



UNIVERSIDAD DE COSTA RICA

# INTRODUCCIÓN A LA COMPUTACIÓN DE ALTO RENDIMIENTO (HPC)

Prof. Marlon Brenes y Prof. Federico Muñoz  
Escuela de Física, Universidad de Costa Rica

---

# Introducción

---

- El concepto de HPC (*high-performance computing*) abarca muchas áreas de investigación. Para definir, podemos decir:
  - **HPC es el uso de servidores, *clusters* y supercomputadoras en conjunto con el software, herramientas, componentes, almacenamiento y servicios asociados para realizar tareas de análisis en el campo científico que requieren de un particular uso extenso de recursos de computación, memoria o manejo de datos**
- HPC es una metodología utilizada en la ciencia e ingeniería para efectos investigativos y producción en la industria, gobierno y academia.

---

# Elementos

---

- Hardware

- 
- Servidores, clusters y supercomputadoras

- Software

- 
- Herramientas, bibliotecas, componentes y servicios

- Problemas


- 
- Problemas a resolver de índole científica o analítica

---

# Infraestructura

---

- **Hardware**

- 
- Servidores, nodos, aceleradores (e.g. GPUs)
  - Redes de alta velocidad
  - Almacenamiento paralelo de alta gama

- **Software**

- 
- Bibliotecas especializadas

- **Problemas**

- 
- Datos técnicos a ser investigados
  - Generación de datos
  - Problemas difíciles/imposibles de resolver de manera analítica

---

# HPC




---

A pesar de que el HPC abarca muchas áreas del ámbito científico y computacional, para efectos prácticos, nuestra aplicación será en la **computación paralela**

---

# Cuando es requerido el uso de HPC?

---

- • Mi problema computacional toma demasiado tiempo para ser resuelto, por lo que necesito más unidades de procesamiento para atacar el problema en un periodo manejable
- • Mi problema es demasiado grande, por lo cual necesito cantidades grandes de memoria para resolverlo
- • Los datos generados por mi problema son demasiado grandes, por lo que requiero grandes cantidades de almacenamiento para depurarlos

---

# Cuando es requerido el uso de HPC?

---


- Algunos ejemplos:



- Simulaciones de muchos cuerpos en ciencia cuántica
- Dinámica molecular
- Simulaciones de Monte Carlo
- Química Cuántica Computacional
- Bioinformática
- Dinámica de Fluidos Computacional
- Ciencia de datos y *machine learning*

# Paralelismo

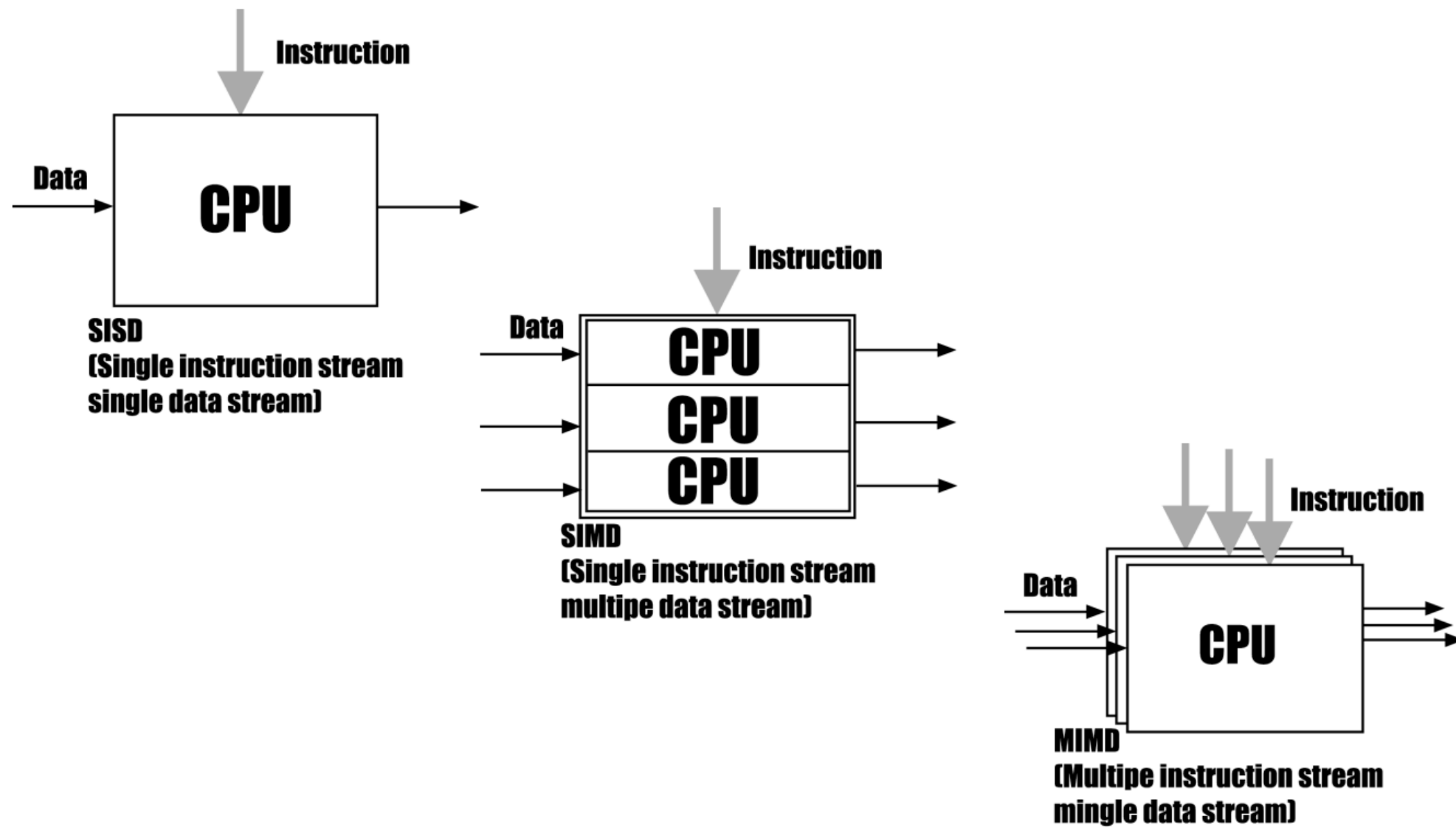
- El paralelismo puede ser categorizado de acuerdo con la taxonomía de Flynn (1966)

