Sesión 9.2 – CS3102 EDA

Prof. Victor Flores

13 de Octubre de 2022

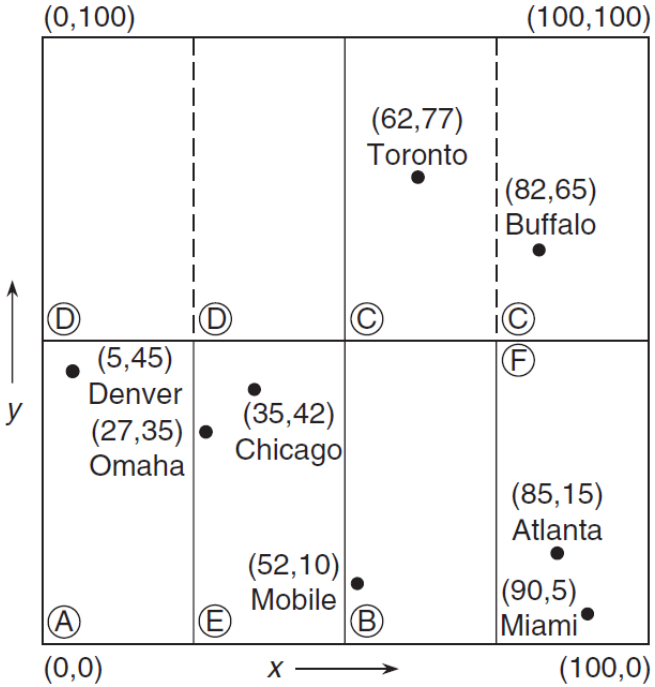
UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Contenido

- EXCELL
- Programación paralela

EXCELL

EXCELL

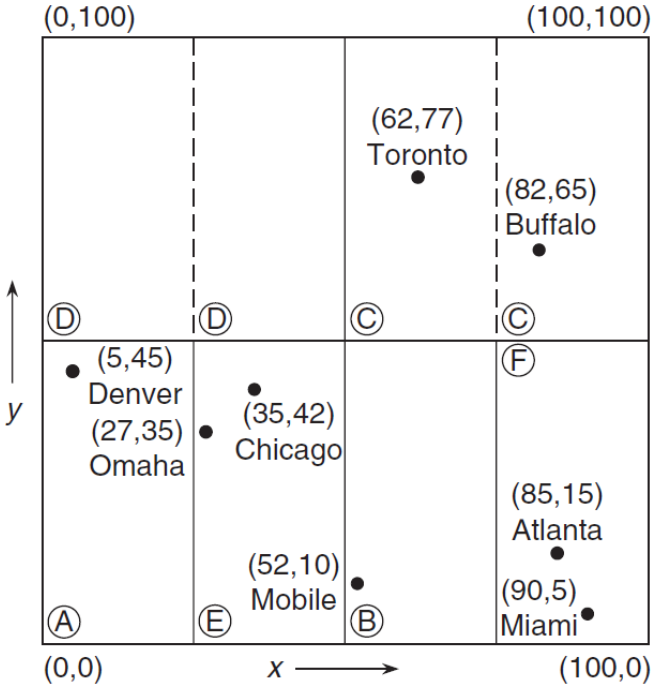


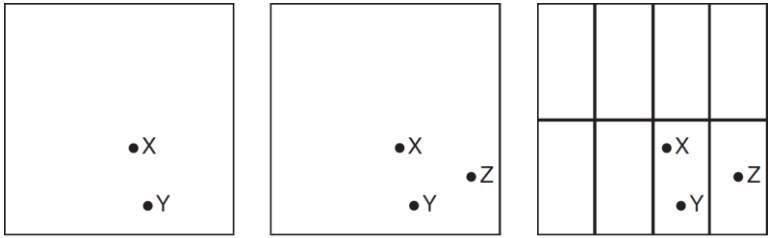
Split

Todos los intervalos, en la dimensión del *split*, se dividen en dos. Como resultado, el tamaño del *grid* se duplica.

- ✗ *Grid array* de gran tamaño
- ✓ Consultas de rango eficientes.

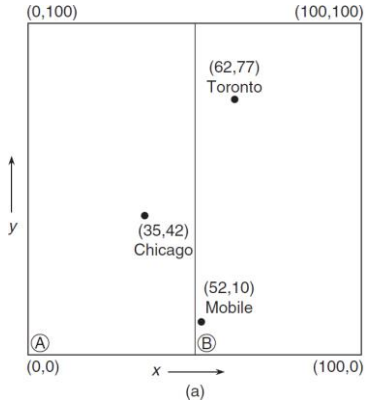
EXCELL



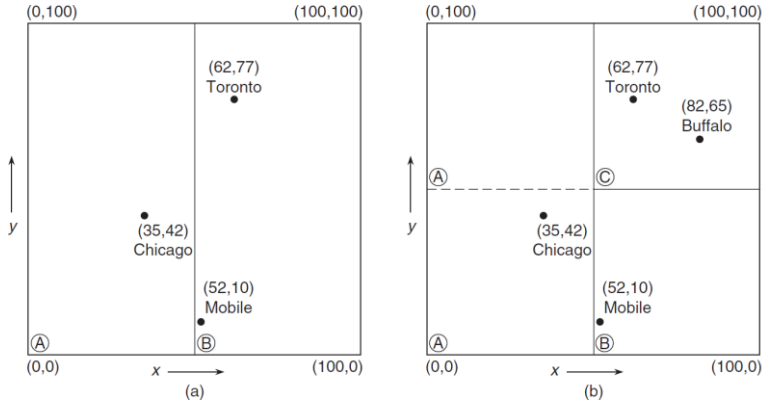


✗ Asumiendo capacidad d=2, la sobrecarga del EXCELL requiere ejecutar **tres veces** el algoritmo de partición de espacio.

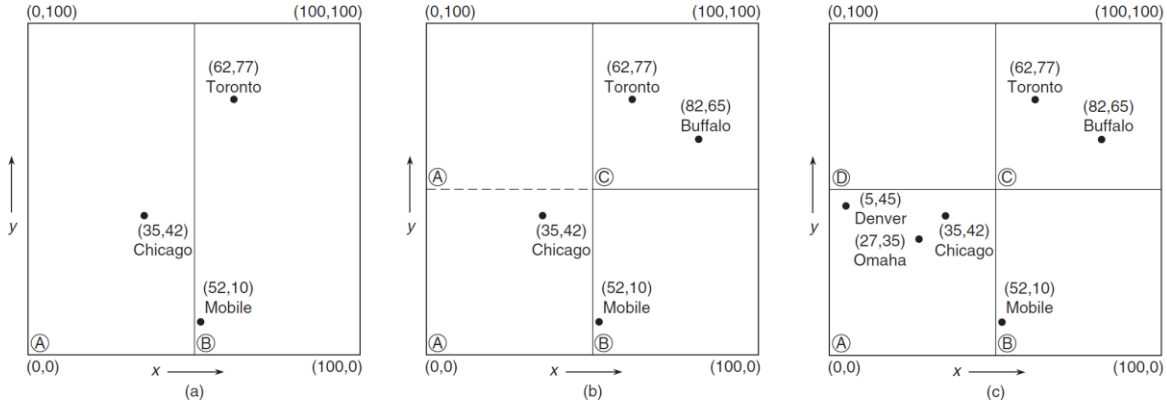
Inserción


(0,100)
(100,100)
(62,77)
Toronto
y
(35,42)
Chicago
(52,10)
Mobile
A
B
(0,0)
x
(100,0)
(a)

