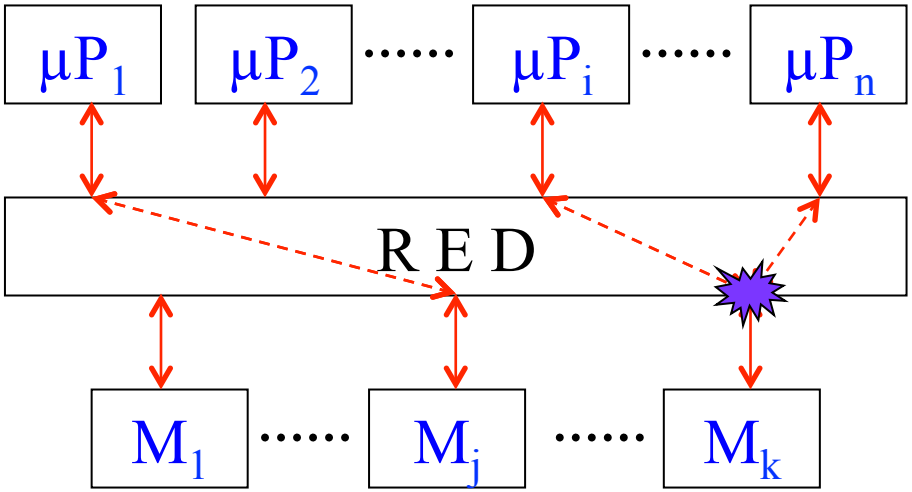


Red de Interconexión

Comunicación Hw <===> Comunicación Sw

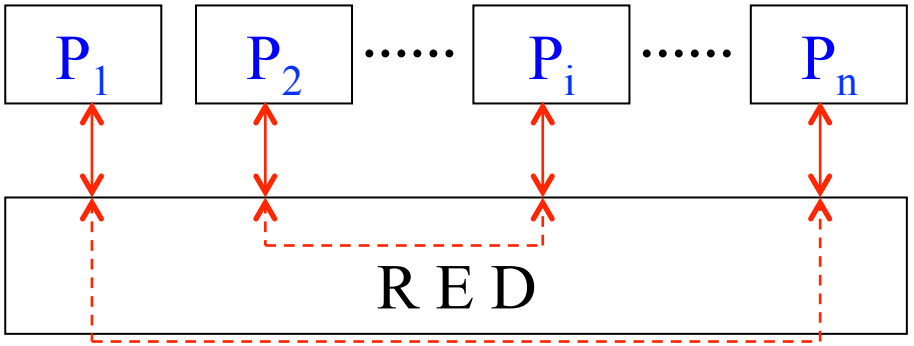
Memoria Común (Load/Store)

Comunicar µPi y Memoria



Paso Mensajes (Send/Receive)

Comunicar Pi con Pj



El parámetro mas importante es la Latencia

Peor de los tiempos para iniciar comunicación

Parámetros de las comunicaciones

Dos parámetros claves en las redes de comunicación son:

- **Latencia: tiempo requerido para inicializar un mensaje. Este es un parámetro crítico en los *códigos grano fino,* el cual requiere mucha comunicación entre procesadores.**
- **Ancho de Banda: porcentaje de sostenibilidad el cual los datos pueden ser enviados a través de la red. Este es un parámetro crítico en *códigos grano grueso,* el cual requiere comunicación de grandes cantidades de datos.**

Optimizar la comunicación

El objetivo es minimizar el tiempo empleado ejecutando una comunicación

- **Minimizar el efecto de la latencia combinando grandes cantidades de pequeños mensajes dentro de un menor número de grandes mensajes..**

Combinar muchos mensajes pequeños en una más grande.

- **dial**
- "Hola mamá"
- **hang up**
- **dial**
- "Como están las cosas?"
- **hang up**
- **dial**
- "en Madrid.?"
- **hang up**
- **dial**
- **……**

- **dial**
- "Hola mamá. Como están las cosas en Madrid.?. Bla ..Bla..."
- **hang up**

- **Transmitiendo un mensaje grande, se tiene que pagar el precio de la latencia una sola vez.**
- **Se transmite más información en menos tiempo.**

