
# Computacíon Paralela y Distribuída

2022-II

José Fiestas

24/08/22

Universidad de Ingeniería y Tecnología
jfiestas@utec.edu.pe

# Unidad 2: Metodos de paralelismo

Objetivos:

1. Velocidad, eficiencia, escalabilidad. Ley de Ahmdal
2. Modelos computacionales en paralelo (PRAM)

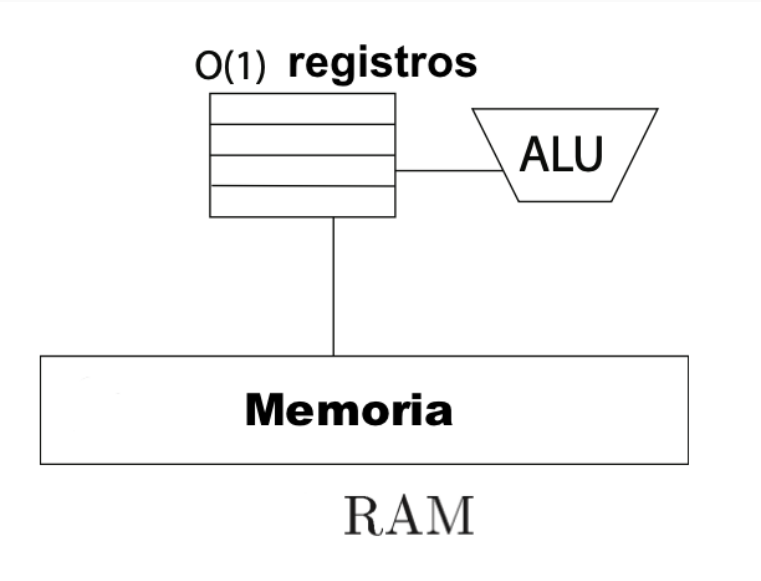
# Modelos computacionales en paralelo (PRAM)

# Modelos Computacionales en Paralelo

Un **modelo computacional** en paralelo es una abstracción de la funcionalidad de un sistema de accesso, manejo y almacenamiento de información, pero que es independiente del hardware/software específico que se utiliza.

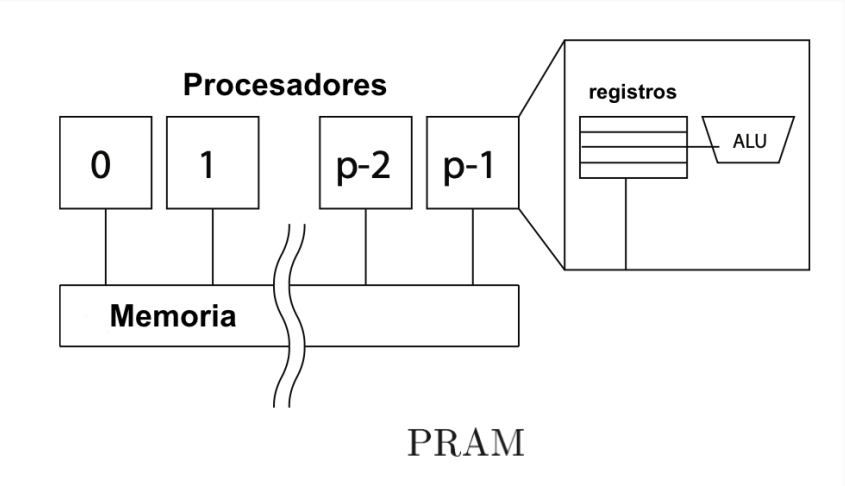
# Modelo RAM (Random Access Machine)

El modelo **RAM** (modelo Von Neumann) es una abstracción de un computador secuencial con memoria infinita, en el cual cada espacio de memoria puede ser accedida en forma directa. Contiene instrucciones para lectura, escritura, y operaciones aritméticas/lógicas. Es útil para el análisis de algoritmos secuenciales.