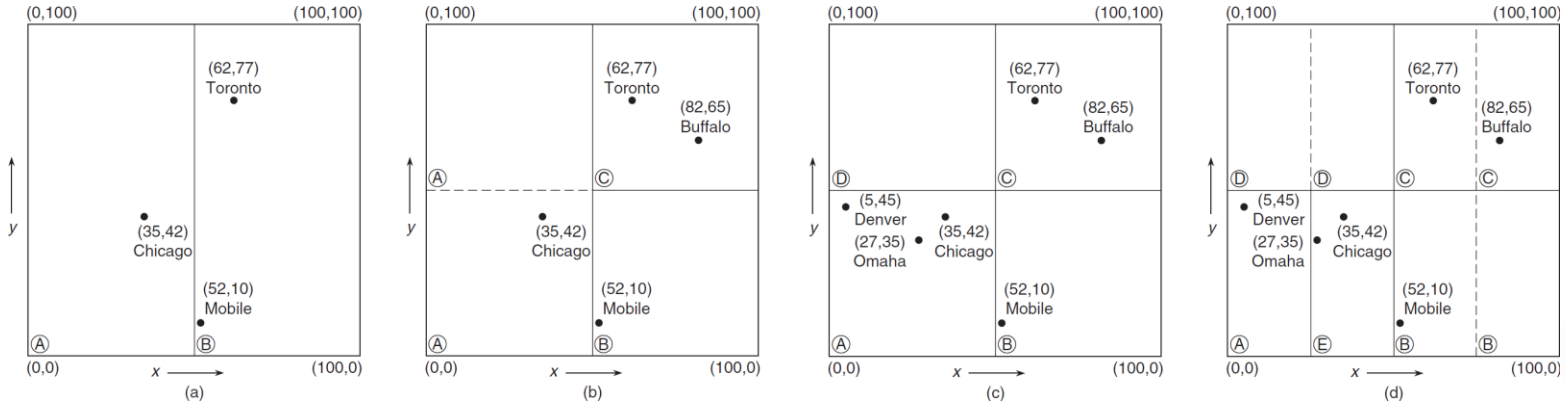
Inserción


(0,100)
(100,100)
(62,77)
Toronto
(35,42)
Chicago
(52,10)
Mobile
A
B
(0,0)
x
(100,0)
y
(a)
(0,100)
(100,100)
(62,77)
Toronto
(82,65)
Buffalo
A
C
(35,42)
Chicago
(52,10)
Mobile
A
B
(0,0)
x
(100,0)
y
(b)

Inserción


(0,100)
(100,100)
(62,77)
Toronto
(35,42)
Chicago
(52,10)
Mobile
A
B
(0,0)
x
(100,0)
y
(a)
(0,100)
(100,100)
(62,77)
Toronto
(82,65)
Buffalo
A
C
(35,42)
Chicago
(52,10)
Mobile
A
B
(0,0)
x
(100,0)
y
(b)
(0,100)
(100,100)
(62,77)
Toronto
(82,65)
Buffalo
D
C
(5,45)
Denver
(27,35)
Omaha
(35,42)
Chicago
(52,10)
Mobile
A
B
(0,0)
x
(100,0)
y
(c)

Inserción


(0,100)
(100,100)
(62,77)
Toronto
(35,42)
Chicago
(52,10)
Mobile
(82,65)
Buffalo
(5,45)
Denver
(27,35)
Omaha
A
B
C
D
E
y
x
(0,0)
(100,0)
(a)
(b)
(c)
(d)

Fusión

Buddy

Los *buckets* candidatos a ser fusionados deben ser *buddies*, es decir, en algún momento anterior formaron parte de la misma celda.

Recuerde que EXCELL solo divide en potencias de dos.

Fusión

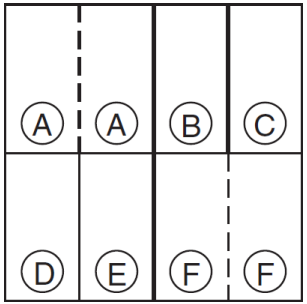
Buddy

Los *buckets* candidatos a ser fusionados deben ser *buddies*, es decir, en algún momento anterior formaron parte de la misma celda.

Recuerde que EXCELL solo divide en potencias de dos.

Directory

Surge cuando cada par de *bucket buddy* en el *grid* cumple el umbral de fusión (similar al caso anterior) o cada par de *grid cell* forman *buckets*.



La imagen representa ambas situaciones:
- Par de *bucket buddy*: B-C, D-E
- Par de *grid cell* forman buckets: A, F