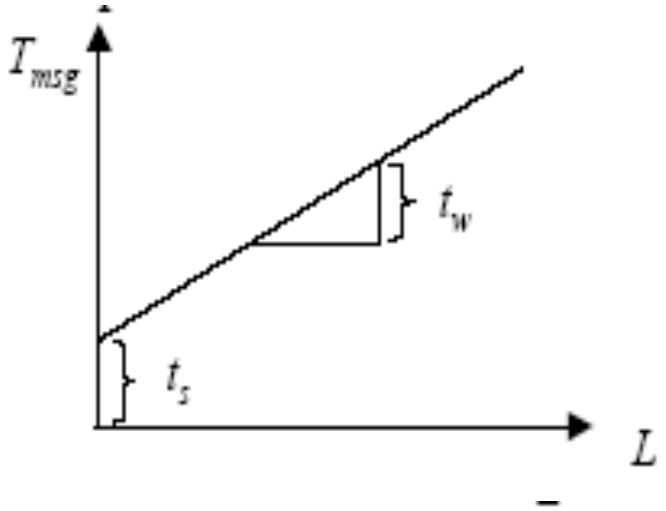
Modelo de comunicación simple

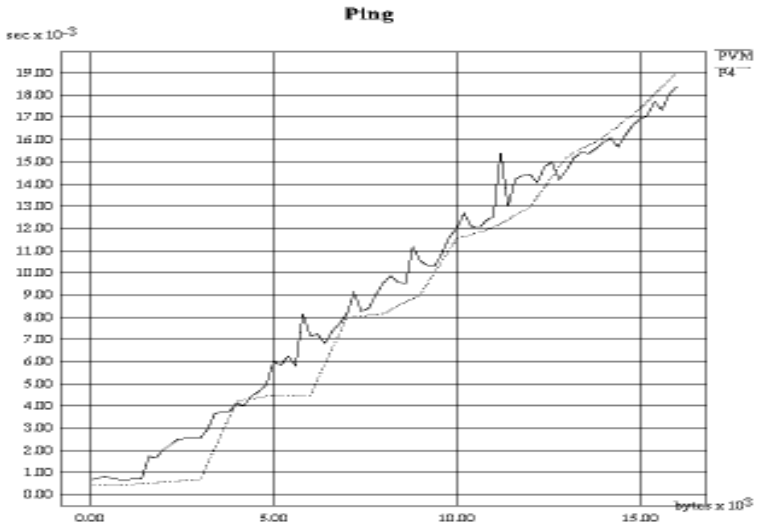
$$t_{msg} = t_s + t_w L$$

t_s tiempo de arranque (latencia)

t_w tiempo de transferencia por palabra

L longitud del mensaje en palabras



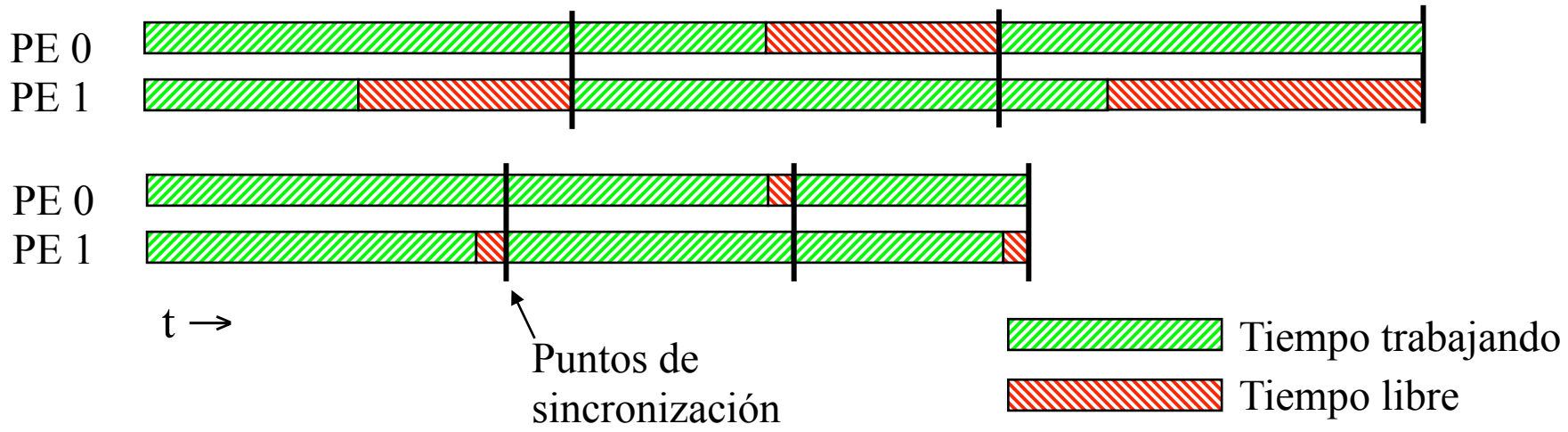


	t_s (µsec)	t_w (µsec)	Bandwidth (Mbits/sec)
IBM SP2	40	0.11	291
Intel DELTA	77	0.54	59
Intel Paragon	121	0.07	457
Meiko CS-2	87	0.08	400
nCUBE-2	154	2.4	13
TM-CM5	82	0.44	73
Workstation on Ethernet	1500	5	6
Workstation on FDDI	1150	1.1	29

Ejecución en Paralelo

La figura muestra el tiempo de ejecución para un código paralelo usando dos procesadores.

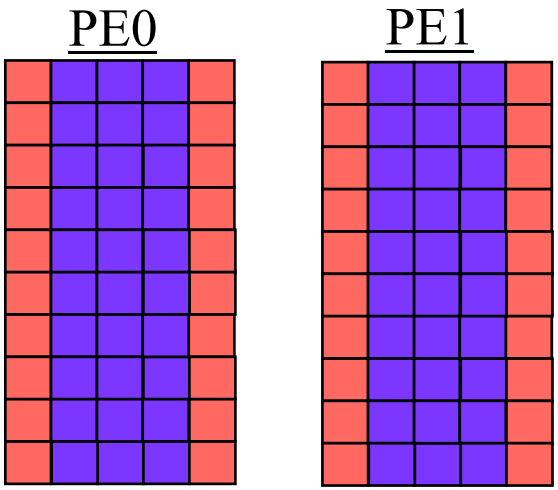
En ambos casos, la cantidad total de trabajo hecho es el mismo, pero en el segundo caso el trabajo está distribuido más efectivamente entre los dos procesadores resultando un tiempo más corto para obtener la solución.