# Modelo PRAM (Parallel Random Access Machine)

El modelo **PRAM** consiste en un conjunto de procesadores idénticos $\{P_1, P_2, ..., P_n\}$, con acceso a una memoria compartida para lectura/escritura de datos. Cada procesador es un RAM, que ejecuta el mismo código en forma sincrónica. No existe una red entre procesos, si no que se comunican a través de la memoria común. Cada proceso, en cada paso, ejecuta:



PRAM

- lectura de data de la memoria común
- ejecución local de instrucciones
- escritura de data en la memoria común

# PRAM

E.g.

Sea una instrucción de la forma **c := a + b**

Cada proceso ejecuta:

```
global read(a,x) // copiar espacio a en la memoria local
global read(b,y) // copiar espacio b en la memoria local
z= a+b // sumar a y b
global write(z,c) // escribir el resultado en el espacio c
```

donde a, b, y c son espacios de memoria compartidos.

# PRAM

Clasificación segun acceso en paralelo a registros de memoria:
{exclusive, concurrent} **read** {exclusive, concurrent} **write**

- **EREW**, cada locación de memoria puede ser leída o escrita por un solo proceso a la vez
- **CREW**, múltiples procesos pueden leer la memoria pero solo uno puede escribir en ella durante el mismo paso (Broadcast).
- **ERCW**, se permite escritura simultánea, pero la lectura es exclusiva en el mismo paso. No es muy usada
- **CRCW,** múltiples procesos pueden leer y escribir en el mismo paso

E.g. búsqueda de un valor en un array de dimensión n, dados p procesos ($p < n$) ¿Cuál será el mejor método?

# PRAM

Escritura simultánea **CW,** exige resolver el problema de escitura en el mismo espacio de memoria:
- **modelo común**: todos los procesos escriben la misma data
- **modelo arbitrario**: todos los procesos escriben un dato arbitrario en el mismo paso
- **modelo combinado**: ejecuta la acumulación de los resultados escritos arbitrariamente por cada proceso en el mismo espacio de memoria y en el mismo paso.
- **modelo prioritario**: los procesos tienen una prioridad de acceso, y aquel con la mayor prioridad podra escribir en memoria.

# Indicadores en PRAM

**Tiempo de ejecución** (paralelo), T(n): designa el número máximo de ordenes que ejecuta un proceso (número de iteraciones)
**Número de procesos activos**, P(n)
**Espacio de memoria** M(n): registros de memoria utilizados.
**Trabajo**, W(n): cantidad de operaciones por proceso.
Costos máximos $W(n) \sim T(n)\ P(n)$