- 
- La clasificación se establece de acuerdo a:
    - El flujo de datos
    - El flujo de instrucciones

| Nombre | Flujo de Instrucciones | Flujo de Datos |
|--------|------------------------|----------------|
| SISD   | Singular               | Singular       |
| SIMD   | Singular               | Múltiple       |
| MIMD   | Múltiple               | Múltiple       |
| MISD   | Múltiple               | Singular       |



# Paralelismo



# Paralelismo

- SISD



- Esta corresponde a la computación más simple (la idea de la máquina de con Neumann), en la cual una instrucción se ejecuta sobre un solo operando. Todo ocurre de manera secuencial.

- SIMD



- A este modelo se le conoce como “computación vectorial”. Arquitecturas actuales (e.g. Intel Vector Instructions)

- MISD



- Básicamente un modelo de carácter académico. Un modelo en el cual se realizan múltiples operaciones sobre el mismo operando. De momento sin aplicaciones prácticas

- MIMD



- Este modelo requiere unidades de procesamiento independientes actuando sobre diferentes operandos. Esta forma de paralelismo es la que queremos alcanzar, pero con procesos en sincronía.

---

# Paralelismo

---

- El modelo de Flynn es un poco anticuado...
  - Estas cuatro formas de dividir el paralelismo se desarrolló inicialmente para clasificar todas las formas de paralelismo
- En los años posteriores a esta idea hemos:
  - Creado sistemas que no se acoplan formalmente a estos conceptos
  - Desarrollado la necesidad de vocabulario adicional que no está cubierto bajo este esquema
- Sin embargo, estos conceptos muy básicos en el HPC y debemos conocerlos. En particular, el modelo SIMD se implementa en la arquitectura computacional actual

# Paralelismo actual

- El paralelismo actual se clasifica en dos clases:

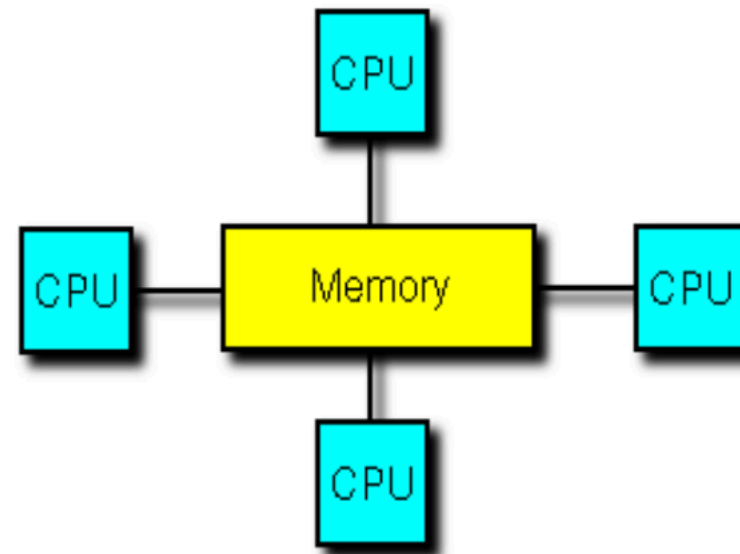
```
graph TD; A[El paralelismo actual se clasifica en dos clases:] --> B[Paralelismo de Memoria Compartida (Shared memory parallelism)]; A --> C[Paralelismo de Memoria Distribuida (Distributed memory parallelism)];
```

Paralelismo de Memoria Compartida  
(Shared memory parallelism)

Paralelismo de Memoria Distribuida  
(Distributed memory parallelism)

- Diferencia: la principal diferencia corresponde a si cada elemento de computación puede acceder los espacios de memoria del otro sin hacer una solicitud de acceso