Programación Paralela

Arquitectura Von Neumann

CU: control unit

ALU: arithmetic logic unit

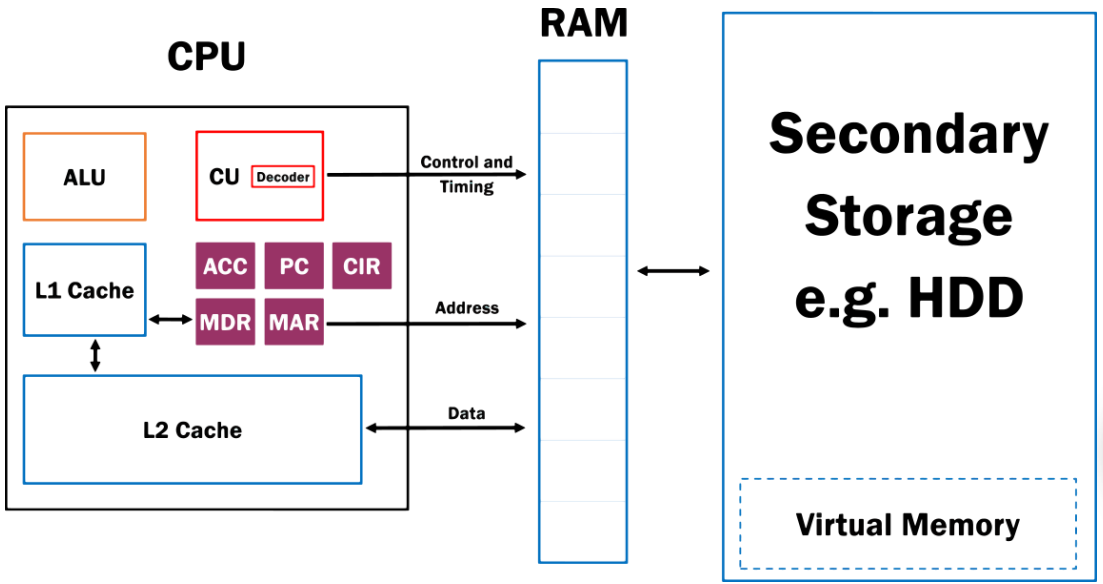
ACC: accumulator

PC: program counter

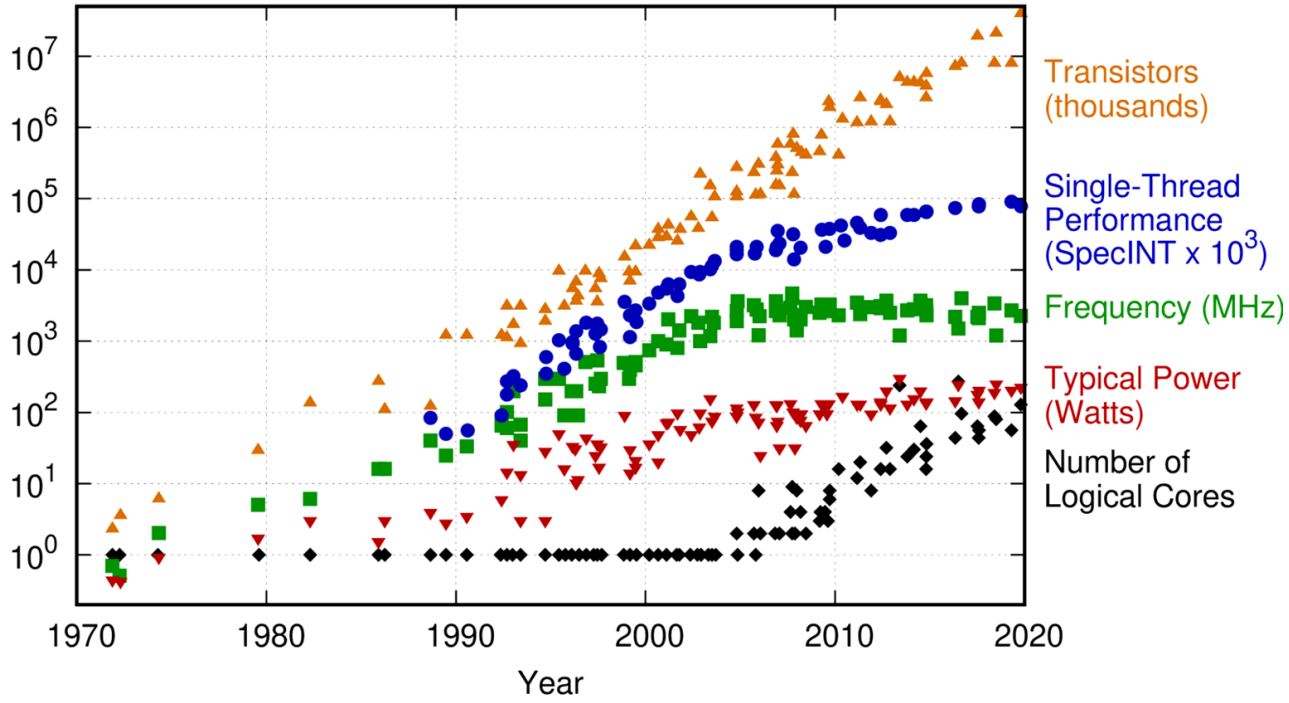
MDR: memory data register

MAR: memory address register

CIR: current instruction register



Evolución de los CPUs



Los tres muros

The Power Wall

La frecuencia no puede ser incrementada sin aumentar sustantivamente la disipación

Los tres muros

The Power Wall

La frecuencia no puede ser incrementada sin aumentar sustantivamente la disipación

The Memory Wall

El acceso a los datos es un cuello de botella

Los tres muros

The Power Wall

La frecuencia no puede ser incrementada sin aumentar sustantivamente la disipación

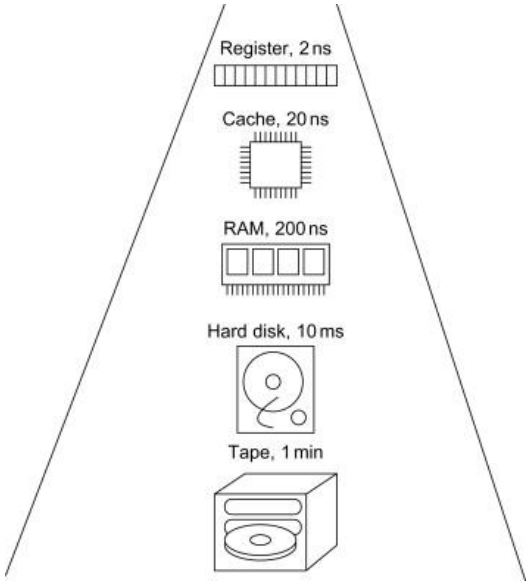
The Memory Wall

El acceso a los datos es un cuello de botella

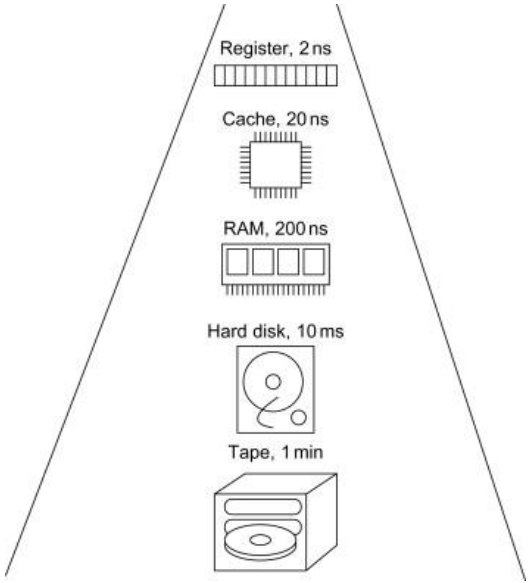
The ILP Wall

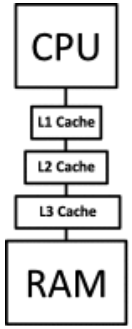
Muchas de las optimizaciones en el paralelismo a nivel de instrucción ya llegaron a cierto límite