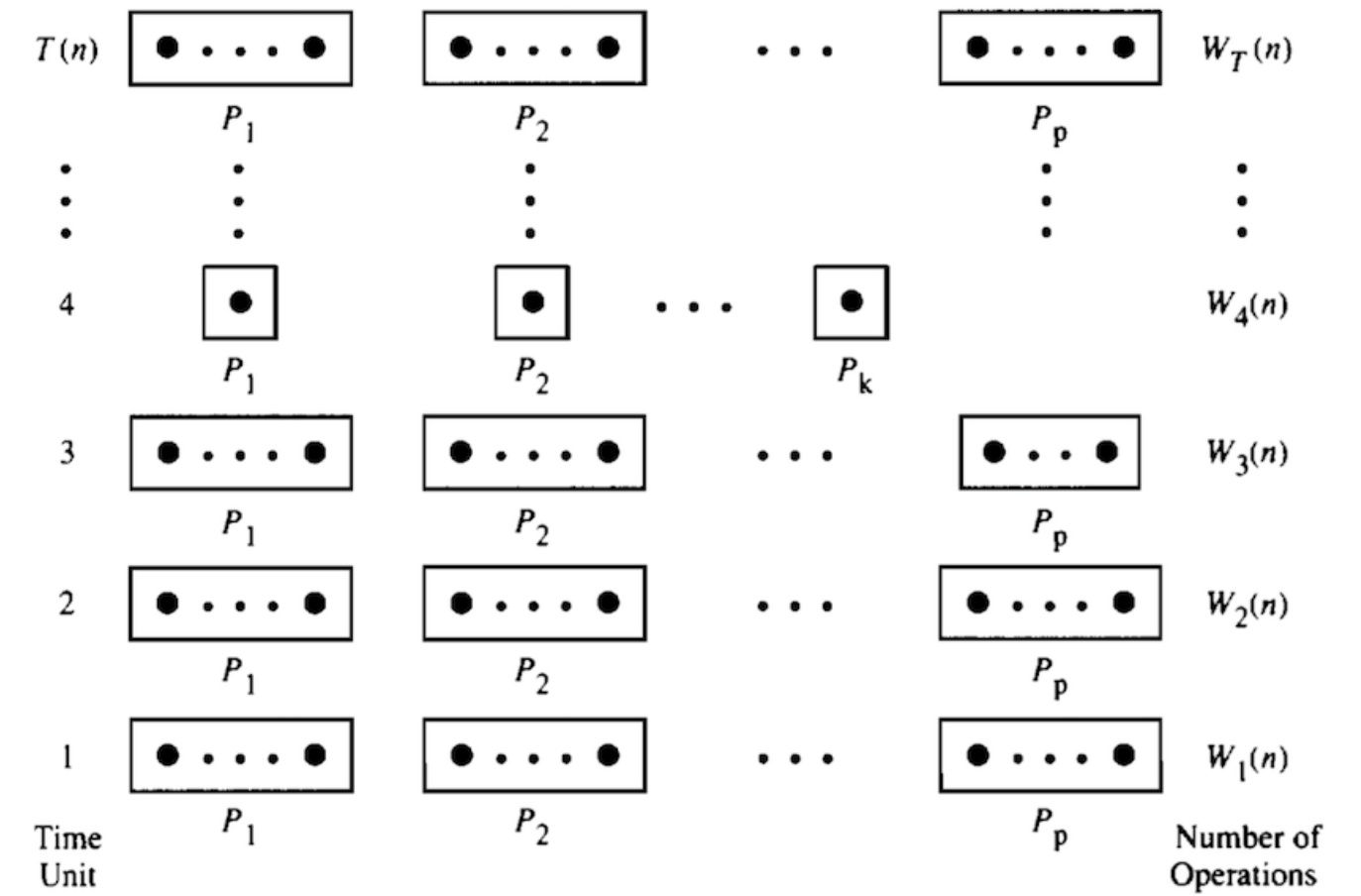
El costo de un algoritmo PRAM sed define como la cantidad de pasos necesarios para completar el programa. Cada paso puede ser de lectura, escritura o de computo local.
El máximo valor es **Worst Case Complexity**

# Paradigma WT (work-time)

Límite inferior: cantidad de operaciones concurrentes por paso
Límite superior: dados p procesos, y un algoritmo con tiempo de ejecución T(n), y trabajo W(n) (cantidad total de operaciones), el algoritmo en un PRAM puede ejecutarse en $\leq \lfloor \frac{W(n)}{p} \rfloor + T(n)$

**Principio WT de asignación de tareas (Teorema de Brent)**
Sea $W_i(n)$ la cantidad de operaciones ejecutadas en el paso i, donde $1 \leq i \leq T(n)$. Si se ejecutan $W_i(n)$ operaciones en $\leq \lceil \frac{W(n)}{p} \rceil$ pasos en paralelo por los p procesos por cada i. Entonces, el algoritmo PRAM logrará ejecutarse en $\leq \sum_i \lceil \frac{W_i(n)}{p} \rceil \leq \sum_i (\lfloor \frac{W_i(n)}{p} \rfloor + 1) \leq \lfloor \frac{W(n)}{p} \rfloor + T(n)$ pasos en paralelo

# Paradigma WT (work-time)



# Velocidad y eficiencia en paralelismo

Hay problemas fáciles de paralelizar, y problemas imposibles de paralelizar. Un objetivo es la aceleración de la ejecución

Si T(n,1) es el costo del algoritmo con un procesador, y T(n,p) el costo de p procesadores, la velocidad está definida como: $S(p) = \frac{T(n,1)}{T(n,p)}$
S(p) es óptima, si S(p) = p

La eficiencia da la carga promedio de los procesadores
$E(n,p) = \frac{S(p)}{p} = \frac{T(n,1)}{pT(n,p)}$

# PRAM: Modelos formales

Lenguaje formal utiliza **pardo** (do in parallel)

for $p_i$, $1 \leq i \leq n$ **pardo**
A(i) := B(i)

Es decir, n operaciones serán ejecutadas en paralelo, i.e. proceso P1 asigna B(1) a A(1), proceso P2 asigna B(2) a A(2), etc

# PRAM: OR, AND

.