Minimizar el tiempo de comunicación solapando Comunicación y Cálculo

- **Comunicación y cálculo no tiene que hacerse secuencialmente, se pueden solapar.**

Ejemplo, una operación es ejecutada sobre una matriz 10 x 10 que ha sido distribuido en dos procesadores. Asumiendo condiciones periódicas en la frontera.



Operación:

```
y(i,j)=x(i+1,j)+x(i-1,j)+x(i,j+1)+x(i,j-1)
```

- Inicializar comunicación
- Ejecutar cálculo en los elementos internos
- Esperar que las comunicaciones finalicen
- Ejecutar cálculo en los elementos frontera

Elementos frontera – requieren datos del procesador vecino

Elementos internos

Tiempo secuencial/Tiempo paralelo

El tiempo de ejecución de un programa secuencial depende de:

- **tamaño de la entrada o carga de trabajo**
- **compilador,**
- **máquina,**
- **programador, ...,**

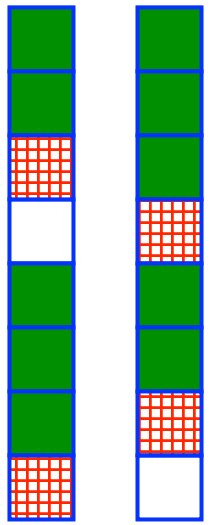
- **El tiempo secuencial es función del tamaño del problema: *t(n)*.**
- **Se mide desde que empieza la ejecución del programa hasta que acaba.**

En un programa paralelo la estimación del tiempo es más compleja y además depende de:

- **número de procesadores *t(n,p)*,**
- **de la manera en que están conectados entre sí (topología)**
- **y con los módulos de memoria (memoria compartida o distribuida)**

- **El tiempo paralelo se mide desde que empieza la ejecución del primero de los procesadores hasta que acaba el último de ellos.**

Tiempo de ejecución paralelo



Esquema de programa paralelo:

Computación 1
Comunicación 1 **(sincronización en memoria compartida)**
Computación 2
Comunicación 2
...

$t(n,p) = t_{comp}(n,p) + t_{comm}(n,p)$

$t_{comp}(n,p)$: suma de los tiempos de las distintas partes de computación

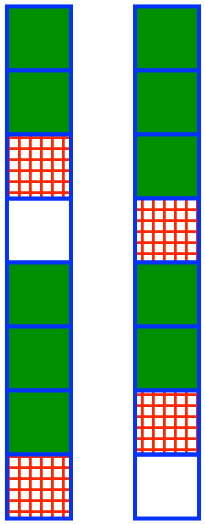
$t_{comm}(n,p)$: suma de los tiempos de las distintas partes de comunicación

DIFICULTADES:

- **No siempre es fácil determinar el orden en que los procesadores empiezan y acaban.**
- **A lo largo de la ejecución de un programa paralelo hay puntos de sincronización de los que no siempre podemos determinar su duración al no saber en el orden en que van a llegar los distintos procesos a estos puntos.**

Tiempo de ejecución paralelo: ejemplo I

Programa paralelo ejecutado en dos procesadores:



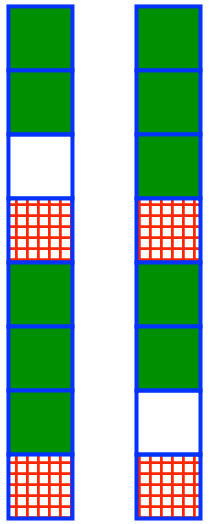
- En cada procesador

$t1 = t_{comp}(n,p)+t_{comm}(n,p)=7$

- **Computación 1 = 2 + 3**
- **Comunicación 1 = 1 + 1**

$t2 = t_{comp}(n,p)+t_{comm}(n,p)=7$

- **Computación 2 = 3 + 2**
- **Comunicación 2 = 1 + 1**



- Tiempo de ejecución del programa

$t(n,p)= t_{comp}(n,p)+t_{comm}(n,p)+t_{overhead}(n,p) = 8$

Tiempo de ejecución en paralelo

Overhead debido a:

- **sincronización,**
- **puesta en marcha de los procesos,**
- **sobrecarga de la red de comunicación,**
- **...**

Por esta razón se debe considerar como tiempo de ejecución:

$$t(n,p) = t_{comp}(n,p) + t_{comm}(n,p) + t_{overhead}(n,p) - t_{solape}(n,p)$$

$t_{comp}(n,p)$: tiempos de las distintas partes de computación

$t_{comm}(n,p)$: tiempos de las distintas partes de comunicación

$t_{overhead}(n,p)$: tiempos de sincronización, latencias,...