Memorias



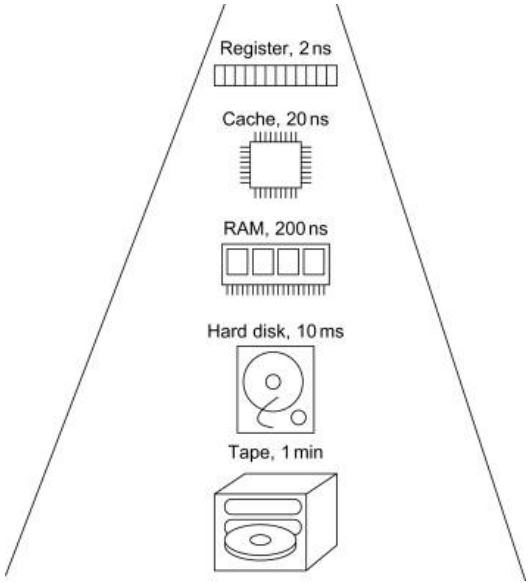
Memorias

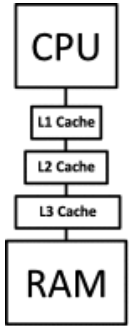




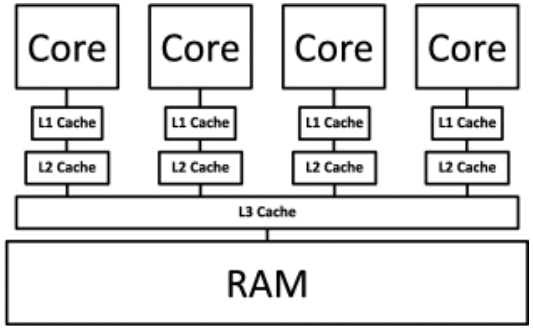
Arquitectura
De Cache

Memorias



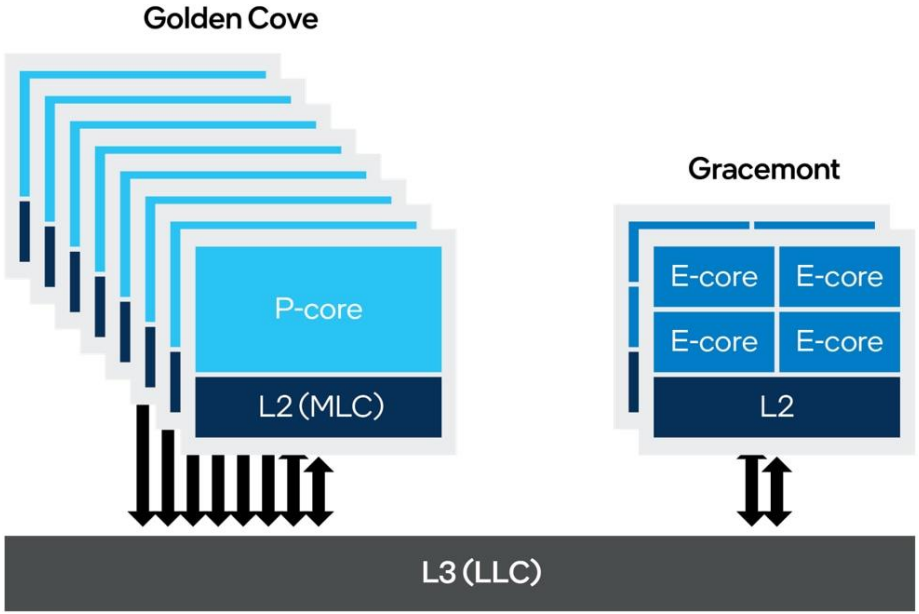


Arquitectura
De Cache

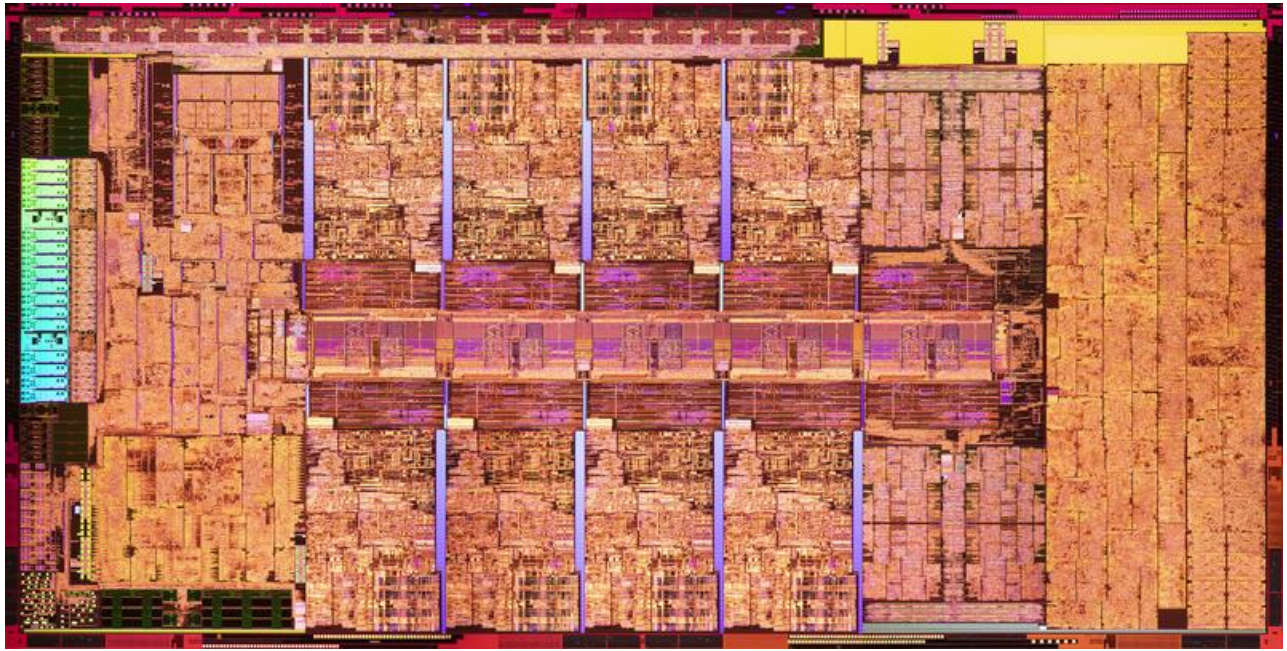


Arquitectura multinúcleo con
cache L3 compartida

Alder Lake (Intel)



Alder Lake (Intel)



Paralelización

¿Los programas pueden ser paralelizados directamente?

