High-Performance Computing para Modelamiento Matemático con (Py)CUDA

Sesión 1: *Introducción al High Performance Computing y Computo Paralelo*

Diego Maldonado

Universidad Católica del Maule

16 de diciembre de 2025

Índice

1. High Performance Computing
   - ¿Qué es High Performance Computing (HPC)?
   - ¿Por qué HPC no es lo mismo que "computación normal"?
   - Hardware típico en HPC (visión general)
   - Software que suele acompañar al hardwares
   - ¿Por qué HPC es importante hoy?
2. Medición del Algoritmo
   - Recursos
   - Notación de la O grande
3. Cómputo paralelo
   - Motivación
   - Modelos de cómputo paralelo
4. Modelo de memoria compartida
   - Coordinación de las unidades de procesamiento
   - Algoritmos y pseudocódigo en la PRAM
   - Gasto de recursos en la PRAM
   - Diferencias entre la PRAM y la RAM
5. Técnicas de programación en la PRAM: división de datos
6. Técnicas de programación en la PRAM: Árbol balanceado
   - Preliminares
   - Multiplicación de matrices
   - Multiplicación de matrices

- **HPC** es el uso de **múltiples recursos computacionales en paralelo** para resolver problemas que serían **demasiado lentos** en un computador tradicional.
- En vez de "un computador más rápido", HPC suele ser:
  - **muchos núcleos** trabajando a la vez (CPU),
  - **muchas GPUs** procesando en paralelo,
  - y/o **muchos nodos** conectados en un clúster.
- Objetivo principal: **reducir tiempo de cómputo** y/o **aumentar el tamaño/precisión** de los modelos.

Diferencia clave

En HPC, el foco está en **paralelizar** y **escalar** el cálculo: dividir el problema en partes y ejecutarlas simultáneamente.

- **Computación tradicional:** 1 máquina, pocos núcleos, ejecución principalmente secuencial.
- **HPC:** decenas/miles de núcleos y/o GPUs; coordinación y comunicación entre procesos.
- HPC requiere pensar en:
  - **rendimiento** (performance),
  - **uso eficiente de memoria**,
  - **comunicación** entre tareas (latencia/ancho de banda),
  - **balance de carga**.

- **CPU (muchos núcleos):** buena para control, lógica, y tareas con ramificación.
- **GPU (miles de hilos):** ideal para operaciones masivas y repetitivas (matrices, filtros, simulaciones).
- **Memoria RAM:** capacidad y velocidad; jerarquía de cachés importa mucho.
- **Almacenamiento:** SSD, y a nivel clúster, sistemas distribuidos.

Idea central

En HPC, **la comunicación y la memoria** pueden ser tan importantes como el cómputo.

- **CPU (muchos núcleos):** buena para control, lógica, y tareas con ramificación.
- **GPU (miles de hilos):** ideal para operaciones masivas y repetitivas (matrices, filtros, simulaciones).
- **Memoria RAM:** capacidad y velocidad; jerarquía de cachés importa mucho.
- **Almacenamiento:** SSD, y a nivel clúster, sistemas distribuidos.

Idea central

En HPC, **la comunicación y la memoria** pueden ser tan importantes como el cómputo.

- **Paralelismo en CPU:** OpenMP / hilos / vectorización SIMD.
- **Paralelismo distribuido:** MPI (Message Passing Interface) para comunicación entre nodos.
- **Aceleración con GPU:** CUDA, OpenCL, o bibliotecas (BLAS/cuBLAS, etc.).
- **Planificación de trabajos:** gestores de colas (p.ej., SLURM) para asignar recursos.

Mensaje

HPC no es solo "más hardware": es un **ecosistema** (hardware + software + buenas prácticas).

- **Paralelismo en CPU:** OpenMP / hilos / vectorización SIMD.
- **Paralelismo distribuido:** MPI (Message Passing Interface) para comunicación entre nodos.
- **Aceleración con GPU:** CUDA, OpenCL, o bibliotecas (BLAS/cuBLAS, etc.).
- **Planificación de trabajos:** gestores de colas (p.ej., SLURM) para asignar recursos.

Mensaje

HPC no es solo "más hardware": es un **ecosistema** (hardware + software + buenas prácticas).

- **Ciencia y simulación:** clima, oceanografía, geofísica, astrofísica, materiales.
- **Ingeniería:** CFD, elementos finitos, optimización de diseños, gemelos digitales.
- **Datos e IA:** entrenamiento e inferencia a gran escala (deep learning, modelos fundacionales).
- **Salud:** genómica, diseño de fármacos, análisis de imágenes médicas.
- **Industria y negocio:** logística, detección de fraude, pricing, análisis masivo en tiempo acotado.

Tendencia actual

El tamaño de los datos y la complejidad de los modelos crecen más rápido que el rendimiento de un solo núcleo.

Medición del Algoritmo

Definición

La complejidad computación es la disciplina que se dedica a discriminar los problemas y algoritmos en función de los recursos computacionales que estos utilizan.

Problema: Es cualquier pregunta. Puede tener entradas (parámetros) y salidas (respuesta a la pregunta). **Ej:** $P_2(x)$: ¿Cuántos años tiene x?

Algoritmo: Conjunto de pasos a seguir con las siguientes propiedades:

1. No ambiguos
2. Bien definidos
3. Finitos

Los algoritmos pueden tener entradas y salidas, que se obtienen al después de ejecutar el algoritmo con las entradas.

Programa: Conjunto de instrucciones escritas en algún lenguajes de programación. Dependen de entradas y al ejecutarse entrega salidas

Definición

La complejidad computación es la disciplina que se dedica a discriminar los problemas y algoritmos en función de los recursos computacionales que estos utilizan.

Problema: Es cualquier pregunta. Puede tener entradas (parámetros) y salidas (respuesta a la pregunta). **Ej:** $P_2(x)$: ¿Cuántos años tiene x?

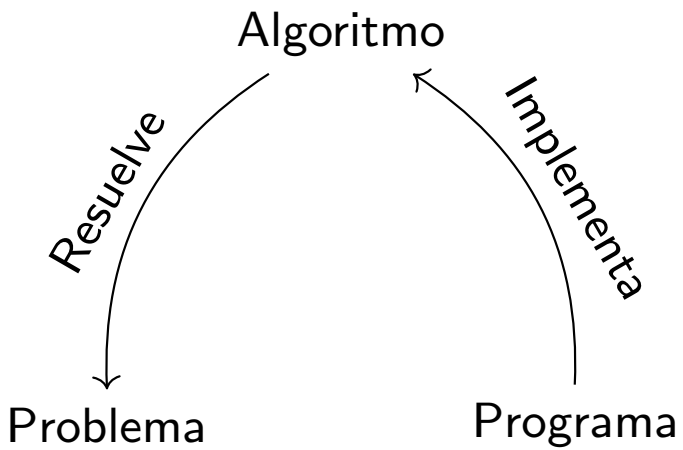
Algoritmo: Conjunto de pasos a seguir con las siguientes propiedades:

1. No ambiguos
2. Bien definidos
3. Finitos

Los algoritmos pueden tener entradas y salidas, que se obtienen al después de ejecutar el algoritmo con las entradas.

Programa: Conjunto de instrucciones escritas en algún lenguajes de programación. Dependen de entradas y al ejecutarse entrega salidas

Relación entre Problema, Algoritmo y Programa:



Existen distintos tipos de recurso computacionales y dependiendo de las limitaciones que existan será de interés uno u otro. Alguno de los principales recursos computacionales estudiados son:

- Tiempo (Número de operaciones).
- Espacio (Memoria (RAM/VRAM)).
- Número de procesadores.
- Información intercambiada entre procesadores (problemas distribuidos).

En este taller nuestro recurso de interés será el Tiempo, Número de procesadores y Memoria (en ese orden).

El tiempo:
Llamaremos operaciones elementales a un conjunto de operaciones que combinadas permita realizar cualquier otra operación en una máquina. Para nuestro estudio consideraremos operaciones elementales:

- Leer y escribir en una posición de un arreglo.
- Operaciones aritméticas elementales de dos números $(+, -, \cdot, :, ...)$.
- Comparaciones entre dos números$(<, >, =, ...)$.