# Memoria compartida



- En el paradigma de memoria compartida los elementos de cómputo tienen acceso a un banco de memoria que todos comparten
- ↪
- Bajo este paradigma, un proceso se particiona en varias imágenes que se mapean sobre elementos físicos de computación (procesadores, usualmente)
  - El modelo de programación en este caso se realiza mediante **hilos (threads)** que corresponden a las imágenes de los procesos
  - La sincronización es muy importante

# Memoria compartida

- Pros



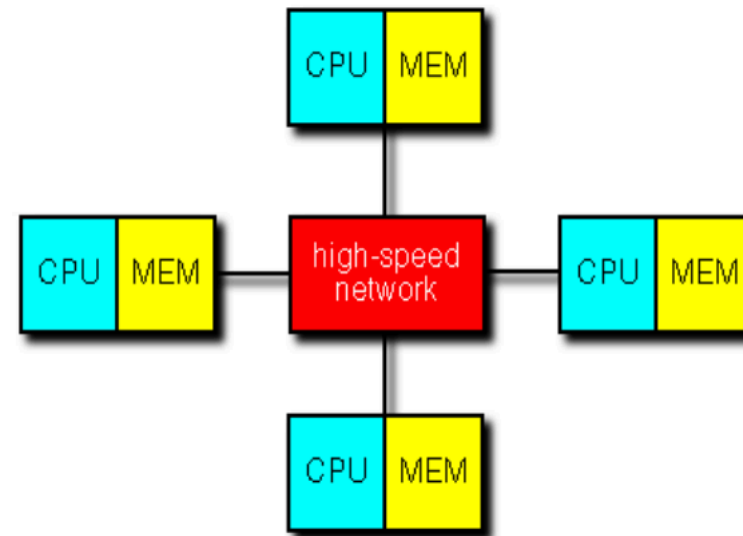
- Los hilos nada más mantienen una copia de la sección de memoria que comparten y todos trabajan juntos para modificarla
- Esto se puede realizar siempre y cuando se pueda coordinar cual hilo puede modificar en cierto punto de tiempo
- Existen bibliotecas que nos permiten generar y manipular las operaciones que cada hilo realiza de manera automática, a pesar de que se puede realizar el control manual

- Cons



- Generar hilos es un proceso de alto costo computacional
- Los hilos deben vivir dentro del espacio de memoria de un proceso y el sistema operativo debe manejar dicha memoria. Como consecuencia, no se pueden **generar en distintas computadoras**
- Podemos solamente generar los hilos que podamos contener en una sola máquina

# Memoria distribuida



- En el paradigma de memoria distribuida los elementos de cómputo tienen acceso a un banco de memoria independiente. Para intercambiar su información, un protocolo de pase de mensajes debe ser establecido a través de una red.
- ↪
- Bajo este paradigma, se generan distintos procesos lógicos que son asignados a elementos de computación (usualmente procesadores)
  - El modelo de programación en este caso se realiza mediante **procesos**
  - La sincronización es muy importante
  - La comunicación entre procesos se realiza manualmente

---

# Memoria compartida

---

- Pros



- Podemos generar la cantidad de procesos que necesitemos, incluso viviendo en computadoras distintas con IDs determinados que se comunican entre ellos
- La generación de procesos es tiene un costo computacional bajo
- Existen bibliotecas para manejar el protocolo de comunicación entre procesos

- Cons




- La comunicación entre procesos no es automática. Debe ser manipulada por el programador
- La transmisión de datos implica que se deben mantener copias entre procesos (usualmente no es tanto un problema)
- Es usualmente más complicado de trabajar para el programador