Paralelización

Código serial

```
sum = 0;

for (i = 0; i < n; i++) {
        x = Compute next value(. . .);
        sum += x;
 }
```

Paralelización

Código serial

```
sum = 0;

for (i = 0; i < n; i++) {
        x = Compute next value(. . .);
        sum += x;
 }
```

$x = 1,4,3,9,2,8,5,1,1,6,2,7,2,5,0,4,1,8,6,5,1,2,3,9$

$\text{sum} = 95$

Paralelización

Código serial

```
sum = 0;

for (i = 0; i < n; i++) {
        x = Compute next value(. . .);
        sum += x;
 }
```

$x = 1,4,3,\mathbf{9,2,8},5,1,1,\mathbf{6,2,7},2,5,0,\mathbf{4,1,8},6,5,1,\mathbf{2,3,9}$

$\text{sum} = 95$

Paralelización

Código serial

```
sum = 0;

for (i = 0; i < n; i++) {
        x = Compute next value(. . .);
        sum += x;
 }
```

Código paralelizado

```
my sum = 0;
my first i = . . . ;
my last i = . . . ;
for (my i = my first i; my i < my last i; my i++) {
        my x = Compute next value(. . .);
        my sum += my x;
 }
```

$x = 1,4,3,\mathbf{9,2,8},5,1,1,\mathbf{6,2,7},2,5,0,\mathbf{4,1,8},6,5,1,\mathbf{2,3,9}$

$\text{sum} = 95$

Paralelización

Código serial

```
sum = 0;

for (i = 0; i < n; i++) {
        x = Compute next value(. . .);
        sum += x;
 }
```

Código paralelizado

```
my sum = 0;
my first i = . . . ;
my last i = . . . ;
for (my i = my first i; my i < my last i; my i++) {
        my x = Compute next value(. . .);
        my sum += my x;
 }
```

$x = 1,4,3,\mathbf{9,2,8},5,1,1,\mathbf{6,2,7},2,5,0,\mathbf{4,1,8},6,5,1,\mathbf{2,3,9}$

$\text{sum} = 95$

Core	0	1	2	3	4	5	6	7
my sum	8	19	7	15	7	13	12	14

Paralelización

Suma global

```
if (I'm the master core) {
        sum = my x;
        for each core other than myself {
                receive value from core;
                sum += value;
        }
} else {
        send my x to the master;
}
```

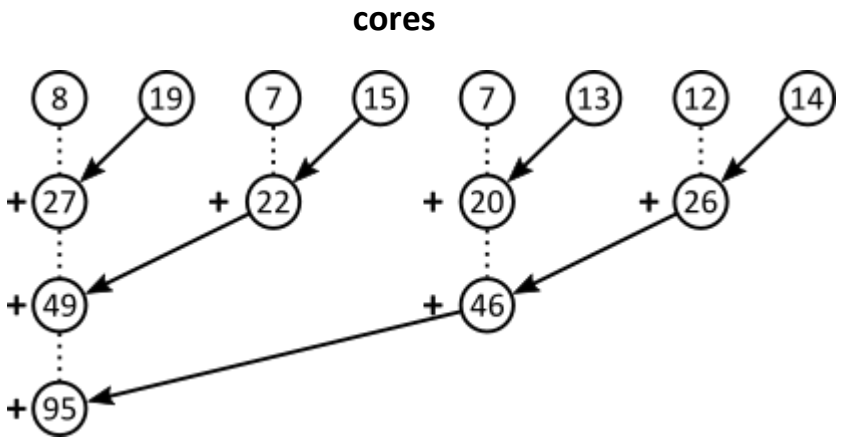
$x = 1,4,3,\mathbf{9,2,8},5,1,1,\mathbf{6,2,7},2,5,0,\mathbf{4,1,8},6,5,1,\mathbf{2,3,9}$

$\text{sum} = 95$

Core	0	1	2	3	4	5	6	7
my sum	8	19	7	15	7	13	12	14

Paralelización

Suma global



Paralelización

Task parallelism

Distribuir diferentes tareas a varios núcleos

Data parallelism

Distribuir los datos a varios núcleos

Paralelización

Task parallelism

Distribuir diferentes tareas a varios núcleos

Data parallelism

Distribuir los datos a varios núcleos

Ejemplo: Profesor principal tiene 100 alumnos, 5 profesores asistentes, un examen a 5 preguntas.

Paralelización

Task parallelism

Distribuir diferentes tareas a varios núcleos

Data parallelism

Distribuir los datos a varios núcleos

Ejemplo: Profesor principal tiene 100 alumnos, 5 profesores asistentes, un examen a 5 preguntas.

Cada profesor asistente corrige 1 pregunta para los 100 alumnos

Cada núcleo realiza la(s) misma(s) tarea(s), con todos los datos

Cada profesor asistente corrige 20 alumnos

Cada uno de los núcleos realiza todas la tareas, con diferentes datos.

Paralelización

Cuando los núcleos pueden hacer un trabajo independiente entonces no hay dificultad

Paralelización

Cuando los núcleos pueden hacer un trabajo independiente entonces no hay dificultad

Pero eso es poco común :(

Paralelización

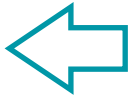
Comunicación entre núcleos

Paralelización

Comunicación entre núcleos

Sincronización

Core 1: Task 1
Core 2: Task 2
Synchronize



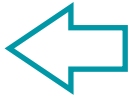
El programa se vuelve más complejo

Paralelización

Comunicación entre núcleos

Sincronización

Core 1: Task 1
Core 2: Task 2
Synchronize



El programa se vuelve más complejo

Load balancing

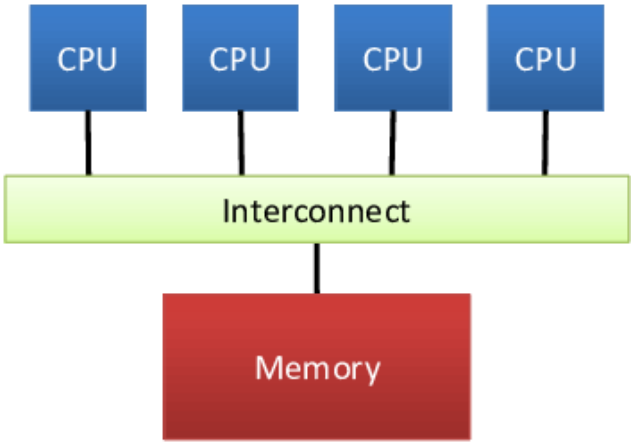
Las tareas se distribuyen equitativamente entre todos los núcleos

En caso contrario, hay **sobrecarga** de uno o más núcleos.

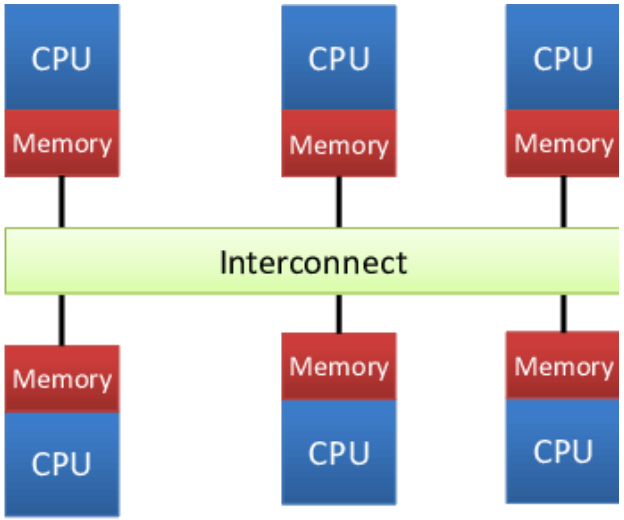
Paralelización

¿Cuál de los algoritmos vistos en clase se puede paralelizar?