$t_{solape}(n,p)$: tiempos con computación y comunicación

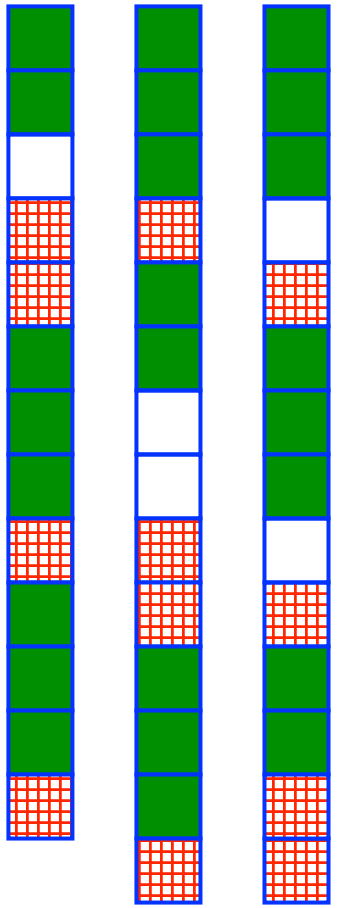
Objetivo

- **Minimizar el tiempo de overhead.**
- **Maximizar el solapamiento.**

Tiempo de ejecución paralelo: ejemplo II

$tproc_i = t_{comp}(n,p)+t_{comm}(n,p) = 12$

- **Computación = 3 + 4 + 4=11**
- **Comunicación 1 = 1 + 1 + 1+1+1+1=6**
- **Overhead 1 = 6**
- **Solapamiento = 9**



$t'(n,p)= t_{comp}(n,p)+t_{comm}(n,p)= 17$

$t(n,p)= t_{comp}(n,p)+t_{comm}(n,p)+$
$t_{overhead}(n,p) - t_{solape}(n,p) = 14$

Análisis del Overhead en programas Paralelos

El tiempo total de ejecución(suma de los tiempos de los proc) en un sistema paralelo es usualmente mayor que el de un sistema serie para resolver el mismo problema

Aparecen Overheads

- **Comunicación interprocesador e Interacciones**
- **Esperas (Idling)**
  - **Desbalanceo de Carga,**
  - **Sincronización,**
  - **Partes serie.**
- **Exceso de Computación**
  - **Algoritmo serie "Sub-optimo"**
  - **Mas operaciones agregadas**

El objetivo es minimizar estos overheads.

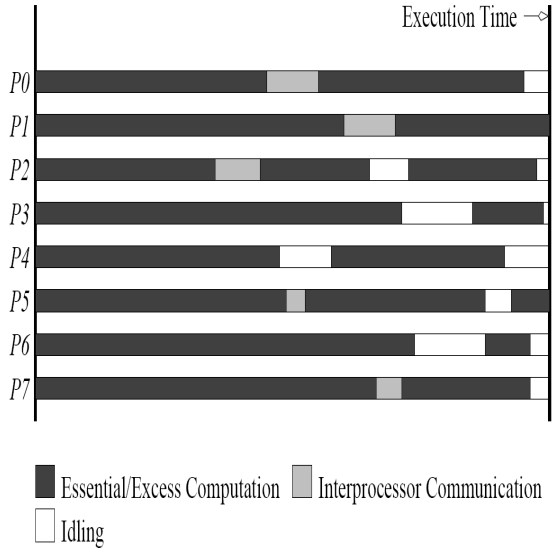


Figure 5.1 The execution profile of a hypothetical parallel program executing on eight processing elements. Profile indicates times spent performing computation (both essential and excess), communication, and idling.

Métrica de Rendimiento: Overhead total paralelo

Sea T_P el tiempo de ejecución paralelo, definido como el tiempo empleado para solucionar un problema en p procesadores

Sea T_{all} el tiempo total colectivamente empleado por todos los elementos de proceso.

Y sea T_S el tiempo serie.

- **La diferencia T_{all} - T_S representa el tiempo total empleado por la combinación de todos los procesadores en trabajo no util. Es el *overhead total*.**

- **Expresando el tiempo colectivamente empleador por todos los elementos de proceso como:**

$$T_{all} = p\, T_P \qquad \textbf{(}p\textbf{ es el número de procesadores).}$$

El overhead (T_o) viene dado por:

$$T_o = p\, T_P - T_S \qquad (1)$$

Métrica de Rendimiento en Sistemas paralelos: Speedup

¿Cuál es el beneficio del paralelismo? Se deben evaluar los siguientes parámetros en cada caso:

- **Speedup (S)**

$$S = Ts/Tp$$

  - **¿Puede conseguirse una aceleración superlinear ?**
    - algoritmos serie no óptimos, extra-computación, características hardware

- **Eficiencia**

$$E = S/p$$

- **Coste (producto procesador-tiempo)**

$$p\ Tp$$

Métrica de Rendimiento en Sistemas paralelos: Coste

Formulación de coste óptimo

- **Coste es el producto del tiempo de ejecución y el número de elementos de proceso empleados (p x T_P).**
- **El coste refleja la suma del tiempo que cada elemento de proceso dedica en la resolución del problema.**