El tiempo se medirá en la cantidad de operaciones elementales ejecutadas (no segundos).

El espacio (memoria):
Se mide en Bits. Se calcula como la suma de la memoria utilizada por todas las variables y arreglos.

El número de procesadores:
Se mide como la cantidad de procesadores asignados en un algoritmo.

En los tres casos, esta cantidad depende crecientemente del tamaño de la entrada.

Para una entrada de un arreglo de n elementos y un solo procesador, calcule el tiempo y memoria utilizados por el siguiente algoritmo:

Max(A)

Entrada: A un arreglo de largo m.

Salida: El elemento más grande de A.

```
1 i = 0
2 masGrande = A[1]
3 Mientras i ≠ m hacer
4 |   i = i + 1
5 |   Si A[i] > masGrande entonces
6 |   |   masGrande = A[i]
7 Devolver masGrande
```

Nota

En el ejemplo anterior, se ve que el tiempo depende de la entrada misma además del tamaño. Para evitar eso se toma como tiempo del peor caso, aquel que toma más tiempo.

Para clasificar algoritmos por diferencias importantes en la cantidad de recurso usado, introducimos la Notación de la $\mathcal{O}$ grande. En general f hará referencia al los recursos usados de un algoritmo (tiempo, memoria etc.) con entrada de tamaño n.

Definición ($\mathcal{O}$-Grande)

Dada una función $f : \mathbb{N} \to \mathbb{R}^+$ y una función $g : \mathbb{N} \to \mathbb{R}^+$, decimos que $f(n) \in \mathcal{O}(g(n))$ (se lee f es de **orden** g) si:

$$\exists C \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \geq n_0 : f(n) \leq Cg(n)$$

Nota 1

$\mathcal{O}(g(n))$ se interpreta como el conjunto de las funciones que *crecen a lo más* como $g(n)$.

Nota 2

$f(n) = \mathcal{O}(g(n))$ se puede interpretar como que, a partir de cierto n, g le gana a f.
Ejemplo: $10n + 10 = \mathcal{O}(n)$ y $1000n = \mathcal{O}(n^2)$.

Nota 3

Nos interesarán los algortimos polonomiales ($\mathcal{O}(n^k)$) y polilogaritmicos ($\mathcal{O}(\log^k(n))$).

Para una entrada de un arreglo de n elementos y un solo procesador, calcule el tiempo y memoria utilizados por el siguiente algoritmo usando la Notación de la $\mathcal{O}$ grande:

Max(A)

Entrada: A un arreglo de largo m.

Salida: El elemento más grande de A.

```
1 i = 0
2 masGrande = A[1]
3 Mientras i ≠ m hacer
4 |   i = i + 1
5 |   Si A[i] > masGrande entonces
6 |   |   masGrande = A[i]
7 Devolver masGrande
```

Definición (Cómputo paralelo)

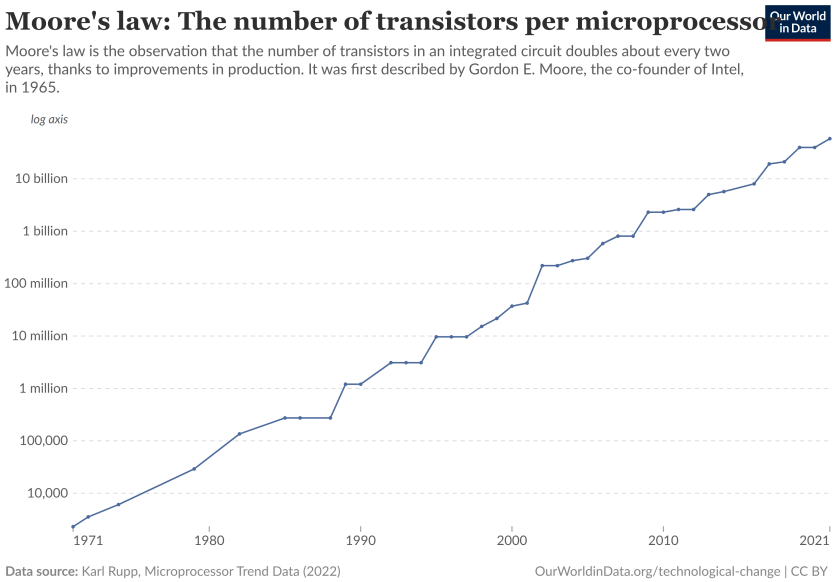
Es el cómputo que se realiza donde dos o más cálculos se realizan simultáneamente.

Definición (Cómputo paralelo)

Es el cómputo que se realiza donde dos o más cálculos se realizan simultáneamente.

Distintos motivos nos llevan a buscar formas alternativas de realizar computo:

- El "inminente" fin de la ley de Moore.
- Los límites físicos para los micro-procesadores (>5nm).
- Las barreras tecnológicas para el desarrollo de micro-procesadores.



Definición (Cómputo paralelo)

Es el cómputo que se realiza donde dos o más cálculos se realizan simultáneamente.

Distintos motivos nos llevan a buscar formas alternativas de realizar computo:

- El "inminente" fin de la ley de Moore.
  - Los límites físicos para los micro-procesadores (>5nm).
  - Las barreras tecnológicas para el desarrollo de micro-procesadores.
- Contextos "naturales" donde realizar cálculo de paralelo. (Juegos en linea)

Definición (Cómputo paralelo)

Es el cómputo que se realiza donde dos o más cálculos se realizan simultáneamente.

Distintos motivos nos llevan a buscar formas alternativas de realizar computo:

- El "inminente" fin de la ley de Moore.
  - Los límites físicos para los micro-procesadores (>5nm).
  - Las barreras tecnológicas para el desarrollo de micro-procesadores.
- Contextos "naturales" donde realizar cálculo de paralelo. (Juegos en linea)
- Calculo de voluntarios (SETI@home).

Definición (Cómputo paralelo)

Es el cómputo que se realiza donde dos o más cálculos se realizan simultáneamente.

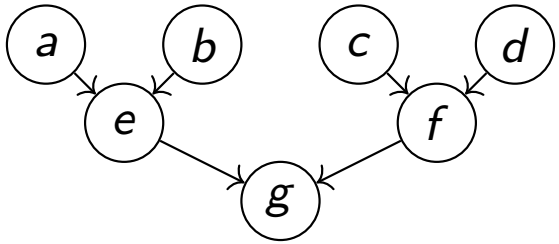
Distintos motivos nos llevan a buscar formas alternativas de realizar computo:

- El "inminente" fin de la ley de Moore.
  - Los límites físicos para los micro-procesadores (>5nm).
  - Las barreras tecnológicas para el desarrollo de micro-procesadores.
- Contextos "naturales" donde realizar cálculo de paralelo. (Juegos en linea)
- Calculo de voluntarios (SETI@home).
- Hardware paralelo más asequible. (procesadores multi-core/tarjetas gráficas)

Dada la flexibilidad de la definición de cómputo paralelo, existen distintos modelos, cada uno con un propósito específico.

- Modelo de grafos acíclicos dirigidos.
- Modelo distribuido.
- Modelo de memoria compartida.

Un grafo acíclico dirigido (DAG en inglés) es un digrafo sin caminos dirigidos que partan y terminen en el mismo vértice.



- Muchos algoritmos (paralelos) pueden explicarse y entenderse en este modelo fácilmente.
- Los vértices de grado de entrada 0 representan la entrada del algoritmo y aquellos de grado de salida 0 representan la salida.
- Los vértices intermedios están etiquetados con operaciones.
- Cada vértice puede ejecutarse independientemente de los otros si sus vecinos de entrada ya tienen calculada su información.

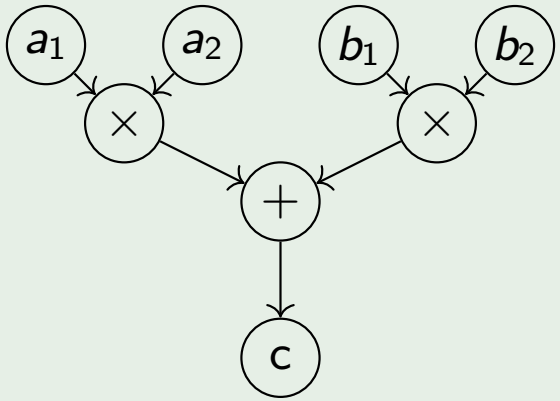
Un algoritmo está representado por una familia de DAGs, un DAG por cada tamaño de entrada posible.

Ventaja: Es fácil de representar, entender y calcular parámetros como en número de operaciones involucradas o el número de unidades de procesamiento requeridas.

Desventaja: Es difícil de implementar.

Ejemplo

El siguiente es un DAG que calcula el producto interior entre (a_1, a_2) y (b_1, b_2) y lo almacena en c.



Ejemplo

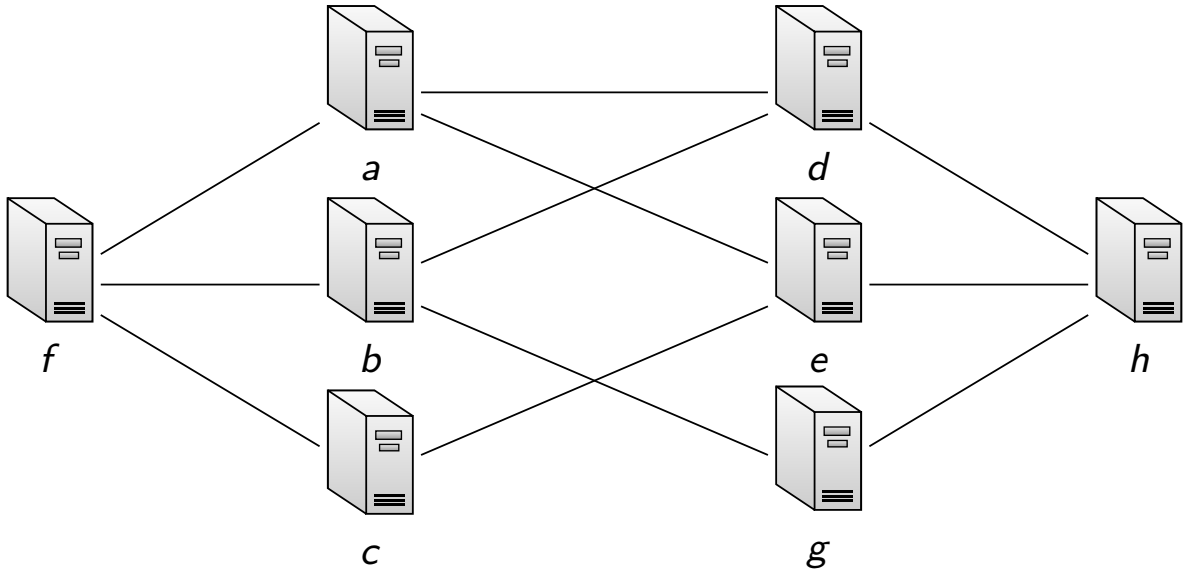
Ejercicios Para cada uno de los siguientes algoritmos, entradas y salidas, dibuje un DAG que lo represente. Las etiquetas de los vértices pueden ser $+, -, \times, /, \sqrt{}, ()^2$ y los números naturales.

1. El determinante de la matriz $A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix}$ donde la salida es el número d.
2. El inversa de la matriz $A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix}$ donde la salida es matriz $B = \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix}$ y $\det(A) \neq 0$.

Nota

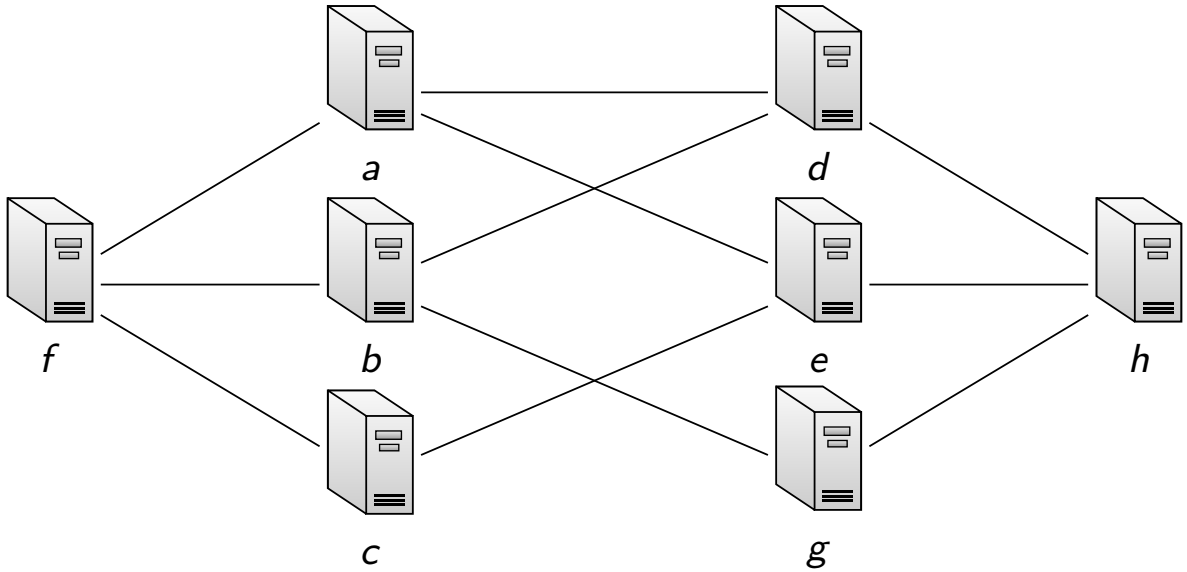
Note que, por ejemplo, el algoritmo para el cálculo de determinante no está representado por un DAG, está representado por una familia de DAGs, un DAG por cada posible tamaño de matriz.

En este modelo, cada unidad de procesamiento tiene su propia memoria y acceso a, en general, sólo a algunas otras unidades de procesamiento, formando una red.



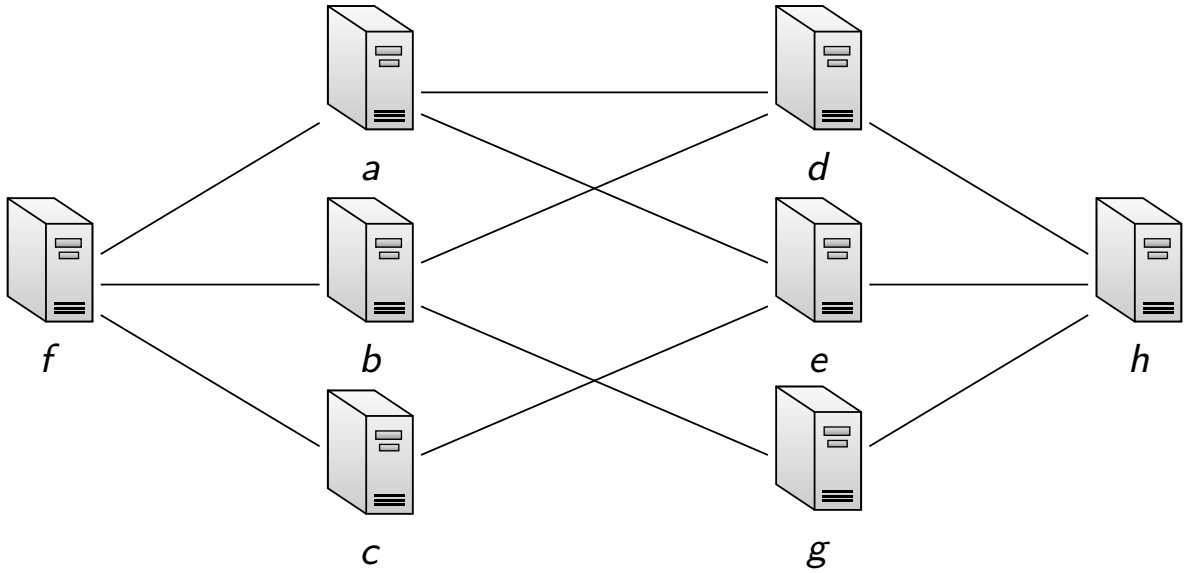
- El desarrollo de algoritmos depende de la topología de la red.