---

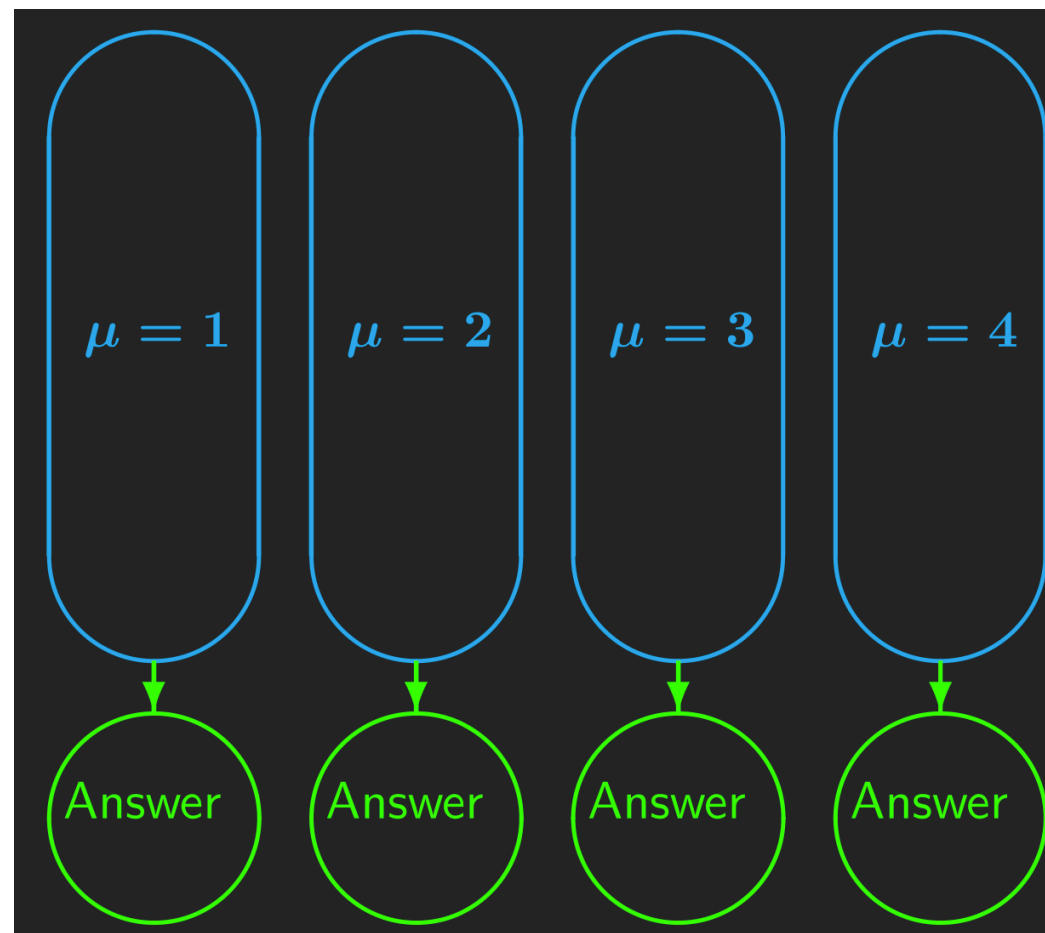
# Concurrencia

---

- La eficiencia de una aplicación ejecutada en un ambiente en paralelo depende del nivel de concurrencia
  - • La concurrencia se refiere a las secciones de la aplicación cuya ejecución se puede realizar de manera independiente por cada uno de los elementos de procesamiento
  - La idea es escribir algoritmos que contengan muchas partes que puedan ser ejecutadas de manera concurrente, sin importar el orden de ejecución
  - En general, la dependencia de datos entre unidades de procesamiento limita la concurrencia

# Paralelismo linear concurrente (escaneo de parámetros)

- La idea es obtener resultados de un modelo variando sus parámetros
- Cada uno de los parámetros se ejecuta sobre una unidad de procesamiento
- De esta forma se pueden evaluar muchos parámetros en serie al mismo tiempo, i.e., de manera concurrente
- Es la forma más básica de uso de procesamiento en paralelo

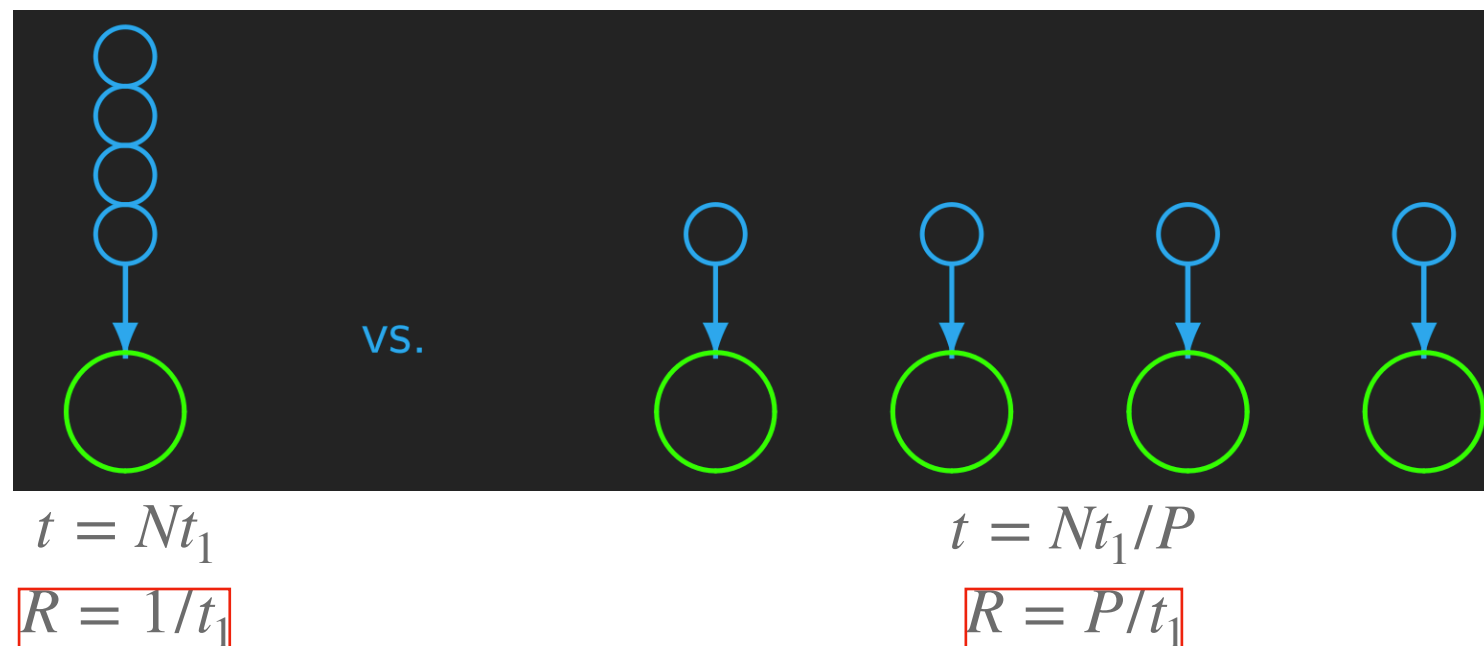


# Rendimiento

- En el modelo de paralelismo lineal concurrente, evaluar el rendimiento es trivial