Sistemas paralelos

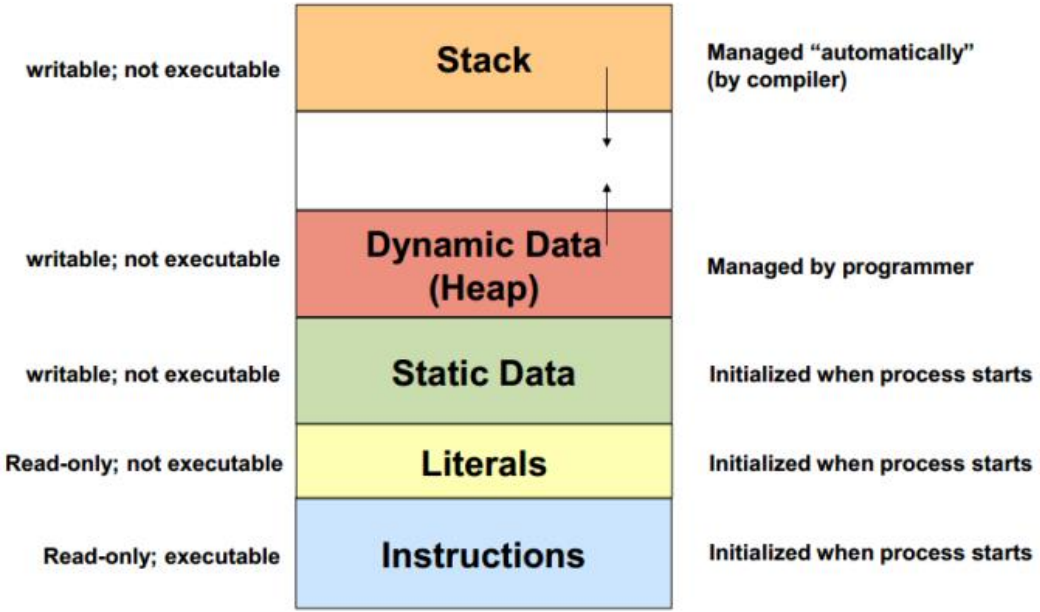


Shared Memory

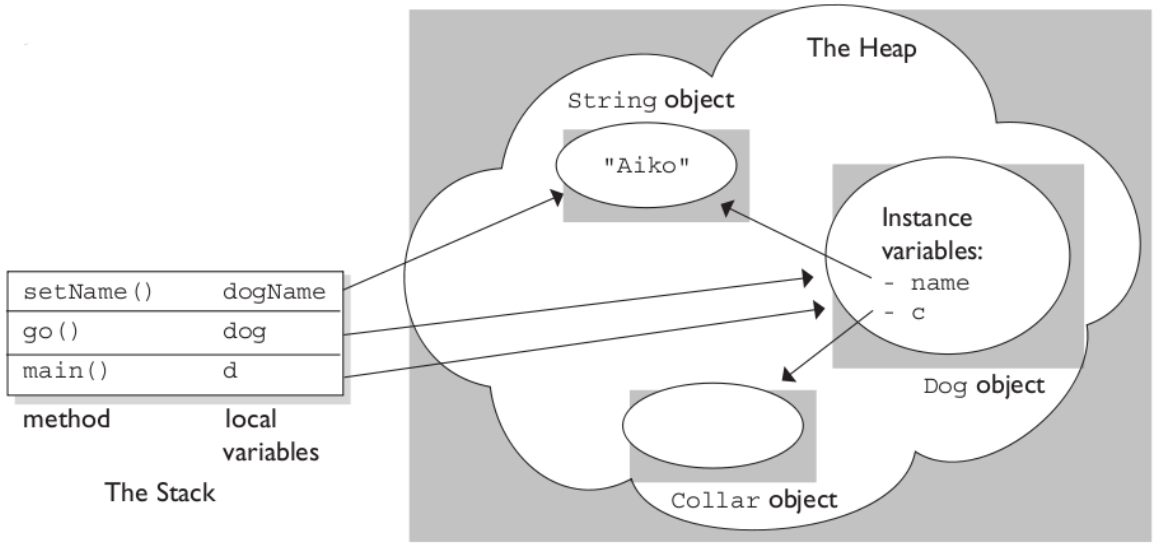


Distributed Memory

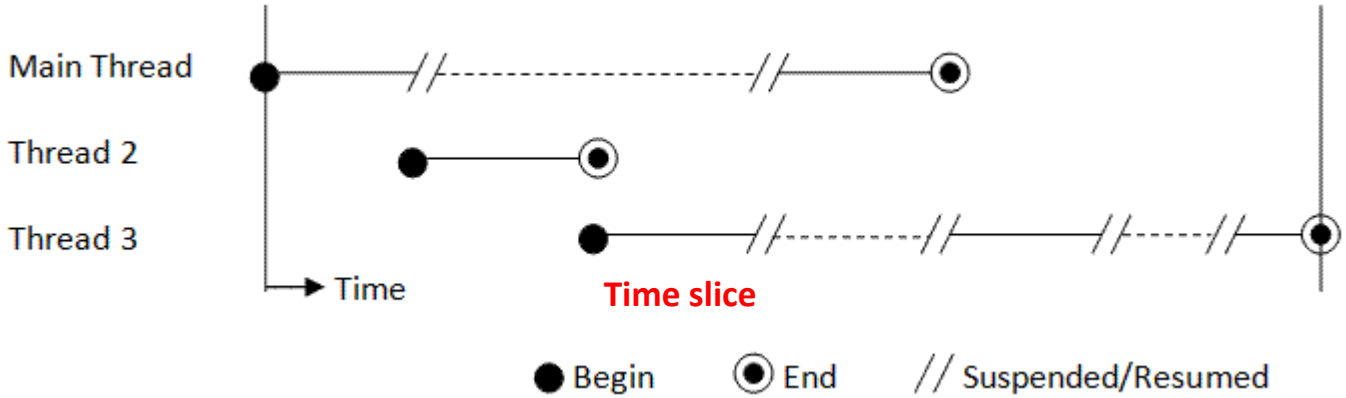
Proceso


writable; not executable
Stack
Managed "automatically" (by compiler)
writable; not executable
Dynamic Data (Heap)
Managed by programmer
writable; not executable
Static Data
Initialized when process starts
Read-only; not executable
Literals
Initialized when process starts
Read-only; executable
Instructions
Initialized when process starts