Global **OR**:

Entrada en $x[1, ...p]$

Result=0

**for** i=1,...,p **pardo**

  **if** $x[i]$ then Result:=1

Global **AND**:

Entrada en $x[1, ...p]$

Result=1

**for** i=1,...,p **pardo**

  **if not** $x[i]$ then Result:=0

# Máximo de n números (CRCW)

**Entrada** : $A[1, ...n]$
**Salida** : $M[1, ...n]$ $M[i] = 1$, iff $A[i] = max_j A[j]$
**forall** $(i, j) \in \{1, ..., n\}^2$ **pardo**
$B[i, j] := A[i] \geq A[j]$
**forall** $i \in \{1, ..., n\}$ **pardo**
$M[i] = \wedge_{j=1}^{n} B[i, j]$

Se distribuyen n procesos a cada elemento de un array A con n números.
1. Cada proceso compara el valor del elemento con el resto, y determina si ese elemento es el mayor del array A (puede usar un array booleana con 0 o 1)
2. El índice del elemento de A donde el array booleano sea 1 será el máximo de A

# Máximo de n números (CRCW)

El máximo de n números se puede calcular en O(1) usando $n^2$ procesadores, lo que lo hace impracticable para n grande.
En forma **secuencial**, P(n)=1, T(n) = O(n) $\rightarrow$ W(n)=O(n),
En **paralelo**, P(n)=O($n^2$), T(n) = O(1) $\rightarrow$ W(n)=O($n^2$)
S(n)=T(1)/T(n)=n
E(n)=1/n
Además,
Optimiza solo T(n) sin considerar P(n), o W(n).

Podemos decir que un algoritmo es **óptimo** cuando **W(n)** es óptimo.
Normalmente comparado con el algoritmo secuencial.

# PRAM: Suma de n números

**Ingreso**: vector A con $n = 2^k$ números

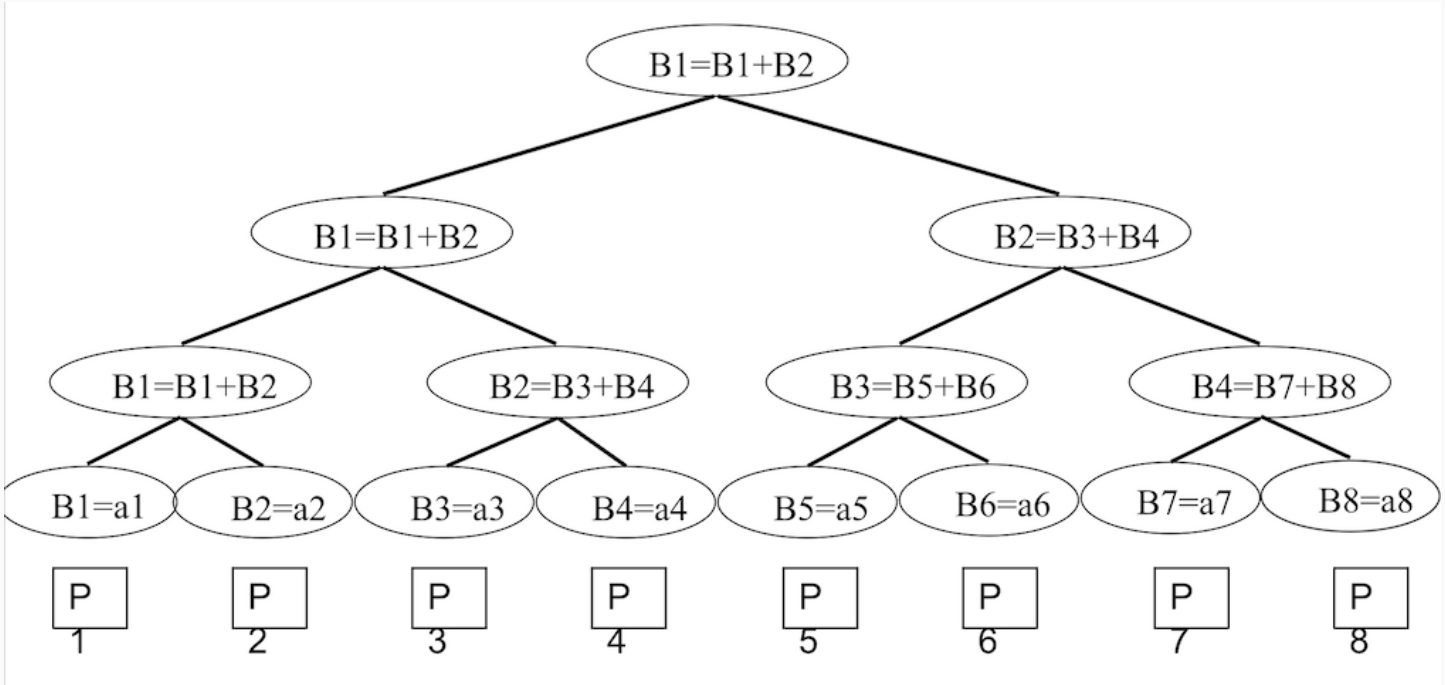
**Salida** : suma S de los números del vector