En este modelo, cada unidad de procesamiento tiene su propia memoria y acceso a, en general, sólo a algunas otras unidades de procesamiento, formando una red.



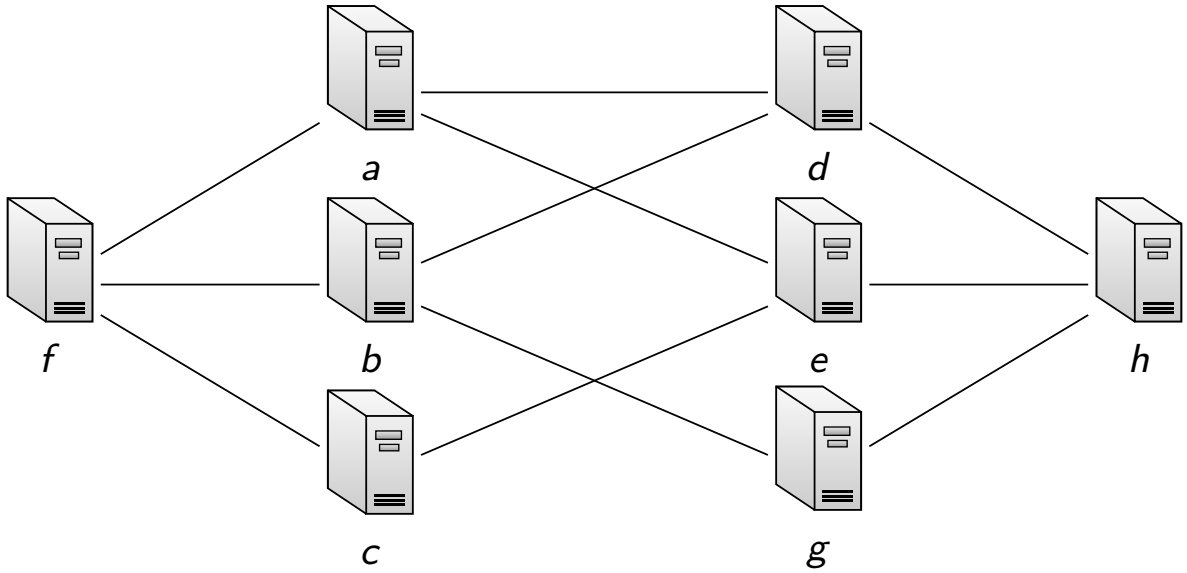
- El desarrollo de algoritmos depende de la topología de la red.
- Hay dos instrucciones nuevas de comunicación:

En este modelo, cada unidad de procesamiento tiene su propia memoria y acceso a, en general, sólo a algunas otras unidades de procesamiento, formando una red.



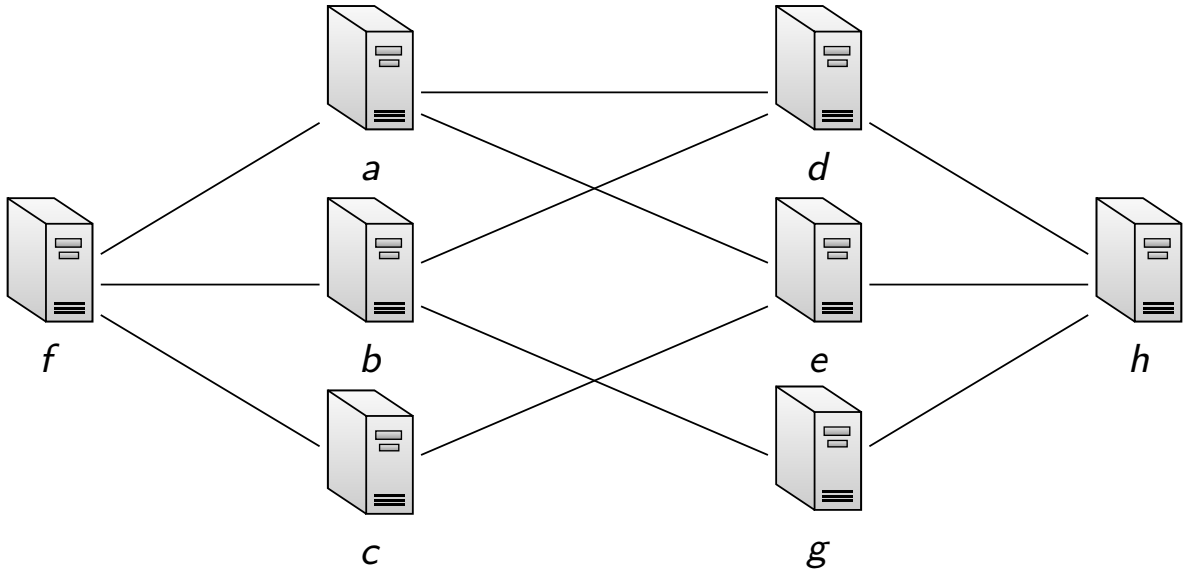
- El desarrollo de algoritmos depende de la topología de la red.
- Hay dos instrucciones nuevas de comunicación:
  - Enviar(A,p): Detiene la ejecución del procesador, entrega una copia de A a p y reanuda la ejecución.

En este modelo, cada unidad de procesamiento tiene su propia memoria y acceso a, en general, sólo a algunas otras unidades de procesamiento, formando una red.



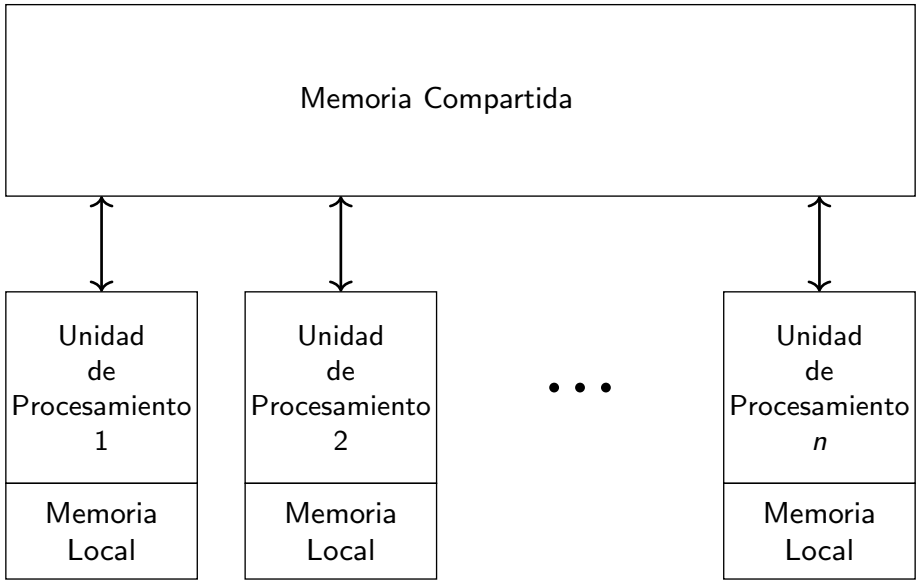
- El desarrollo de algoritmos depende de la topología de la red.
- Hay dos instrucciones nuevas de comunicación:
  - Enviar(A,p): Detiene la ejecución del procesador, entrega una copia de A a p y reanuda la ejecución.
  - Recibir(A,p): Detiene la ejecución del procesador, recibe una copia de A desde p y reanuda la ejecución.

En este modelo, cada unidad de procesamiento tiene su propia memoria y acceso a, en general, sólo a algunas otras unidades de procesamiento, formando una red.



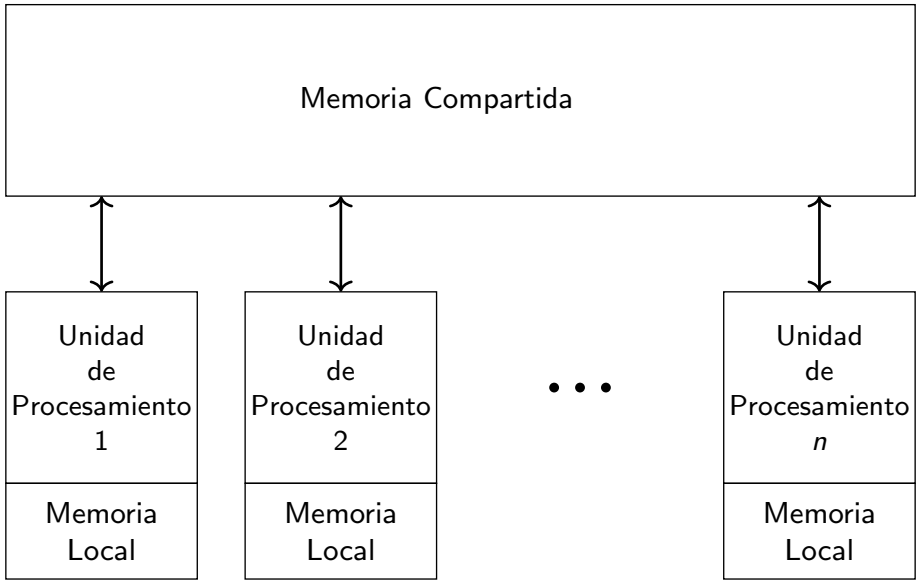
- El desarrollo de algoritmos depende de la topología de la red.
- Hay dos instrucciones nuevas de comunicación:
  - Enviar(A,p): Detiene la ejecución del procesador, entrega una copia de A a p y reanuda la ejecución.
  - Recibir(A,p): Detiene la ejecución del procesador, recibe una copia de A desde p y reanuda la ejecución.
- Una limitante es la comunicación entre las unidades de procesamiento.

En este modelo, cada unidad de procesamiento tiene acceso a una memoria compartida (o global) además de su memoria local, y con ella puede intercambiar información con el resto.



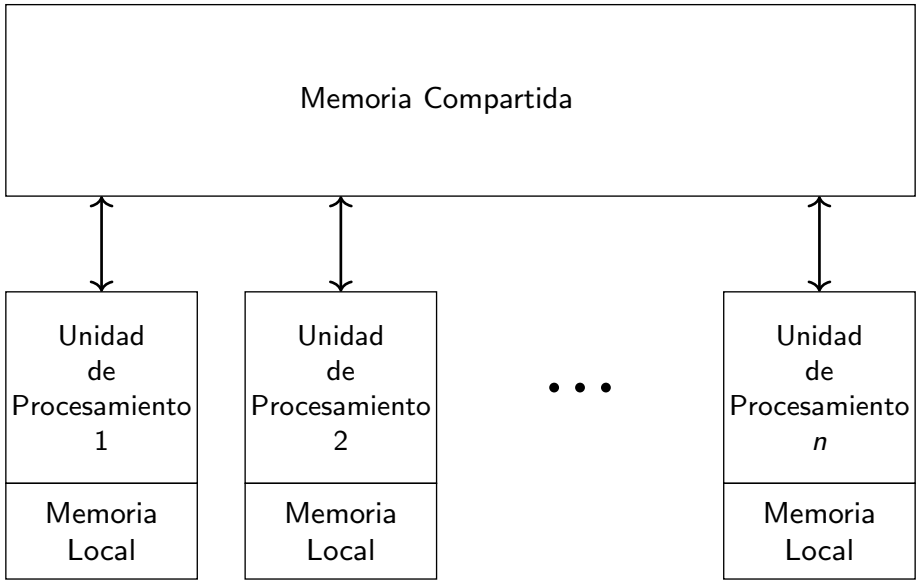
- Hay dos instrucciones nuevas asociadas a la memoria global:

En este modelo, cada unidad de procesamiento tiene acceso a una memoria compartida (o global) además de su memoria local, y con ella puede intercambiar información con el resto.



- Hay dos instrucciones nuevas asociadas a la memoria global:
  Lectura Global(A,B): Crea una copia de A desde la memoria compartida a B en la memoria local.

En este modelo, cada unidad de procesamiento tiene acceso a una memoria compartida (o global) además de su memoria local, y con ella puede intercambiar información con el resto.

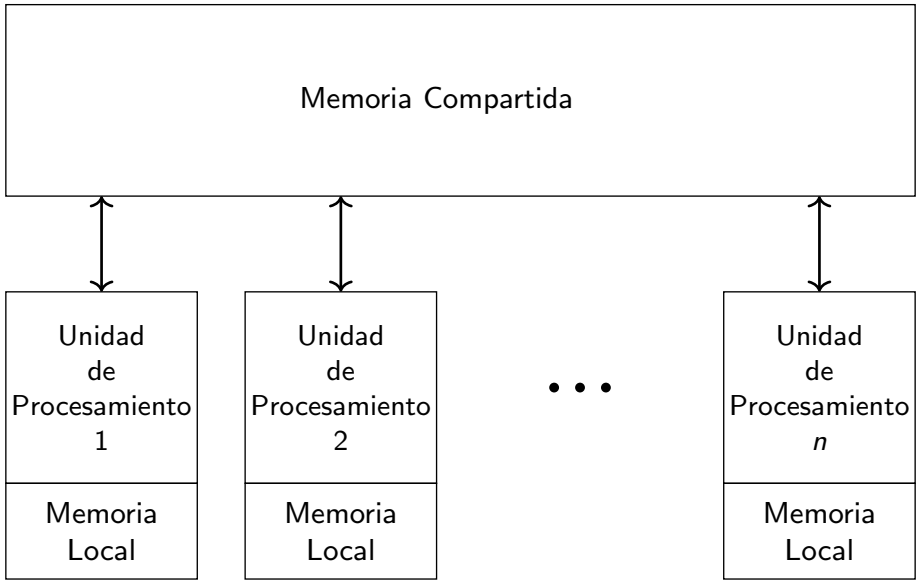


- Hay dos instrucciones nuevas asociadas a la memoria global:

  Lectura Global(A,B): Crea una copia de A desde la memoria compartida a B en la memoria local.

  Escritura Global(A,B): Crea una copia de A desde la memoria local a B en la memoria compartida.

En este modelo, cada unidad de procesamiento tiene acceso a una memoria compartida (o global) además de su memoria local, y con ella puede intercambiar información con el resto.



- Hay dos instrucciones nuevas asociadas a la memoria global:
  Lectura Global(A,B): Crea una copia de A desde la memoria compartida a B en la memoria local.
  Escritura Global(A,B): Crea una copia de A desde la memoria local a B en la memoria compartida.
- El procesamiento puede ser síncrono o asíncrono.

- Entre los distintos modelos de cómputo, elegiremos el modelo de memoria compartida, que llamaremos PRAM (*Parallel Random-Access Machine*).
- Trabajaremos con un sistema del tipo SIMD (*Single Instruction, Multiple Data*), es decir, cada procesador tiene la misma programación, pero pueden acceder a distintos datos.
- Asumiremos que los las unidades de procesamiento funcionan sincrónicamente. En la practica, en caso de no tener sincronia, se deben fijar los puntos de sincronización.

En la PRAM, las distintas unidades de procesamiento puede acceder a la memoria compartida, generando conflicto si dos unidades diferentes quieren acceder al mismo espacio de memoria compartida simultáneamente.
Esto induce a los siguientes sub modelos, basado en

- si la lectura puede ser simultanea[1] o exclusiva (*Concurrent* o *Excluive Read*) o
- si la escritura puede ser simultanea o exclusiva (*Concurrent* o *Excluive Write*).

Estos son:

EREW: exclusive read, exclusive write,

CREW: concurrent read, exclusive write,

ERCW: exclusive read, concurrent write,

CRCW: concurrent read, concurrent write.

En potencia, el orden sería EREW $<$ CREW $<$ CRCW.
En la practica si se tiene escritura simultanea es fácil tener lectura simultanea, por lo que ERCW se reemplaza por CRCW.

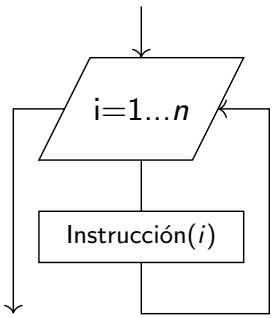
[1] o concurrente en algunos textos

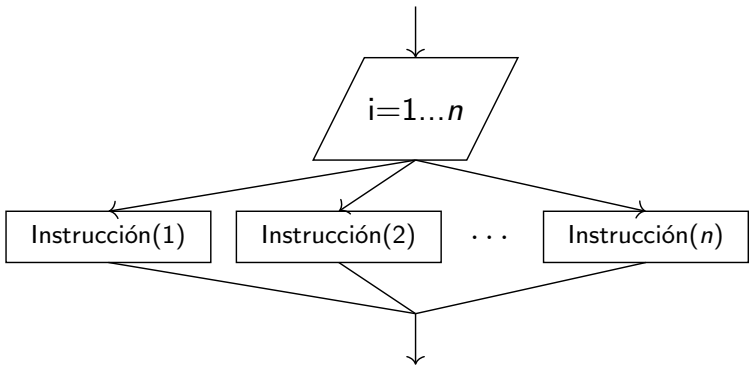
Para expresar que ciertas instrucciones se realizaran simultáneamente por varios procesadores, introduciremos la sentencia hacer en paralelo, también conocida como parfor o pardo (*parallel for* o *parallel do*)

Hacer en paralelo

1 **Para** $i = 1....n$ **hacer en paralelo**
2 | instrucciones

En este caso, para cada valor de i o v, un procesador ejecutará simultáneamente las instrucciones que aparecen debajo, como si fuera una sentencia "Para cada".





En una sentencia de este tipo, el número de operaciones realizadas es n, pero el tiempo es 1, pues las n instrucciones se realizaron simultáneamente. Por otro lado, el número de procesadores utilizados es n.

Definición

Dado un algoritmo paralelo con entrada de tamaño n, denotaremos $T(n)$ y $P(n)$ el tiempo paralelo y el número de procesadores que el algoritmos utiliza.

Ejemplo

Consideremos el siguiente algoritmo paralelo:

`Duplicador`(A)

Entrada: A un arreglo de largo n.

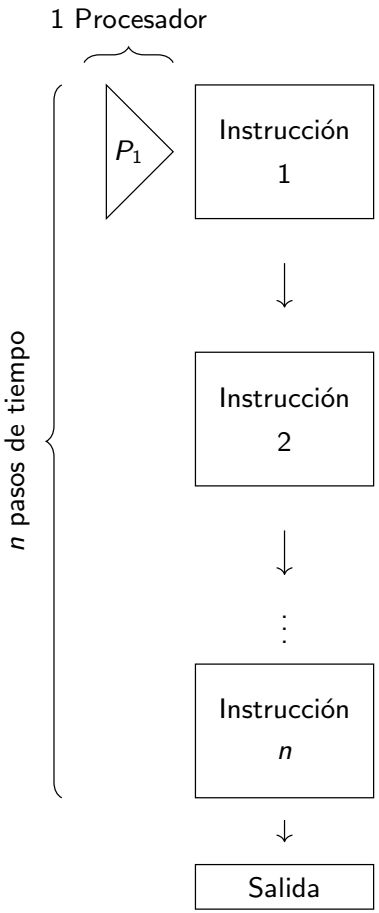
Salida: B, un arreglo donde cada entrada es el doble de esa entrada en A.

1 **Para** $i = 1....n$ **hacer en paralelo**
2 $\quad B[i] = 2A[i]$
3 **Devolver** B

Este algoritmo usa $\mathcal{O}(n)$ procesadores ($P(n) = \mathcal{O}(n)$) y tiempo 1 ($T(n) = \mathcal{O}(1)$). Si tenemos p procesadores, entonces el tiempo que toma es $T(n) = \mathcal{O}(n/p + 1)$ operaciones paralelas.

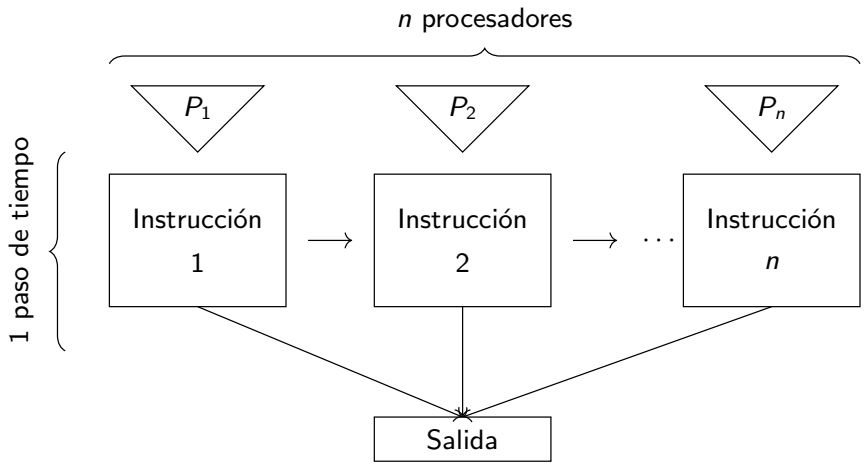
Modelo RAM

1 **Para** $i = 1....n$ **hacer**
2 | Instrucción i
3 **Devolver** "Salida"



Modelo PRAM

1 **Para** $i = 1....n$ **hacer en paralelo**
2 | Instrucción i
3 **Devolver** "Salida"



Ejemplo

Consideremos el siguiente algoritmo paralelo:

Entrada: Cadena de bits $w_1 ... w_n$.

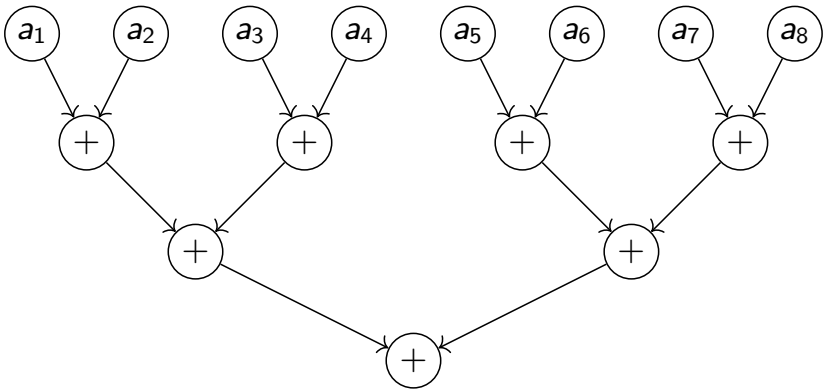
Salida: Es o no es una cadena de 0s y 1s intercalados. (0101010101... o 1010101010...)

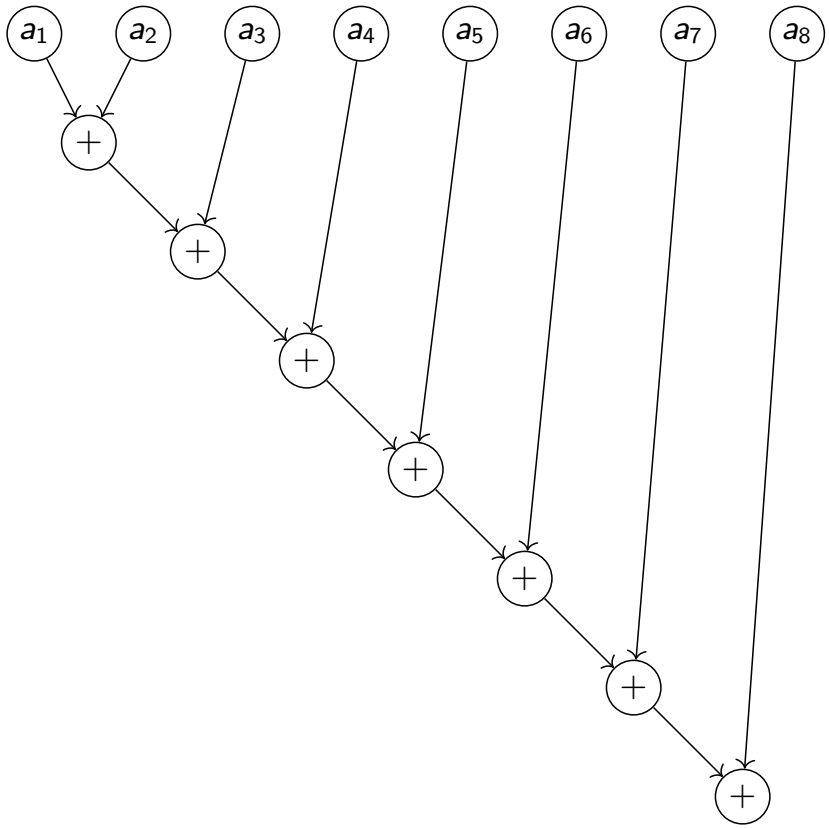
```
1 salida = "Es intercalada"
2 Para i = 1, ..., n − 1 hacer en paralelo
3 |   Si w_i = w_{i+1} entonces
4 |   |   salida = "No es intercalada"
5 Devolver salida
```

1. Determine la cantidad de recursos utilizados.
2. ¿Qué tipo de concurrencia debe admitir la PRAM que corra este algoritmo?

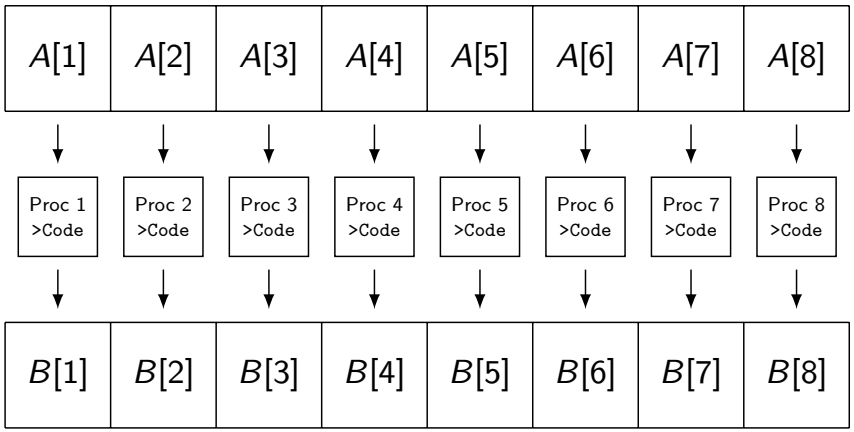
Técnicas de programación en la PRAM: división de datos

Las técnicas algorítmicas en maquinas PRAM se basan en la idea de aprovechar la gran cantidad de procesadores para obtener una mejora exponencial en los tiempos de cálculo. Esto se logra repartiendo eficientemente los procesadores en las tareas a realizar.





Una técnica simple es la de dividir los datos de entrada en partes independientes y luego aplicar un algoritmo componente a componente para encontrar la salida.
Esta técnica funciona cuando la naturaleza del problema permite la separación en partes independientes.



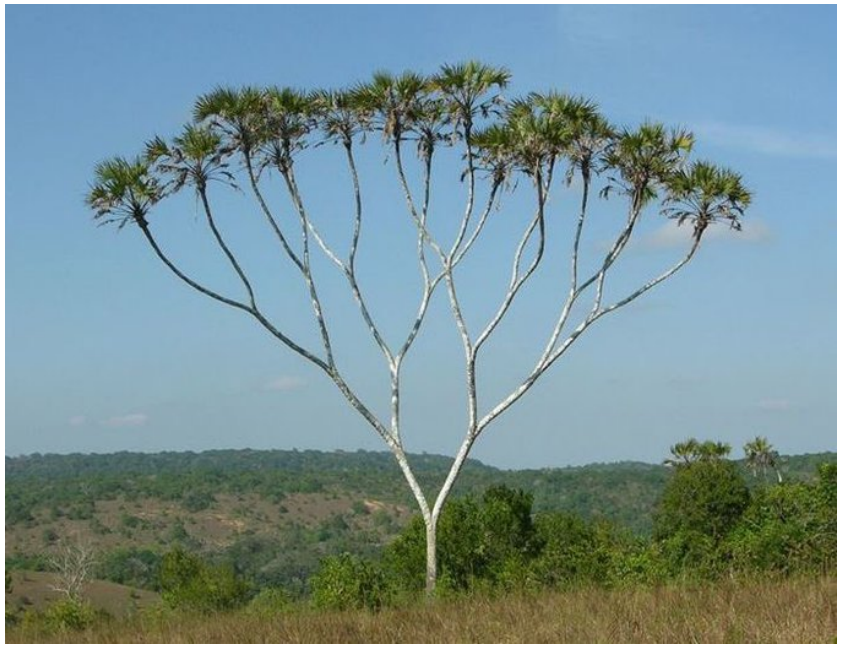
Duplicador(A)

Entrada: A un arreglo de largo n.

Salida: B, un arreglo donde cada entrada es el doble de esa entrada en A.

1 **Para** $i = 1....n$ **hacer en paralelo**
2 $\quad B[i] = 2A[i]$
3 **Devolver** B

El tiempo que toma es el tiempo que toma ejecutar un solo código independiente del tamaño de la entrada, teniendo suficientes procesadores.



Lema

Dado un árbol binario completo de $2^n - 1$, *su profundidad es* $n - 1$

Definición

Una operación binaria $* : A \times A \to A$ es asociativa si $\forall a, b, c \in A$, $(a * b) * c = a * (b * c)$

Ejemplos

Ejemplos de operaciones binarias asociativas:

- $+$, $\cdot$ en $\mathbb{R}$ y en $\mathcal{M}_n(\mathbb{R})$.
- máx, mín en $\mathbb{R}$
- $\wedge, \vee, \veebar$ en $\{0, 1\}$

La técnica del árbol balanceado consiste en resolver un problema mediante un algoritmo con un DAG que sea un árbol balanceado. La profundidad del árbol nos dará el tiempo en paralelo (logarítmico).

En particular nos va a interesar el siguiente problema

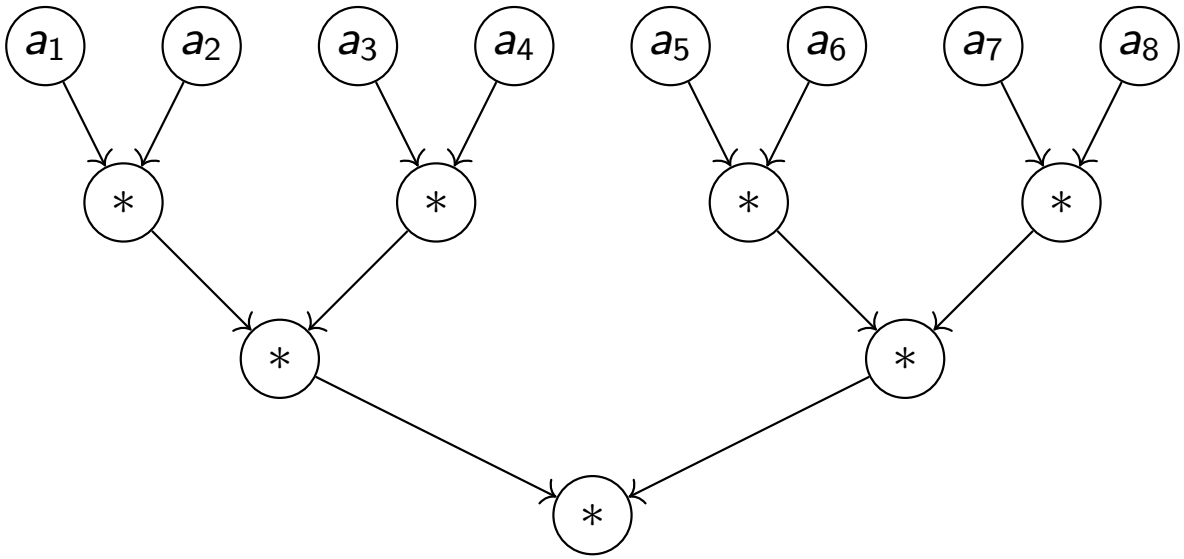
Suma de prefijos (Prefix-sum)

Dado un arreglo $[a_1, ..., a_n] \in A^n$ y $*$ una operación binaria interna asociativa en A. ¿Cuánto vale $a_1 * a_2 * ... * a_n$?

EL problema de Suma de Prefijos tiene una solución mediante un algoritmo paralelo eficiente que viene de la siguiente asociación de las operaciones.

$$a_1 * a_2 * a_3 * a_4 * a_5 * a_6 * a_7 * a_8 = [(a_1 * a_2) * (a_3 * a_4)] * [(a_5 * a_6) * (a_7 * a_8)]$$

que se representa en el siguiente DAG



pero necesitamos una versión en pseudo-código para la PRAM

SumaDePrefijosRecursiva(a, n)

Entrada: $a = [a_1, ..., a_n] \in A^n$, $n = 2^l$

Salida: $a_1 * ... * a_n$

```
1 Si n = 1 entonces
2 |   Devolver a_1
3 Para i = 1....n/2 hacer en paralelo
4 |   y_i = a_{2i-1} * a_{2i}
5 SumaDePrefijosRecursiva(y, n/2)
```

Teorema

El algoritmo de SumaDePrefijosRecursiva:

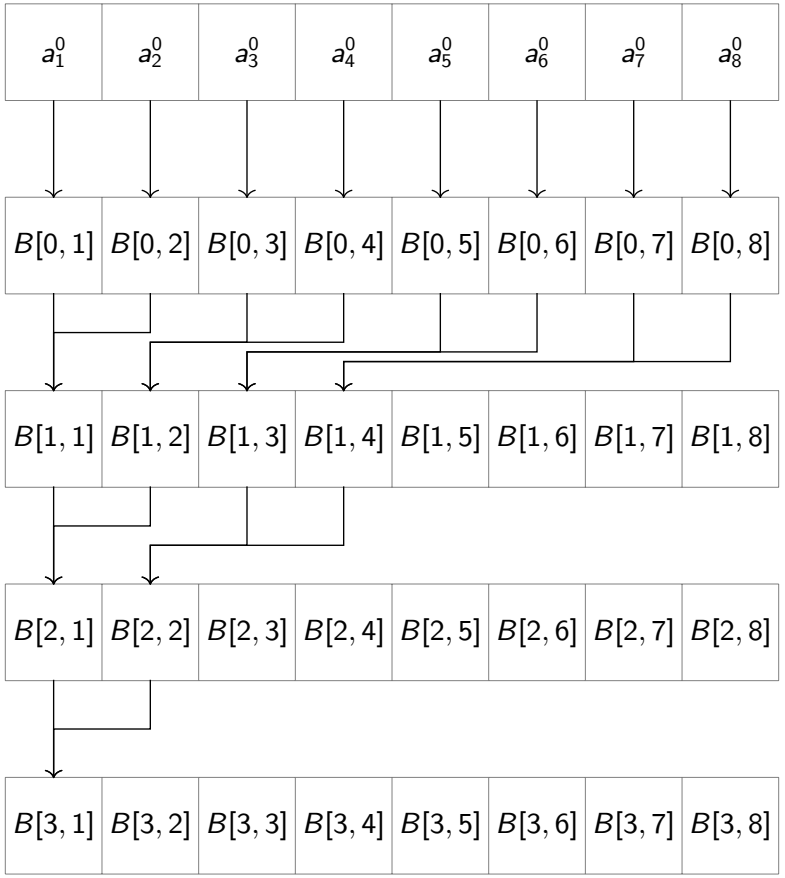
- *es correcto.*
- *corre en tiempo $\mathcal{O}(\log n)$ y usa $\mathcal{O}(n)$ procesadores.*
- *modelo EREW.*

SumaDePrefijos(a, n)

Entrada: $a = [a_1, ..., a_n] \in A^n$, $n = 2^l$

Salida: $a_1 * ... * a_n$

1 **Para** $i = 1...n$ **hacer en paralelo**
2 $\quad B[0, i] = a[i]$
3 **Para** $t = 1....l$ **hacer**
4 $\quad$ **Para** $i = 1....2^{l-t}$ **hacer en paralelo**
5 $\quad\quad B[t, i] = B[t-1, 2i-1] * B[t-1, 2i]$
6 **Devolver** $B[l, 1]$



Ejemplo (Ejercicio)

Construya un algoritmo paralelo que resuelva los siguientes problemas:
Suma (a): Dado un arreglo $a = [a_1, ..., a_n] \in \mathbb{R}^n$ ¿Cuánto vale $a_1 + a_2 + ... + a_n$?
Producto (a): Dado un arreglo $a = [a_1, ..., a_n] \in \mathbb{R}^n$ ¿Cuánto vale $a_1 \cdot a_2 \cdot ... \cdot a_n$?
Máximo (a): Dado un arreglo $a = [a_1, ..., a_n] \in \mathbb{R}^n$ ¿Cuánto vale $\max\{a_1, ..., a_n\}$?
Mínimo (a): Dado un arreglo $a = [a_1, ..., a_n] \in \mathbb{R}^n$ ¿Cuánto vale $\min\{a_1, ..., a_n\}$?

Teorema

Sean $a, b \in \mathbb{R}^n$, entonces hay un algoritmo que calcula $a \cdot b$ y toma $t(n) = \mathcal{O}(\log n)$ y usa $p(n) = \mathcal{O}(n)$ en una PRAM EREW.

Demostración.

ProductoPunto(a, b)

Entrada: $a = [a_1, ..., a_n], b = [b_1, ..., b_n] \in \mathbb{R}^n$, $n = 2^l$

Salida: $a \cdot b$

1 **Para** $i = 1...n$ **hacer en paralelo**
2 $\quad c[i] = a[i] \cdot b[i]$
3 **Devolver** Suma(c) ; ▷ Ver ejercicio anterior

□

Ejemplo (Ejercicio)

Dibuje el DAG del algoritmo ProductoPunto(a, b).

Teorema

Sean $A \in \mathcal{M}_{n,m}(\mathbb{R})$ y $B \in \mathcal{M}_{m,l}(\mathbb{R})$ dos matrices compatibles para la multiplicación, entonces hay un algoritmo paralelo que calcula $A \cdot B$ y toma $t(N) = \mathcal{O}(\log N)$ y usa $p(N) = \mathcal{O}(N^2)$ en una PRAM CREW, donde N es el tamaño de la entrada.

Demostración.

ProductoMatricial(A, B)

Entrada: $A \in \mathcal{M}_{n,m}(\mathbb{R})$ y $B \in \mathcal{M}_{m,l}(\mathbb{R})$

Salida: $A \cdot B$

1 **Para** $(i,j) \in \{1,...,n\} \times \{1,...,l\}$ **hacer en paralelo**
2 $\quad C[i,j]$=ProductoPunto($A[i,:], b[:,j]$)
3 **Devolver** C ; ▷ Ver ejercicio anterior

□

Ejemplo (Ejercicio)

Dibuje el DAG del algoritmo ProductoMatricial(A, B) para $A, B \in \mathcal{M}_n(\mathbb{R})$ con $n = 2$ y $n = 3$.

Ejemplo (Ejercicio)

Construya un algoritmo paralelo que resuelva los siguientes problemas:
Potencia (A,p) Dado una matriz $A \in \mathcal{M}_n(\mathbb{R})$ ¿Cuánto vale A^p? **Clausura transitiva (A)** Dado una matriz $A \in \mathcal{M}_n(\{0,1\})$ ¿Cuánto vale A^T?