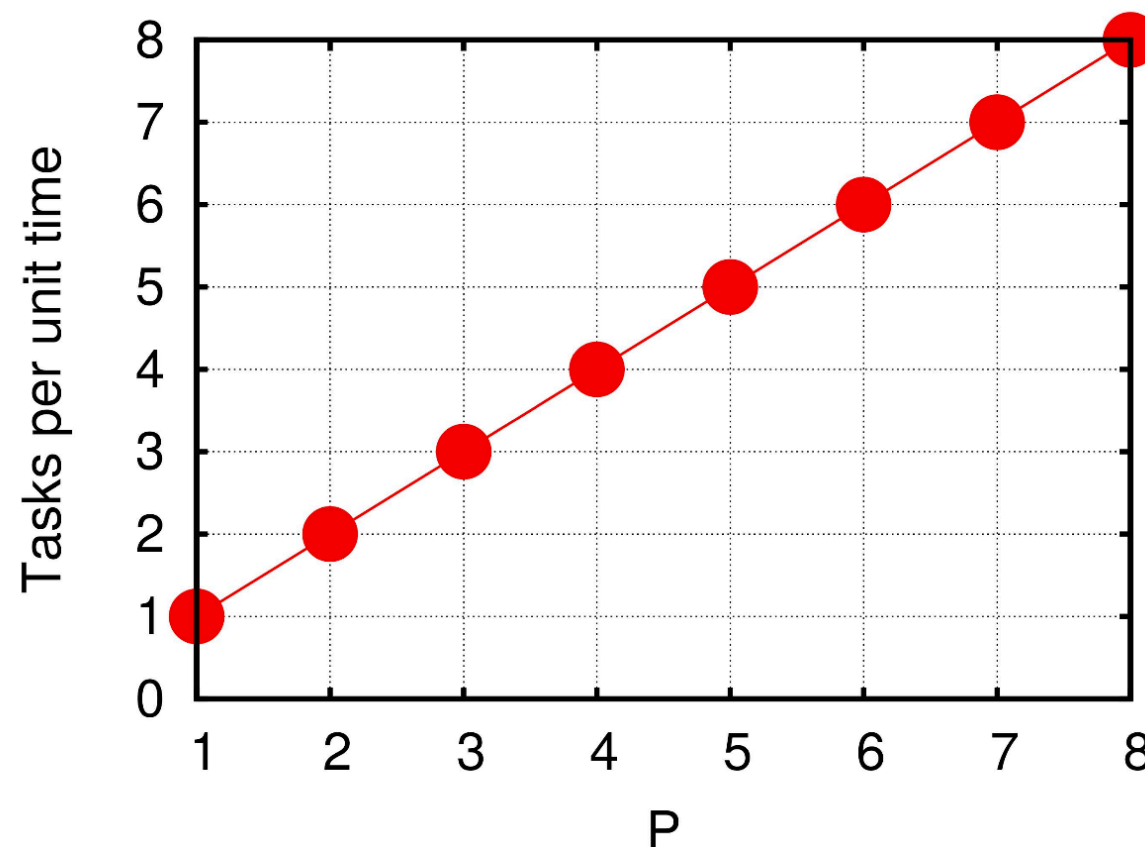
$$R = \frac{N}{t} \rightarrow \text{El rendimiento es el número de tareas por unidad de tiempo}$$

- Para maximizar el rendimiento, debemos realizar la mayor cantidad de tareas de manera concurrente.
- reducir el tiempo de ejecución de una tarea, lo cual usualmente está fuera de nuestro control después de optimizar el algoritmo
- Para operaciones lineares concurrentes, el rendimiento es trivial: usando  $P$  unidades de procesamiento, el rendimiento  $R$  escala por un factor de  $P$



# Escalabilidad

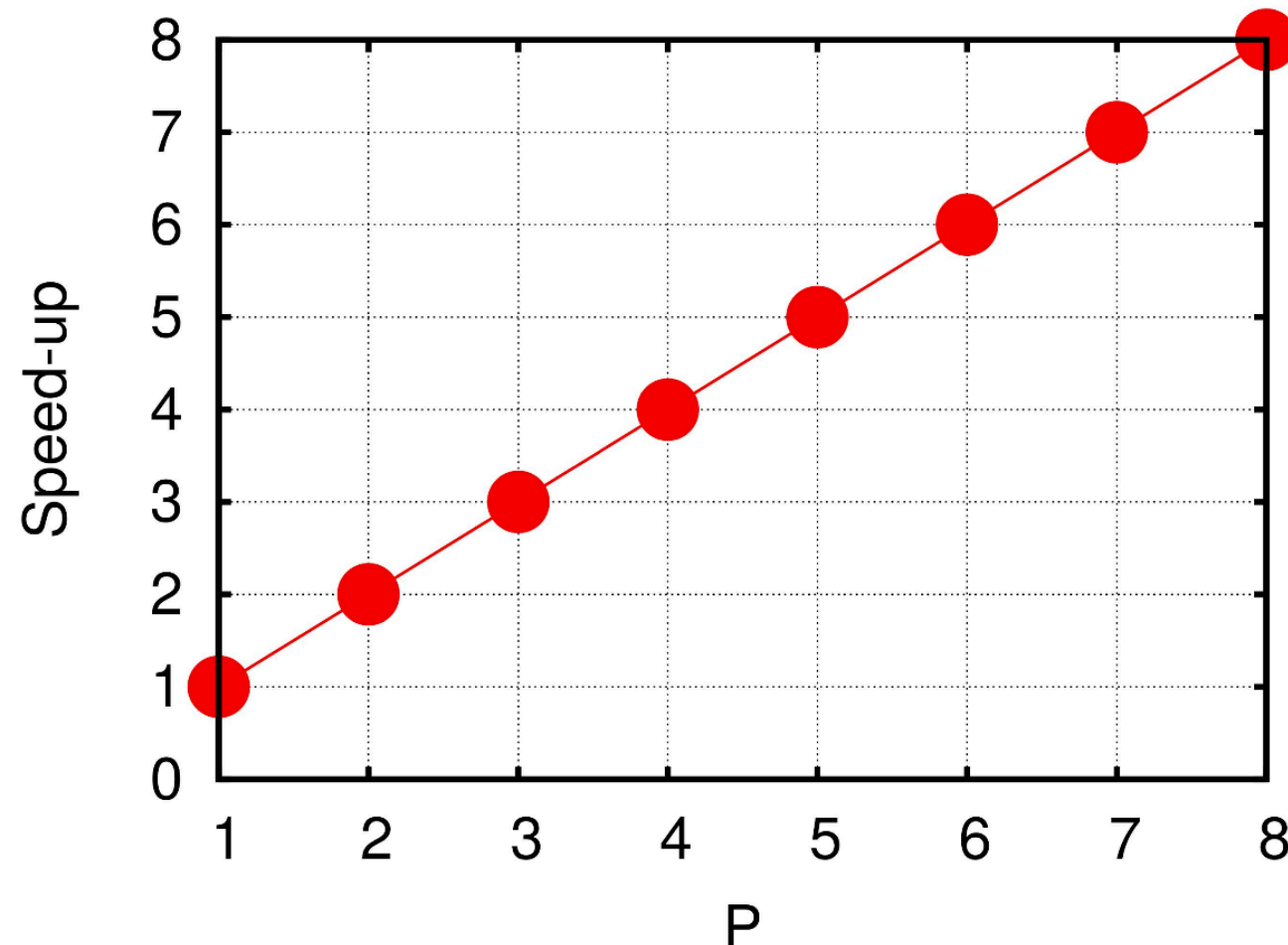
- La escalabilidad se refiere a como incrementa el rendimiento conforme el número de unidades de computación incrementa
- Para el caso de paralelismo linear concurrente,  $R \propto P$
- A esto se le conoce como escalabilidad lineal o perfecta



# Aceleración (*speedup*)

- La aceleración se refiere a cuánto es más rápido resolver el problema conforme aumenta el número de unidades de procesamiento
- Se mide utilizando el tiempo que toma resolver el problema en serie (con solo una unidad de procesamiento) dividido por tiempo en paralelo

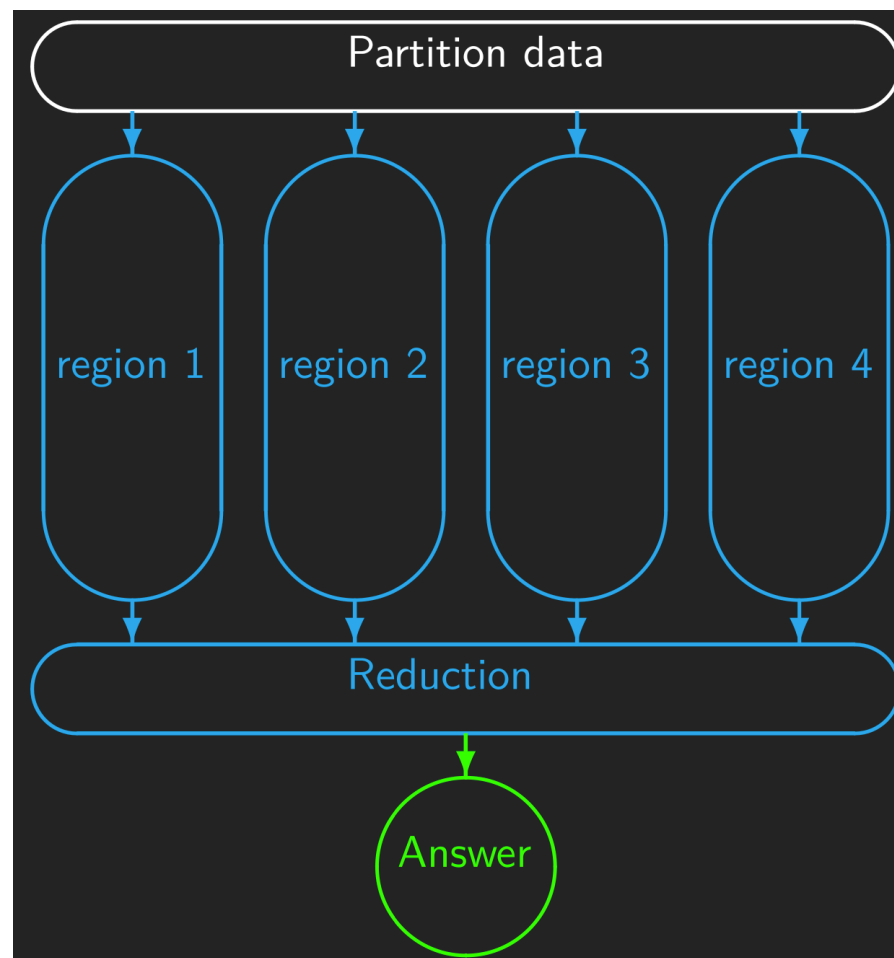
$$S = \frac{t_{\text{serial}}}{t(P)}$$



- La ejecución linear concurrente da a lugar a aceleración lineal
- Existen aplicaciones donde la aceleración es casi lineal si la sección no concurrente de la aplicación es muy pequeña (e.g., Monte Carlo)

# Paralelismo no ideal

- Existen tres casos generales diferentes en problemas de paralelismo:
  - Problemas de paralelismo lineal concurrente
  - Problemas de paralelismo trivial
  - Problemas de paralelismo no ideal
- En los problemas de paralelismo trivial se puede alcanzar aceleración casi lineal dado que existe muy poca comunicación entre elementos de computación
- El caso más general es el problema de paralelismo no ideal



$t_s \rightarrow$  Tiempo que tarda la parte serial (partición + reducción)

$t_p \rightarrow$  Tiempo que tarda la parte que se puede paralelizar

# Ley de Amdahl

- En el caso no ideal, la aceleración corresponde a

$$S = \frac{t_P + t_s}{(t_P/P) + t_s}$$

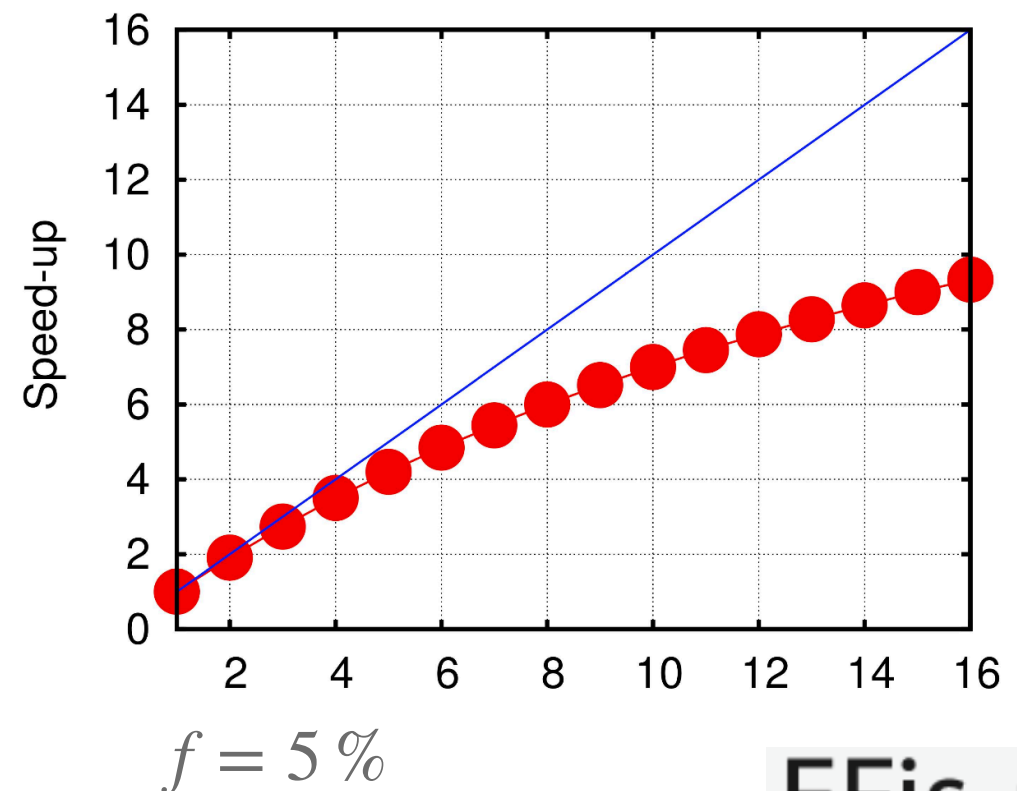
- Podemos definir  $f = \frac{t_s}{t_P + t_s}$  como la fracción serial, tal que

$$S = \frac{1}{f + (1 - f)/P}$$

- Nótese que

$$\lim_{P \rightarrow \infty} S = \frac{1}{f}$$

- Lo cual implica que de manera asintótica, la parte serial siempre domina. La aceleración siempre va a estar limitada por la parte serial sin importar P.



---

# Paralelismo no ideal: no localidad

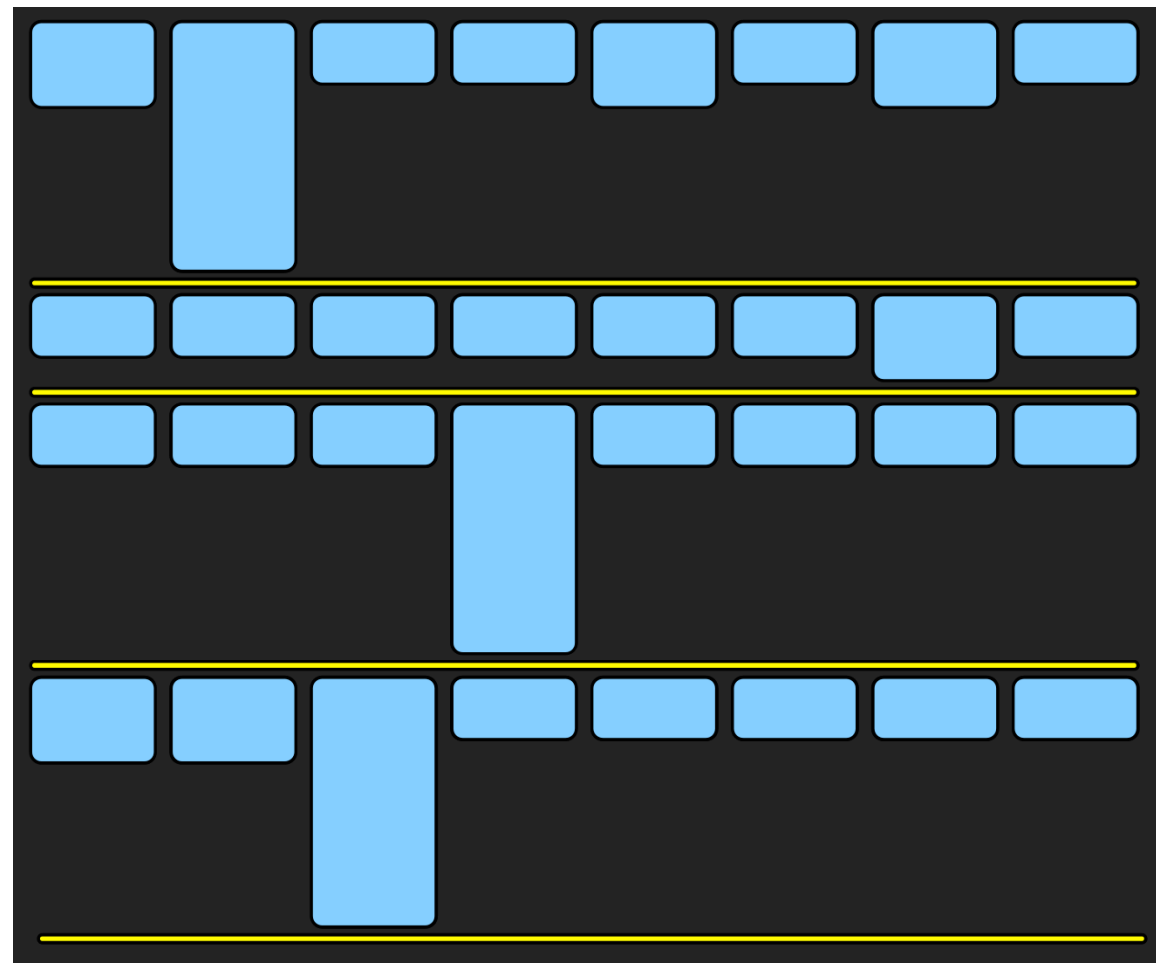
---

- Uno de los desafíos más importantes en la computación en paralelo se refiere a la localidad de elementos en memoria
  - En algoritmos de memoria distribuida, la información entre elementos de procesamiento se comunica mediante interfaces de paso de mensajes
  - La comunicación es aproximadamente tres órdenes de magnitud más lenta que los cálculos con números flotantes
- Esto se puede aliviar, en parte, con hardware que esconden la localidad usando caches o movimiento automático de datos
- Sin embargo, **la comunicación no puede ser completamente evitada**
  - Un mejor diseño evita comunicación en la medida de lo posible
  - Memoria compartida: generando datos locales a cada hilo
  - Memoria distribuida: previniendo la mayor cantidad de datos que cada proceso pueda almacenar



# Paralelismo no ideal: balance de carga

- El balance de carga también es un concepto de diseño importante de considerar
- Se debe realizar a nivel de la carga de computación que cada elemento de procesamiento debe trabajar
- Puede ser poco trivial determinar esto. Se suelen usar pruebas para determinar y balancear las cargas



---

# Demostración

---

- Acceso y arquitectura del cluster del CICIMA:
  - <https://sites.google.com/view/cluster-cicima/home>