Pseudocódigo:

1. **for** i = 1 to n **pardo**
   B[i]=A[i]
   **endfor**
2. **for** h = 1 to log(n) **do**
   **for** i = 1 to $n/2^h$ **pardo**
   B[i]=B[2i-1]+B[2i]
   **endfor**
   **end**
3. S:=B(1)

# PRAM: Suma de n números

Para n=8, p=8



# PRAM: Suma de n números

En el paso 1 se ejecutan n operaciones
En el paso 2 se ejecutan $\sum_{j=1}^{log(n)} n/2^j$
En el paso 3 se ejecuta 1 operación
Por lo tanto:

$$W(n) = W_1(n) + W_2(n) + W_3(n) = n + \sum_{j=1}^{log(n)} n/2^j + 1 = O(n)$$

$$T(n) = O(log(n) + 2) = O(log(n))$$

Este algoritmo ejecuta un modelo PRAM EREW

# W(n) vs costo

Dado un algoritmo paralelo con tiempo de ejecución T(n) y un total de W(n) operaciones (tareas), este puede simularse en un PRAM de p procesos en $T_p(n) = O(\frac{W(n)}{p} + T(n))$. El costo (máximo) está definido como

$$C_p(n) = T_p(n) \cdot p = O(W(n) + T(n) \cdot p)$$

W(n) y C(n) coinciden cuando $p = O(\frac{W(n)}{T(n)})$

Para la suma de números:
W(n)=O(n) , T(n) = O(log(n)), $C_p(n) = O(n + plog(n))$
I.e. si $p = O(\frac{W(n)}{T(n)}) = O(\frac{n}{log(n)})$
Entonces, $C_p(n) = O(n + \frac{n}{log(n)} log(n)) = O(n) = W(n)$

# Algoritmo óptimo en paralelo

Un algoritmo en paralelo es considerado óptimo si $W(n) = O(T^*(n))$, donde $T^*(n)$ es el tiempo secuencial óptimo
Ya que un algoritmo óptimo puede ejecutarse en un PRAM de p procesos en un tiempo $T_p(n) = O(\frac{T^*(n)}{p} + T(n))$ (Teorema de Brent), el speedup se calcula como

$$S_p(n) = \Omega(\frac{T^*(n)}{\frac{T^*(n)}{p} + T(n)}) = \Omega(\frac{pT^*(n)}{T^*(n) + pT(n)})$$

Por lo tanto, un speedup óptimo ($S_p(n) = \mathrm{O(p)}$) sucede cuando $p = O(\frac{T^*(n)}{T(n)})$

# PRAM

Para emular más cercanamente a un computador real, se han creado varios modelos PRAM, como:
- **SB-PRAM**, una arquitectura MIMD - UMA, que utiliza operadores lógicos para acceder a la memoria compartida, simulados bajo un esquema round-robin (equitativo). Estas máquinas son un ejemplo de procesadores multithreading.
- **phase PRAM**, en el que las operaciones están divididas en fases, tal que los procesos pueden trabajar sin sincronizacion en las fases, pero son sincronizados al final de las fases
- **delay PRAM**, introduce un retraso entre el momento en que se produce la data en un proceso, y el momento en que es utilizada por otro proceso, con el objetivo de medir retrasos en acceso a memoria.
- **block PRAM**, donde el acceso. memoria toma un tiempo l+b, donde l es el tiempo de inicialización de la comunicación, y b es el tamaño del bloque de memoria accedido.