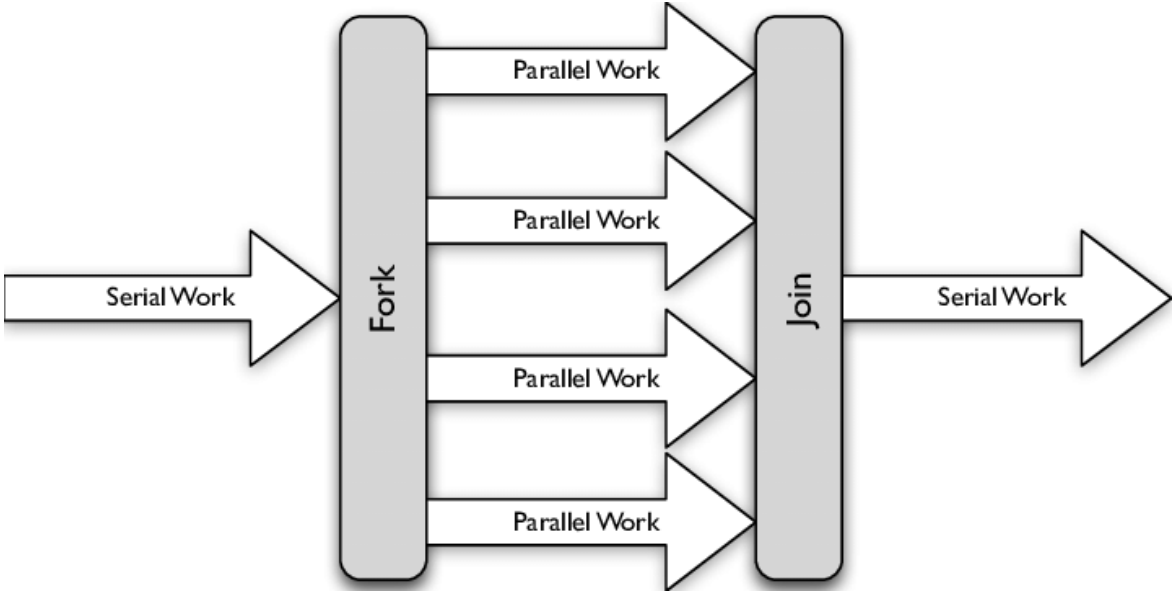
Proceso


The Heap
String object
"Aiko"
Instance variables:
- name
- c
Dog object
Collar object
setName()
dogName
go()
dog
main()
d
method
local variables
The Stack

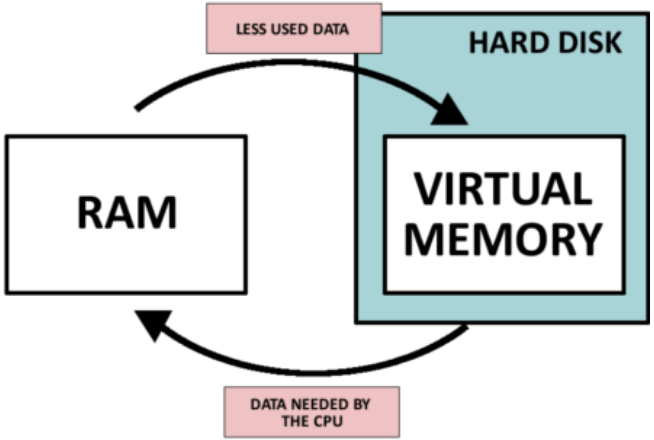
Multitasking

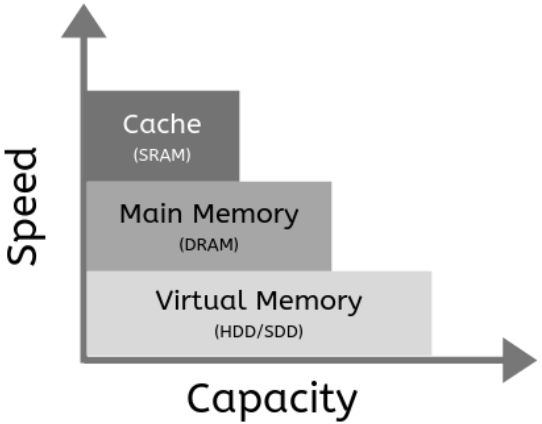


Threading


Serial Work
Fork
Parallel Work
Parallel Work
Parallel Work
Parallel Work
Join
Serial Work

Memoria Virtual





Shared memory

Variables privadas

Variables compartidas

Shared memory

Threads dinámicos

Threads estáticos

Shared memory

Problemas…

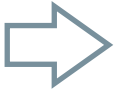
- Dos threads acceden a un mismo dato para cambiarlo
- **Race condition**
- **Critical section**
- Es la responsabilidad del programador de asegurar una exclusión mutua entre threads

Shared memory

Busy-waiting

Shared memory

Busy-waiting

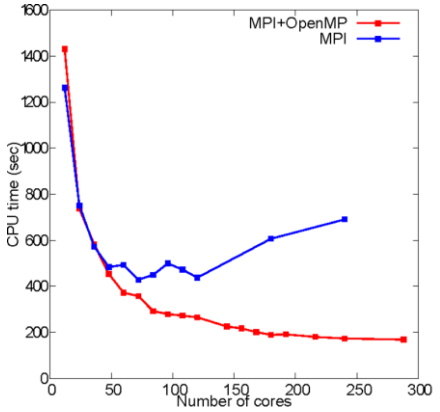


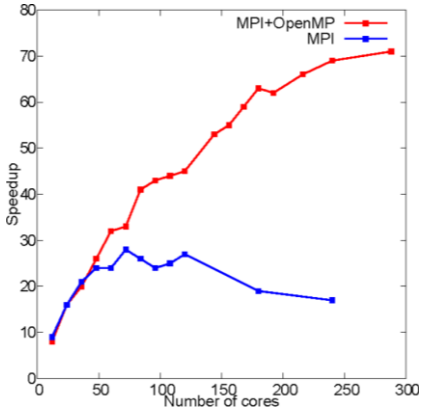
Se trata de calificar una función thread safe o no