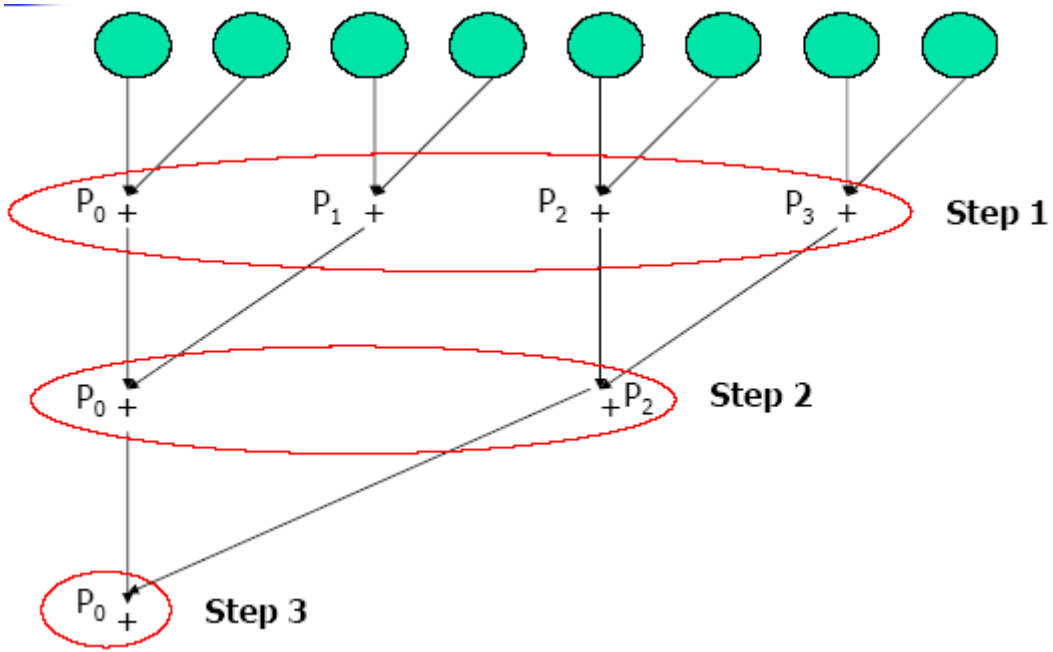
Un sistema paralelo se denomina de coste óptimo si el coste de solucionar un problema en un computador paralelos es idéntico asintóticamente al coste serie.

- **Como $E = T_S / p\, T_P$, para un sistema de coste óptimo, $E = O(1)$.**

El coste también se denomina trabajo o producto *procesador-tiempo*.

Ejemplo: Métricas de rendimiento

Ejemplo: Considere el problema de sumar n números utilizando n elementos de proceso.



- **Cada número está en la memoria local de un procesador**
- **Si n es una potencia de dos, se puede realizar esta operación en $\log n$ paso propagando las sumas parciales en una arbol binario de procesadores.**
- **Las líneas representan las comunicaciones**

Ejemplo: Métricas de rendimiento

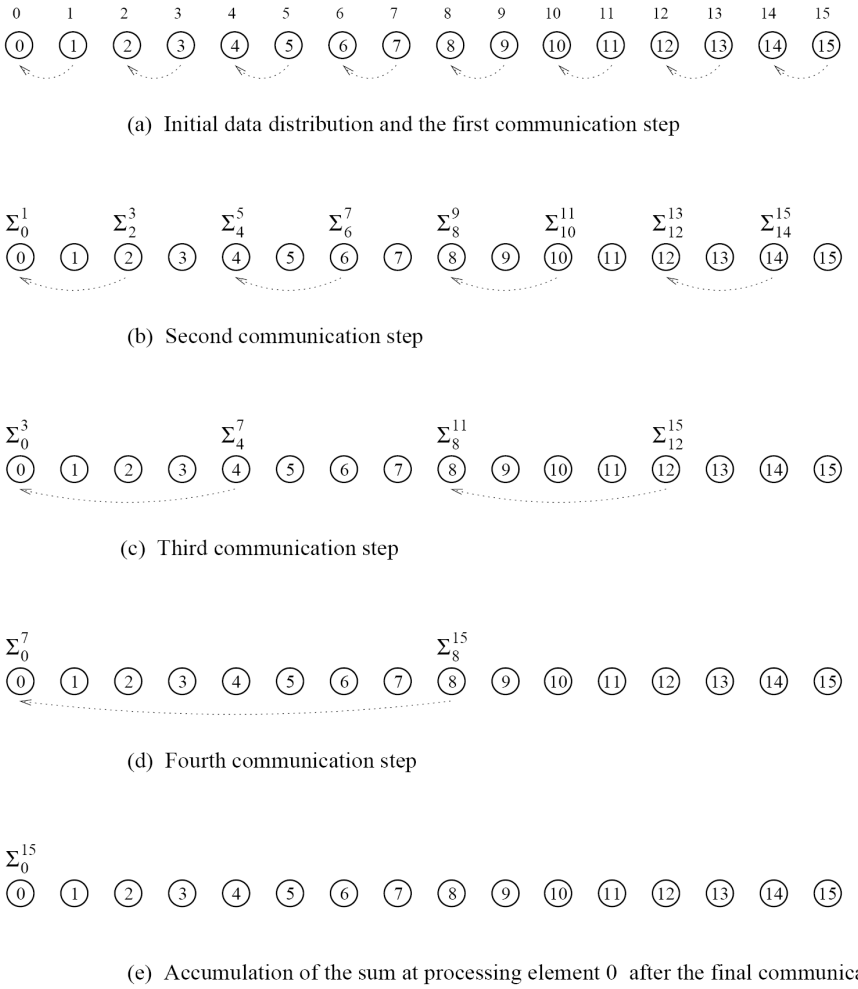


Figure 5.2 Computing the globalsum of 16 partial sums using 16 processing elements. Σ_i^j denotes the sum of numbers with consecutive labels from i to j.

Objetivo: Realizar la suma de 16 números

- **Se usan 16 elementos de proceso, cada uno con un número.**
- **El símbolo $\sum_i^j$ denota la suma de números con identificares consecutivos de i a j**

Ejemplo: Métricas de rendimiento

Suma de n números:

- **Si una suma de dos números tarda un tiempo, t_c y la comunicación de una palabra toma un tiempo $t_{comm}=t_s + t_w$, el tiempo $t_c + t_{comm}$ se mantiene constante y el tiempo paralelo depende del numero de pasos:**

$$T_P = \Theta\ (\log n) \qquad T_P(n)= \log(n)* (t_c + t_{comm})$$

- **El tiempo de ejecución del algoritmo serie es del orden:**

$$T_S = \Theta\ (n) \qquad T_S\ (n)= (n-1)* (t_c + t_{comm})$$

tiempo serie mas justo sin comunicaciones $T_S\ '(n)= (n-1)* (t_c\ )$

- **El Speedup S vendrá dada por:**

$$S = \Theta\ (n / \log n) \qquad S=(n-1)/\log(n)$$

- **La eficiencia es**

$$E = \Theta\ (n / \log n)/n\ = \Theta\ (1 / \log n)$$

y por tanto el coste no es optimo

Efecto de la granularidad en el Rendimiento

Frecuentemente, usando un menor número de procesadores mejora el rendimientos del sistema paralelo.

- **Utilizar un número de procesadores para ejecutar un algoritmo paralelo, menor que el máximo posible del sistema se denomina " reducir el escalado" o *scaling down* de un sistema paralelo.**
- **Una forma de reducir el escalado es considerar cada procesador en el sistema original como varios procesadores virtuales y se corresponden con cada elemento de proceso del sistema scaled down.**
  - **Puesto que el número de elementos de proceso disminuye en un factor n / p, la computación de cada elemento de proceso se incrementa en un factor n / p.**
  - **El coste de comunicación no debería incrementarse en este factor, puesto que alguno de los procesadores virtuales están físicamente asignados al mimo elemento de proceso y pueden compartir datos unos con otros.**
    - Por tanto la razón básica de conseguir una mejora de rendimiento es por incremento de la granularidad.

Efecto de la granularidad en el rendimiento

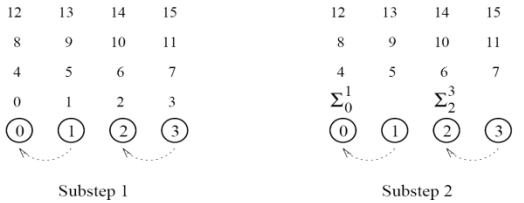
Suma de n números en p procesadores:

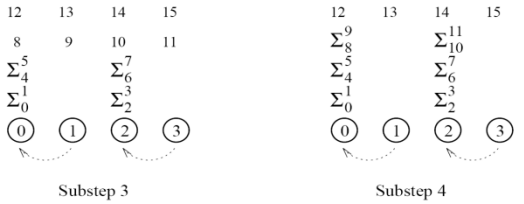
- **Disminuyendo el número de procesadores.**
- **Alcanzar el coste óptimo**
- **Escalado down**

Considerse el problema de sumar n numeros en p elementos de proceso tal que $p < n$ y ambos n y p son potencias de 2.

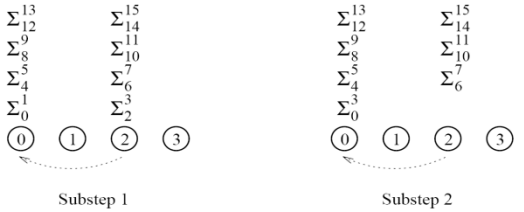
Ejemplo: suma de 16 números en 4 procesadores.

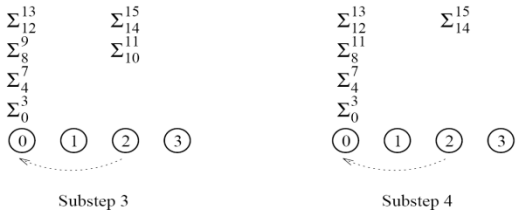
Efecto de la granularidad en el rendimiento





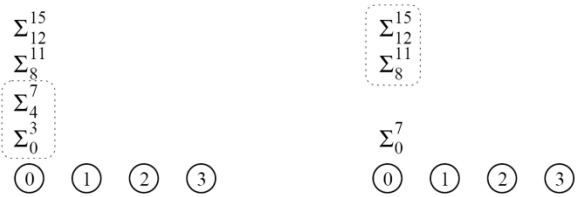
(a) Four processors simulating the first communication step of 16 processors



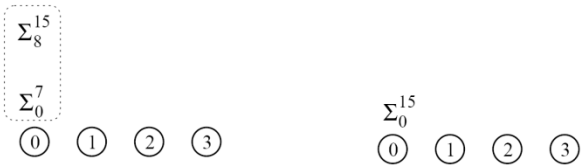


(b) Four processors simulating the second communication step of 16 processors

Figure 5.5 Four processing elements simulating 16 processing elements to compute the sum of 16 numbers (first two steps). Σ_i^j denotes the sum of numbers with consecutive labels from i to j.



(c) Simulation of the third step in two substeps



(d) Simulation of the fourth step (e) Final result

Figure 5.5 (continued) Four processing elements simulating 16 processing elements to compute the sum of 16 numbers (last three steps).

Efecto de la granularidad en el rendimiento

- **Ejemplo: suma de 16 números en 4 procesadores de coste óptimo.**

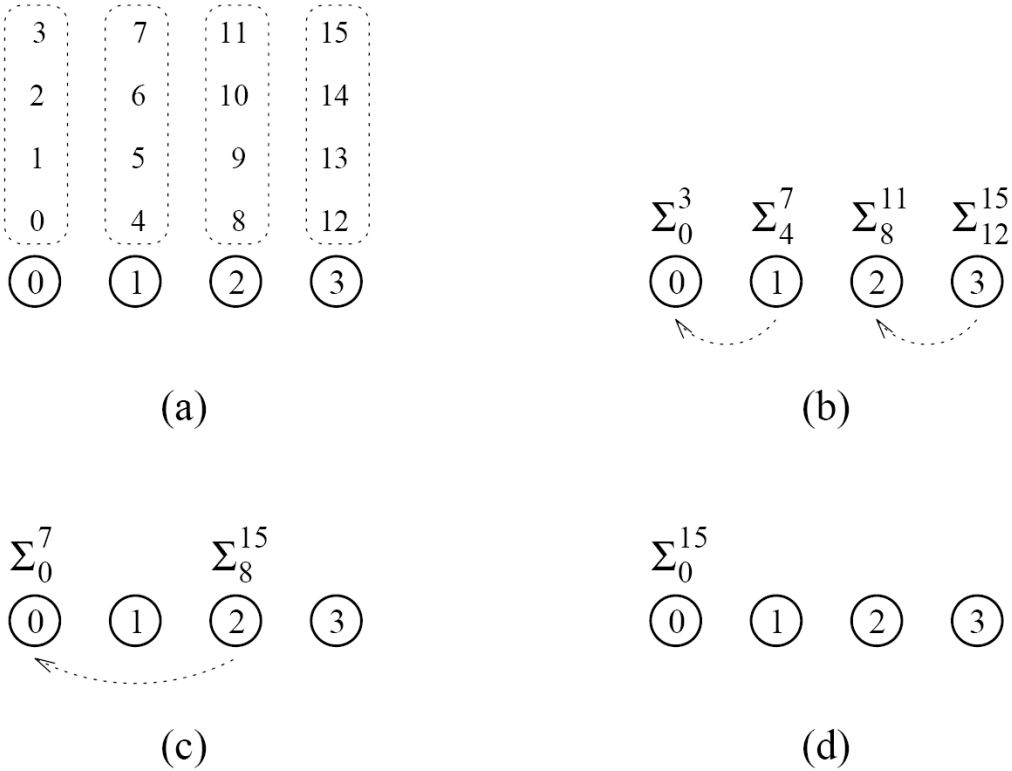


Figure 5.6 A cost-optimal way of computing the sum of 16 numbers using four processing elements.

Escalabilidad de un sistema paralelo con p procesadores

⋆ **Se necesita predecir el rendimiento del algoritmo paralelo conforme se incrementa el número de procesadores p.**

⋆ **Características del overhead To**

- **Lineal con el número de procesadores.**
  - ? Componentes serie

⋆ **Dependencia con *Ts***

- **Típicamente sub-lineal.**

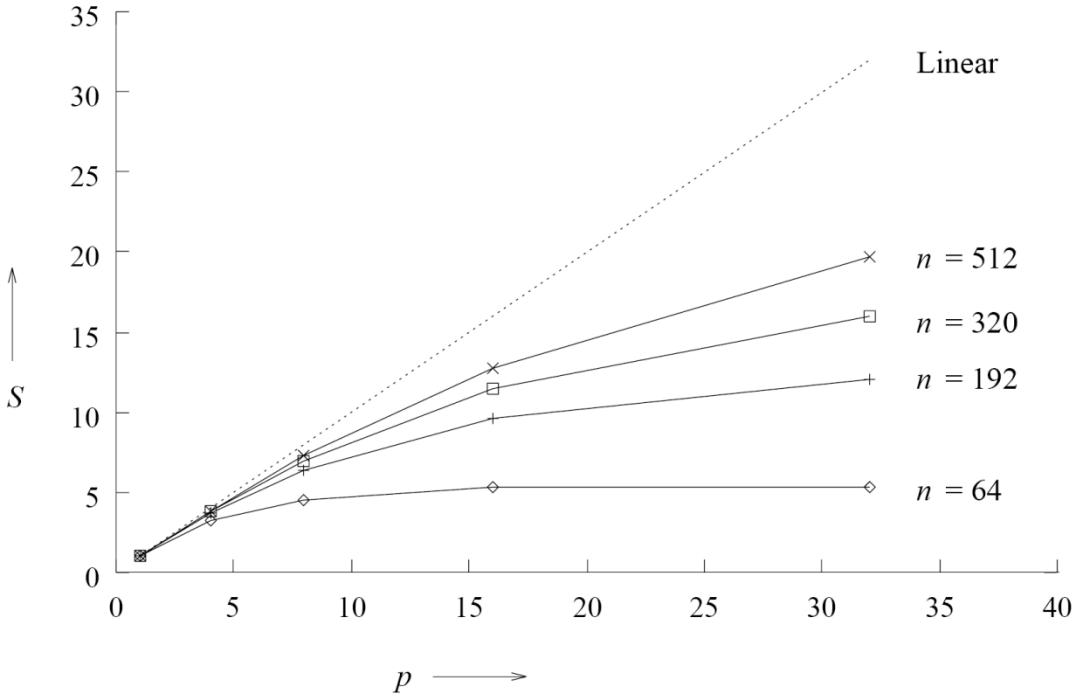


Figure 5.8 Speedup versus the number of processing elements for adding a list of numbers.

Escalabilidad de un sistema paralelo

⋆ **La eficiencia disminuye conforme se incrementa el número de procesadores y se mantiene el tamaño del problema fijo.**

⋆ **La eficiencia se incrementa conforme se incrementa el tamaño del problema y se mantiene el número de procesadores fijo.**

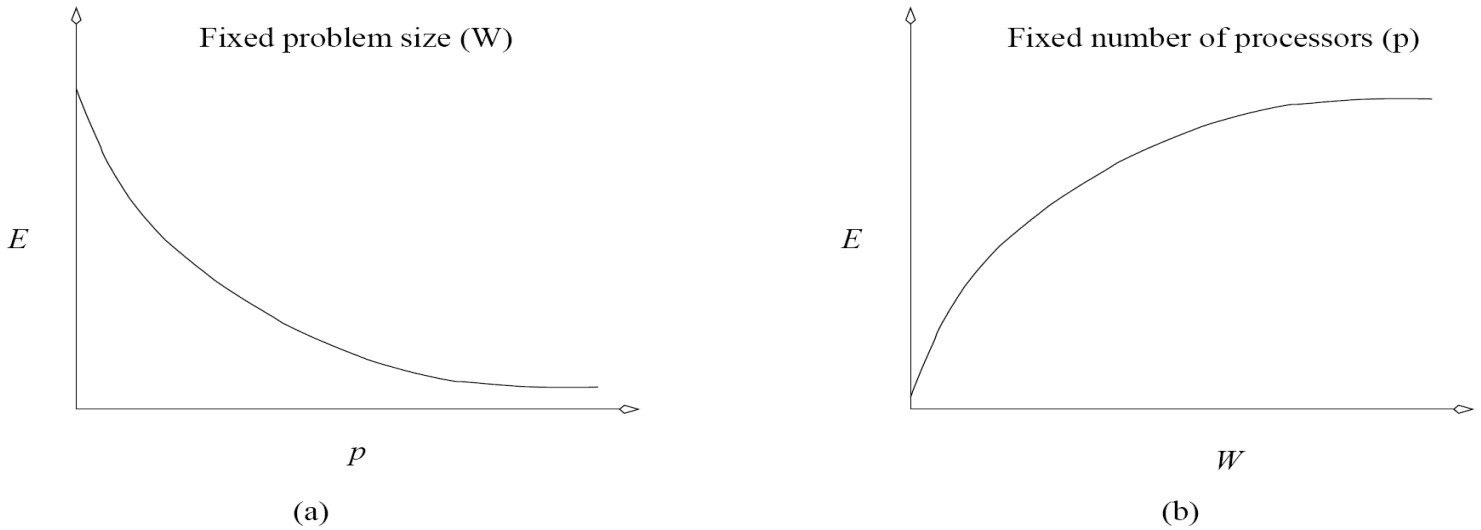


Figure 5.9 Variation of efficiency: (a) as the number of processing elements is increased for a given problem size; and (b) as the problem size is increased for a given number of processing elements. The phenomenon illustrated in graph (b) is not common to all parallel systems.

Escalabilidad de un sistema paralelo

Formulación escalable

- Un sistema paralelo se denomina escalable si se mantiene la eficiencia constante cuando se incrementa p y se incrementa el tamaño del problema.
- La escalabilidad y el coste óptimo están relacionados.
- La función de la isoeficiencia
  - Da la tasa del crecimiento que debe tener el tamaño del problema en relación al número de procesadores p
- Los algoritmos que requieren que el problema del tamaño crezcan con una ratio menor son mas escalables.

Table 5.1 Efficiency as a function of n and p for adding n numbers on p processing elements.

n	$p = 1$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
64	1.0	***0.80***	0.57	0.33	0.17
192	1.0	0.92	***0.80***	0.60	0.38
320	1.0	0.95	0.87	0.71	0.50
512	1.0	0.97	0.91	***0.80***	0.62