# Modelos Computacionales en Paralelo

**Modelo BSP** (bulk synchronously parallel):

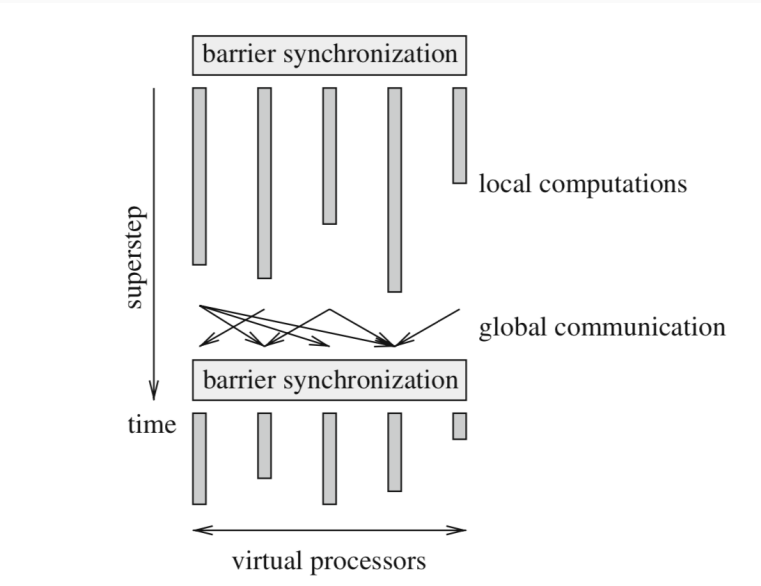
Es una computadora que consiste en procesos con memoria local y capaz de comunicarse punto a punto a través de una red.

La idea es que el software sea independiente de la arquitectura, y sea, por consecuencia, portable.
El cómputo está organizado en **supersteps**, que a su vez, constan de tres fases:
- cálculo simultáneo en cada proceso
- comunicación entre procesos
- barrera de sincronización que culmina la operación de comunicación y hace data visible a los procesos

# Modelos Computacionales en Paralelo

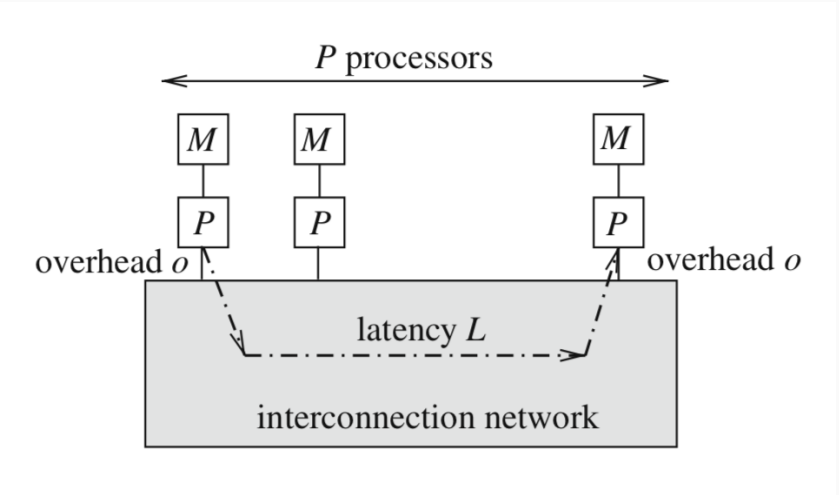
**Modelo BSP** (bulk synchronously parallel):
Una unidad de cálculo se compone de:
- **p**: procesos virtuales, que ejecutan cálculos en cada superstep
- **s**: la velocidad de ejecución en número de operaciones (aritmética/lógica) por segundo, por proceso
- **l**: número de pasos necesarios por barrera de sincronización
- **g**: número de pasos promedio necesarios para transferir una palabra, tal que la ejecución de un proceso con m palabras, necesita $l \cdot m \cdot g$ pasos.

**Modelo BSP** (bulk synchronously parallel):
El tiempo
de ejecución de un superstep
($T_{superstep}$) esta dado por:
$T_{superstep} = max(w_i) + h \cdot g + l$
Donde
$w_i$ es el tiempo necesitado por
proceso i, $h \cdot g$ es el tiempo total
de comunicación y l es el tiempo
de la barrera de sincronización.



**Modelo log P**

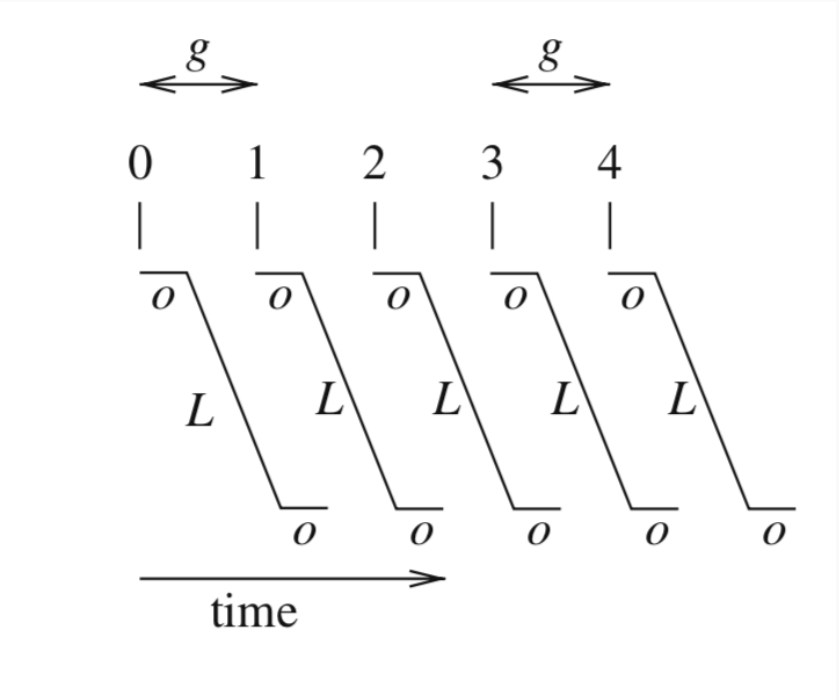
Tiene la estructura de BSP, i.e. un sistema de memoria distribuído, con los siguientes parámetros:




- **L** (latencia), es un límite superior de la latencia de la red, al enviar un mensaje pequeño.
- **o** (overhead), el tiempo requerido en comunicación, durante el cual no se puede ejecutar ninguna otra operacion
- **g** (gap), es el tiempo mínimo entre envio/recibo de mensajes consecutivos por proceso
- **P** (procesos), el número de procesadores de la máquina

**Modelo log P**

El tiempo de
ejecución de un algoritmo en
el modelo **log P** es el máximo
de los tiempos de ejecución
entre todos los procesos
La transmisión
de un mensaje largo se hace
en varios mensajes pequeños.



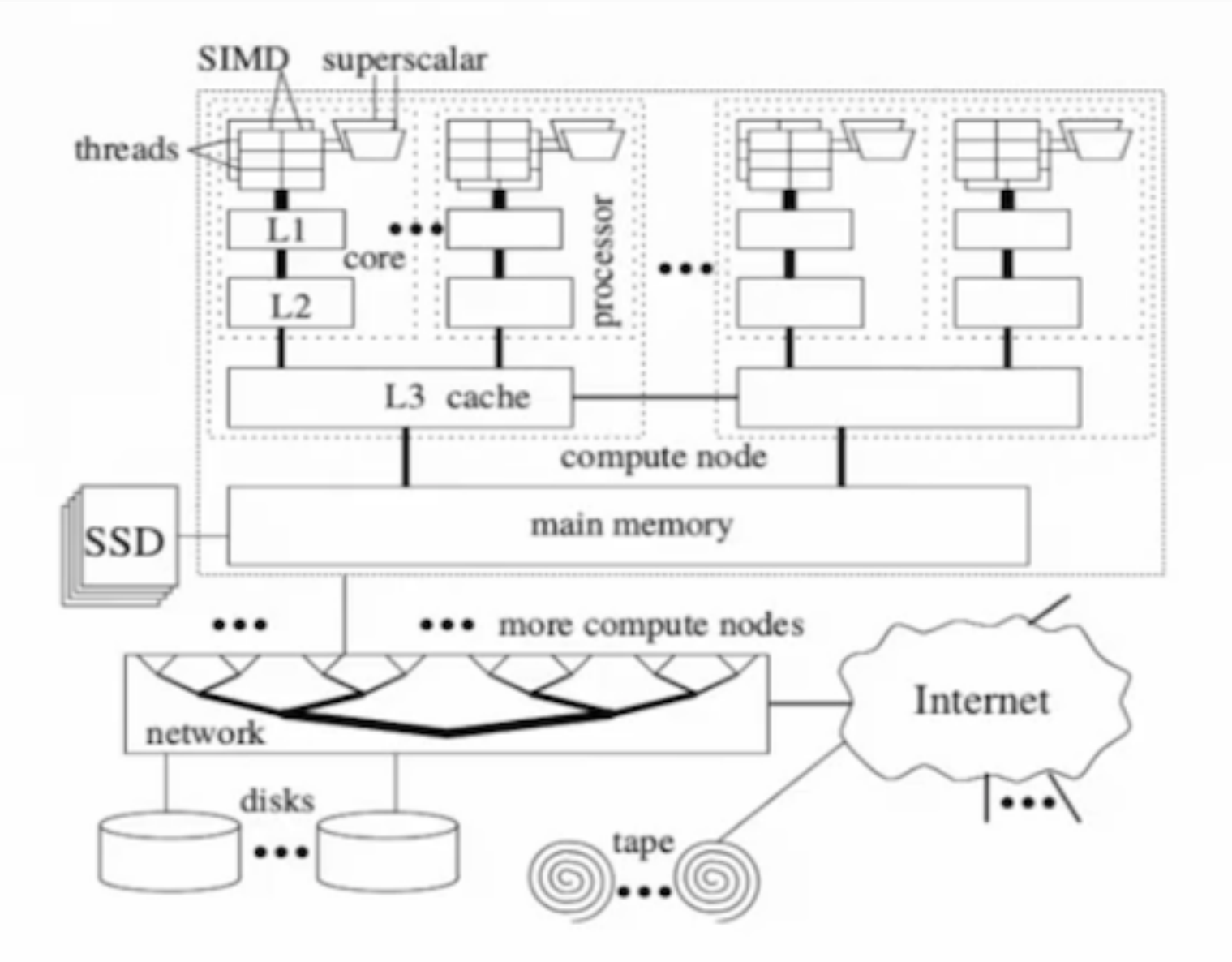
# Modelos Computacionales en Paralelo

**Modelo realístico**:
- asincrónico
- CRQW (concurrent read, queued write), el costo de p procesos accediendo a un espacio de memoria simultáneamente, es O(p)
- Operaciones de escritura consistentes con instrucciones atómicas
- jerarquía en distribución de memoria

Estrategia es crear modelos con pocos niveles de paralelismo (híbridos)

# Modelos Computacionales en Paralelo

## Modelo realístico:



# Modelos Computacionales en Paralelo

**DAG** $\rightarrow$ **PRAM**:
Cada nivel del DAG que puede ser paralelizado, se representa como un modelo PRAM

**DAG** $\rightarrow$ Red:
- Nodos son procesos
- Aristas son líneas de comunicación en la red
- Tiempo de ejercución depende de los nodos, aristas, y de la longitud de las rutas en cada nivel del DAG.

**PRAM** $\rightarrow$ **DAG**:
Identificar bucles y condicionales con distintos procesos en PRAM

# Bibliografía i



Joseph Jájá *An introduction of parallel algorithms.* University of Maryland. *** Ch1.3.2., 1.5.


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.