Speed-up y eficiencia

Speed-up

$$S = \frac{T_{\text{serial}}}{T_{\text{parallel}}}$$





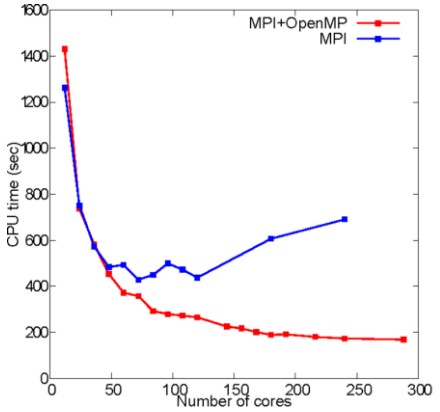
Speed-up y eficiencia

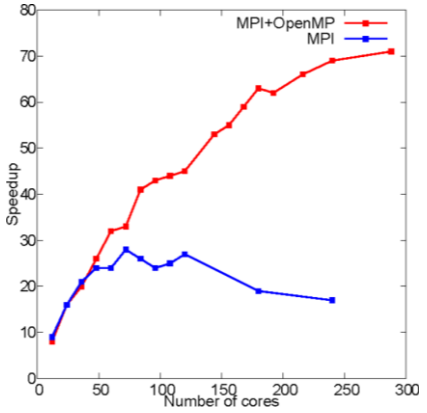
Speed-up

$$S = \frac{T_{\text{serial}}}{T_{\text{parallel}}}$$

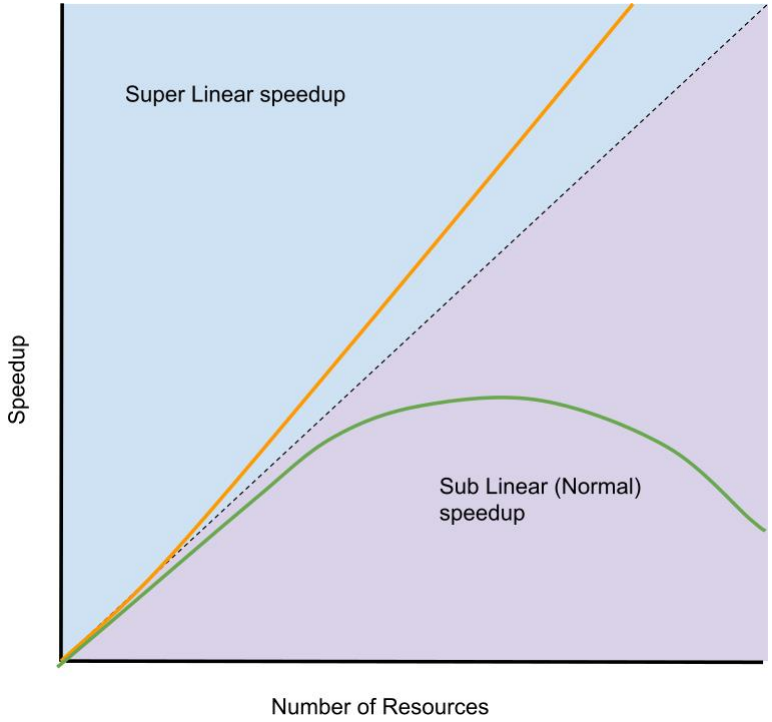
Efficiency

$$E = \frac{S}{p} = \frac{\left(\frac{T_{\text{serial}}}{T_{\text{parallel}}}\right)}{p} = \frac{T_{\text{serial}}}{p \cdot T_{\text{parallel}}}$$

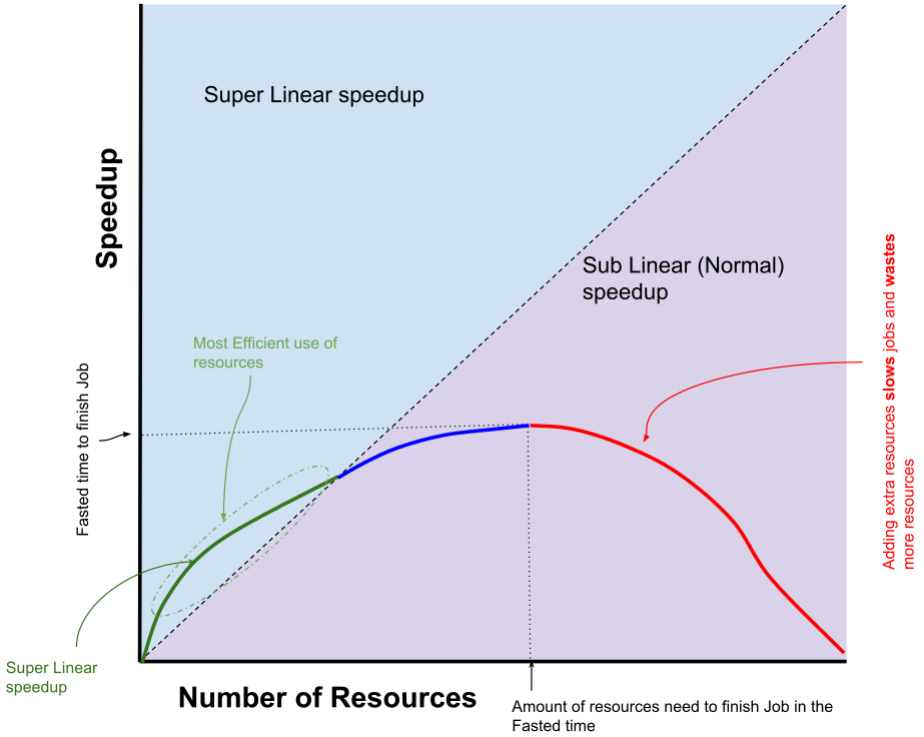




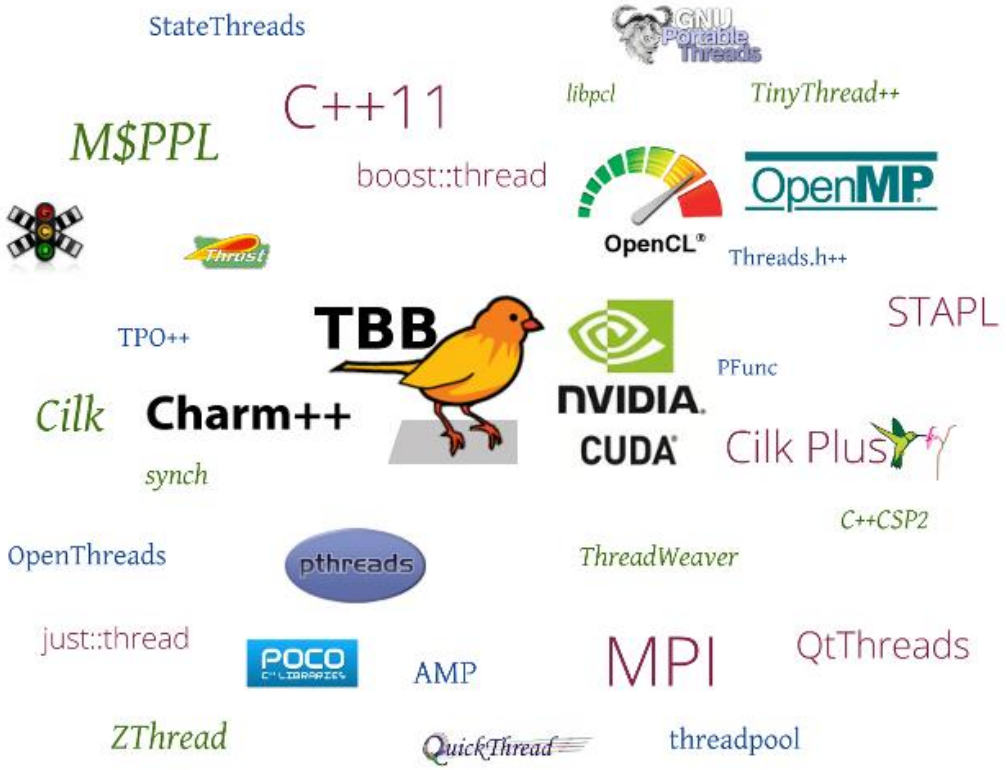
Speed-up y eficiencia



Speed-up y eficiencia



Frameworks y bibliotecas



¿Preguntas?

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA