Auxiliar Nº3 – Programación paralela

CC7515 - Computación en GPU

Profesora:

Nancy Hitschfeld

Auxiliares:

Pablo Pizarro R. Sergio P. Salinas

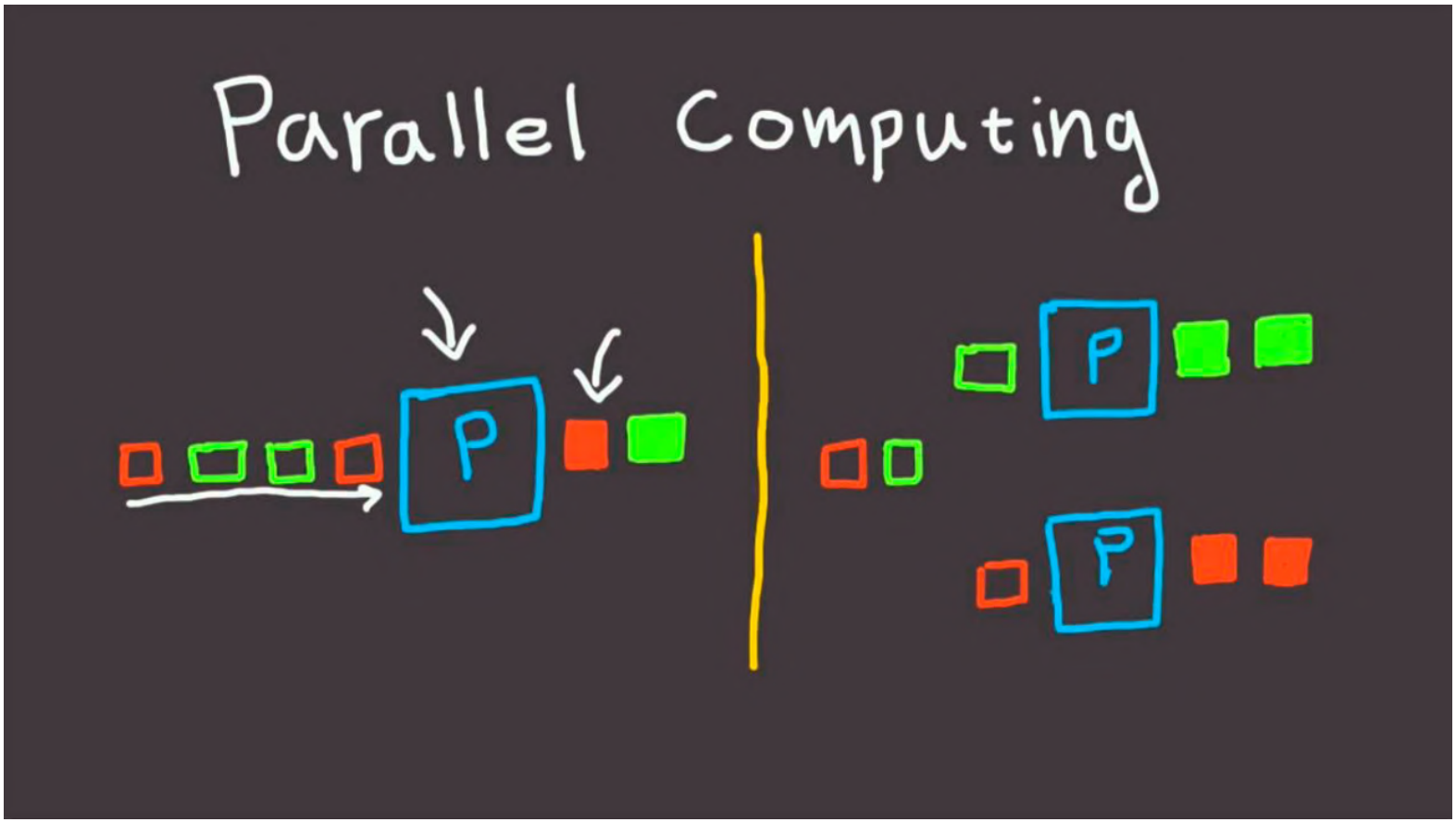
Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

3 de abril de 2022

Contenidos

¿Qué es la programación paralela?

Programa que ejecuta una serie de operaciones en paralelo (al mismo tiempo).



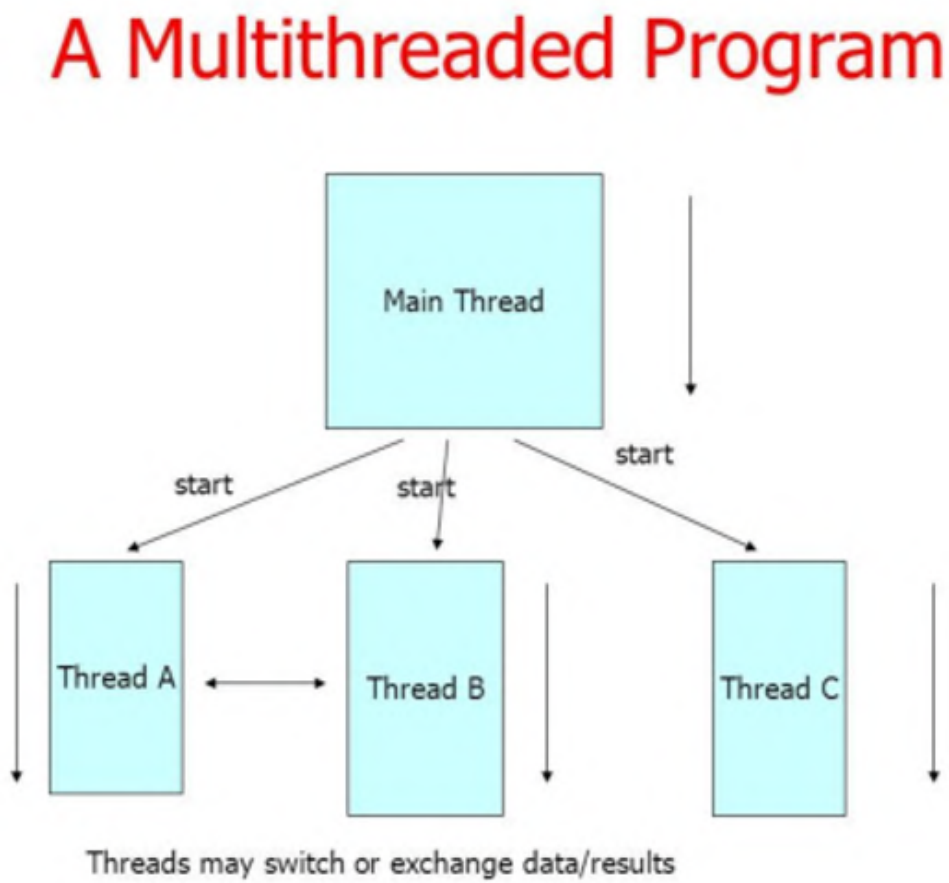
Programación paralela

- Puede convertir la ejecución a una no determinista.
- Toma ventaja de las arquitecturas actuales.
- Hay que mirar los problemas desde otro punto de vista.

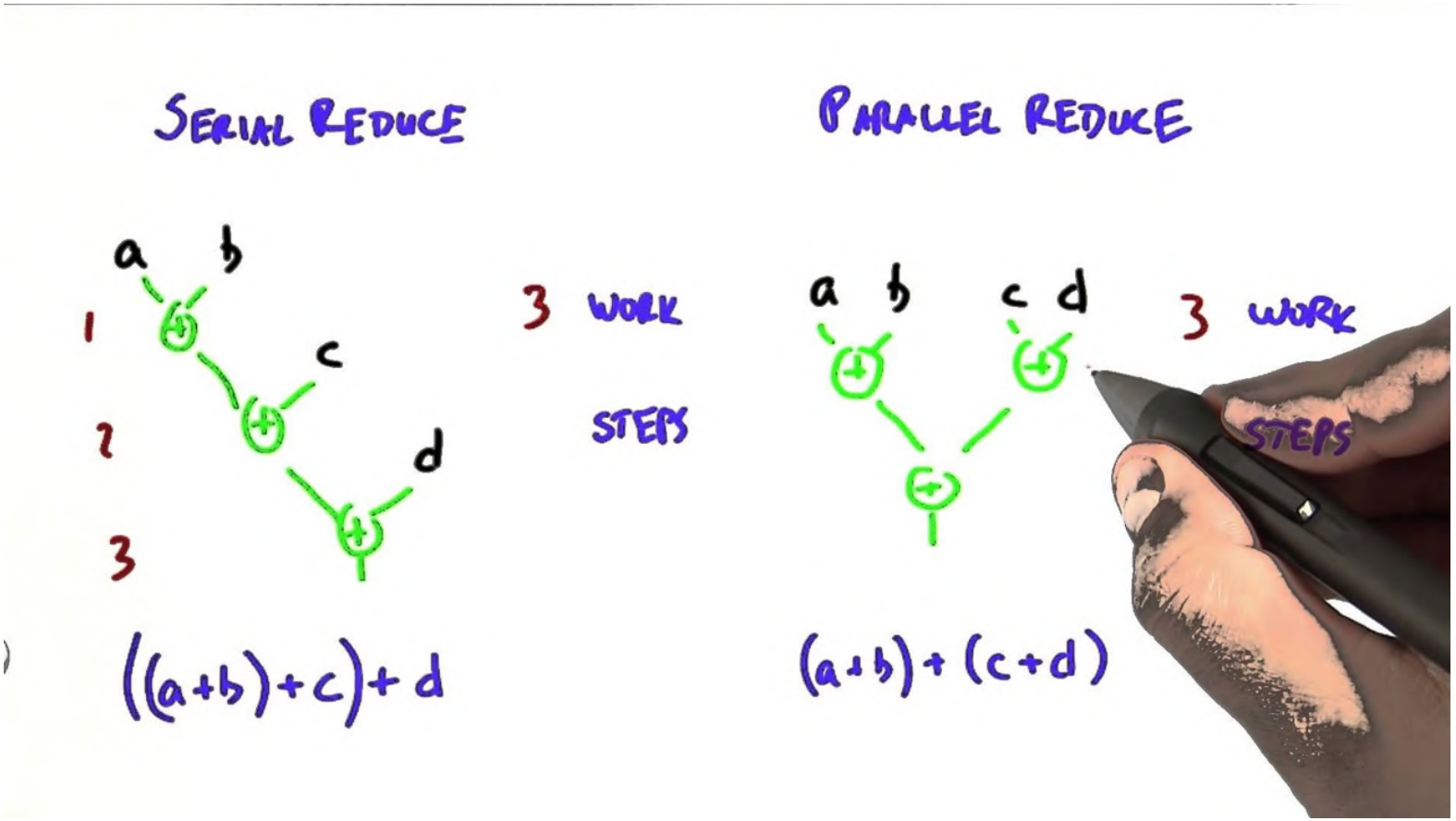
Nociones básicas

- **Task**: Secuencia de instrucciones que deben ejecutarse secuencialmente.
- **Ejecución** concurrente: Múltiples tareas independientes que **pueden** ejecutarse simultáneamente. Si dos tareas son dependientes entonces no son concurrentes.

Nociones básicas



Nociones básicas



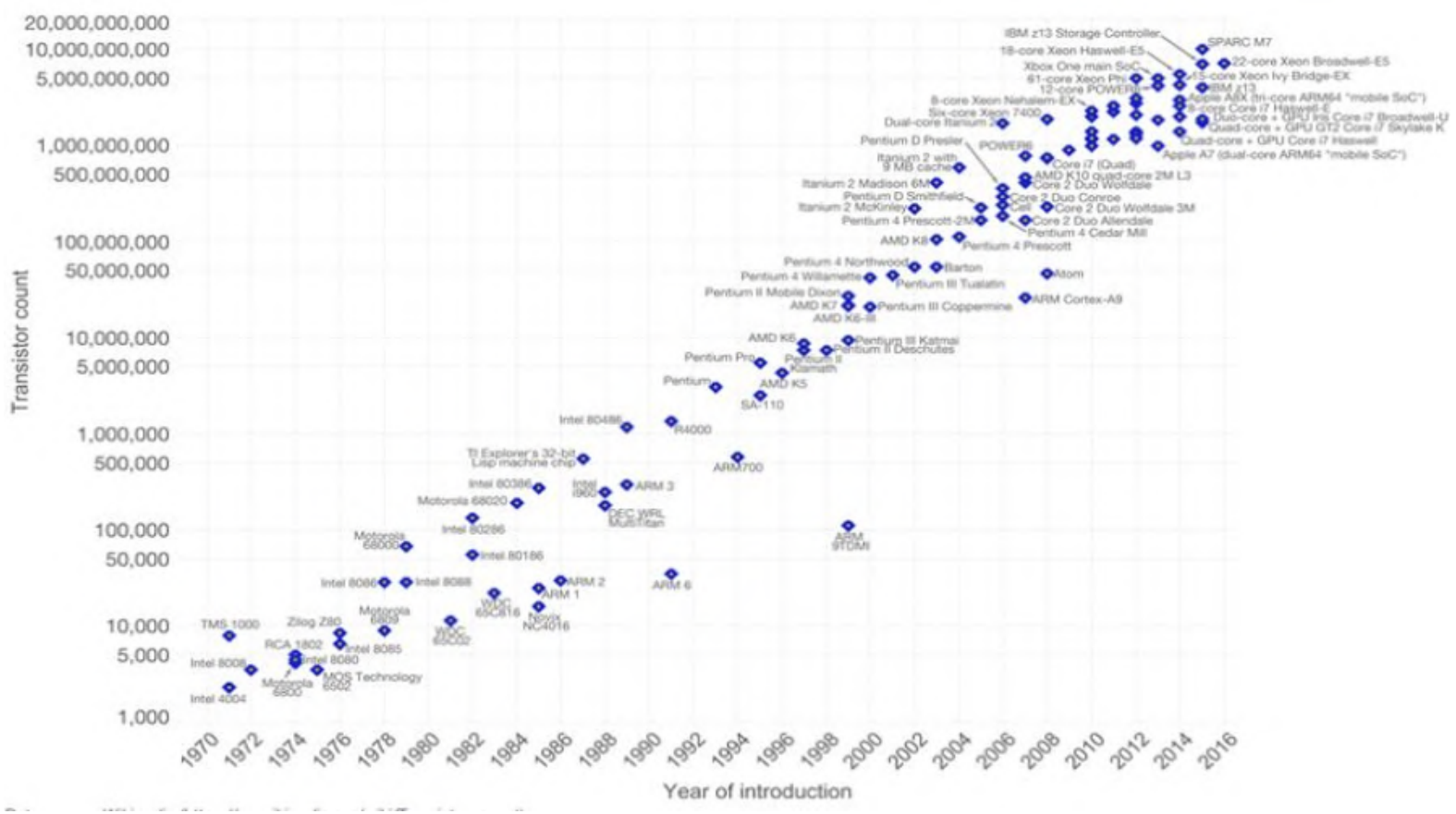
¿Qué ganamos con el paralelismo?

- Una solución más rápida.
- Resolver problemas más grandes.
- Uso efectivo de los recursos del computador.

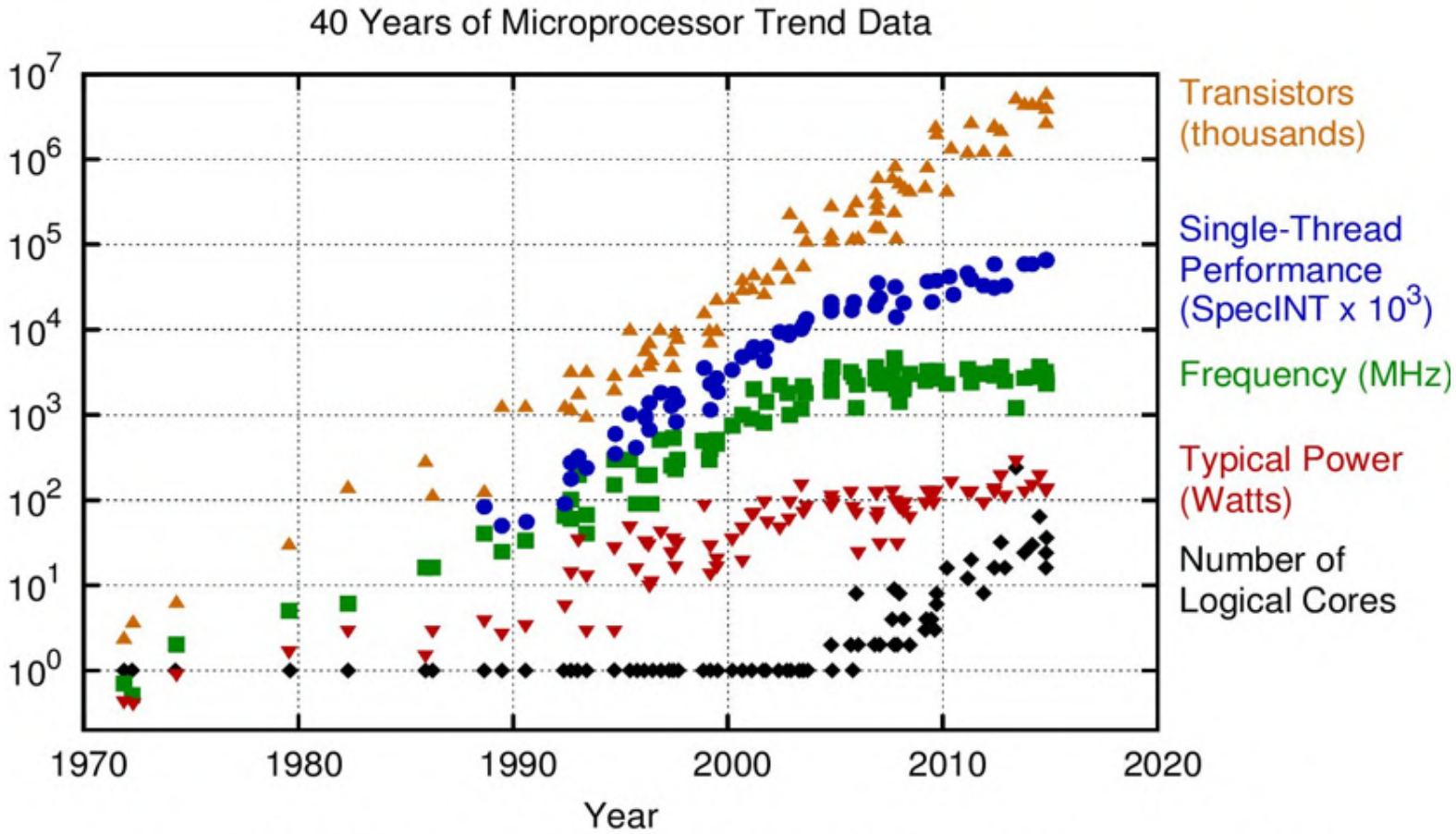
¿Para qué usar paralelismo si podemos mejorar los procesadores?

Intel Core-X Series (Kabylake-X, Skylake-X)							
Processor	Cores/ Threads	L3 Cache	PCIe Lanes	Base Clock	Turbo Clock 2.0	Turbo Clock 3.0	Launch
Core i9-7920X	12C/24T	16.5 MB	44	TBD	TBD	TBD	August
Core i9-7900X	10C/20T	13.75 MB	44	3.3 GHz	4.3 GHz	4.5 GHz	June
Core i9-7820X	8C/16T	11 MB	28	3.6 GHz	4.3 GHz	4.5 GHz	June
Core i9-7800X	6C/12T	8.25 MB	28	3.5 GHz	4.0 GHz	-	June
Core i7-7740K	4C/8T	8 MB	16	4.3 GHz	4.5 GHz	-	June
Core i7-7640K	4C/4T	6 MB	16	4.0 GHz	4.2 GHz	-	June

¿Qué nos impide aumentar la cantidad de transistores?

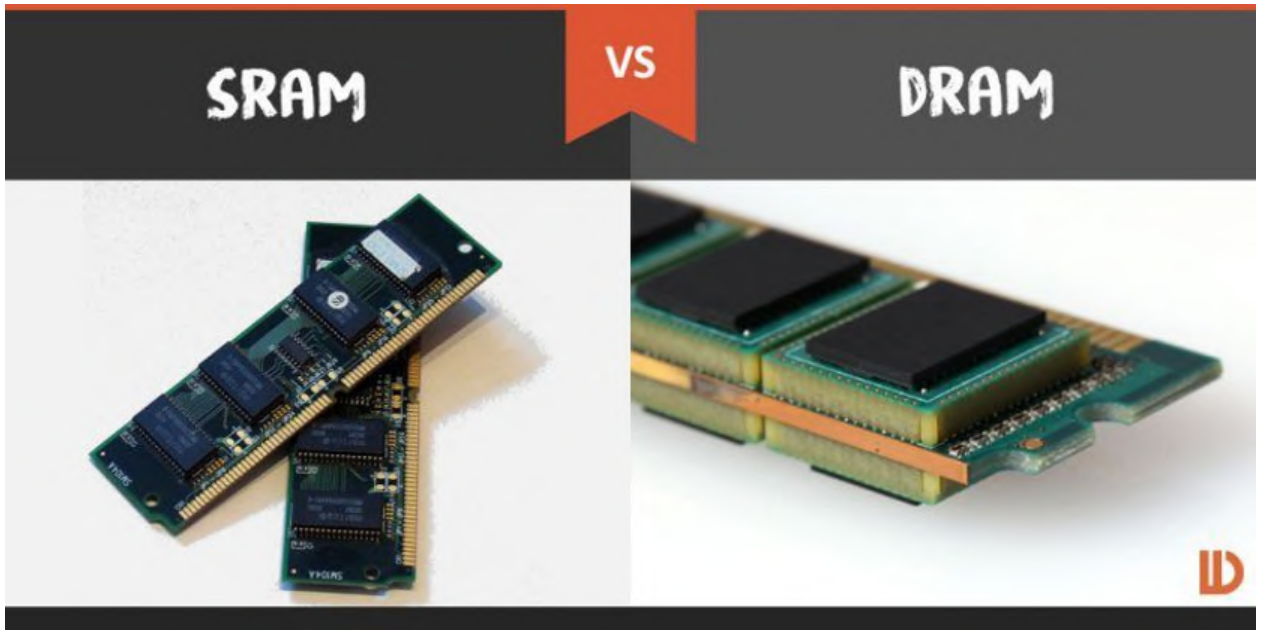


¿Qué nos impide aumentar la cantidad de transistores?



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

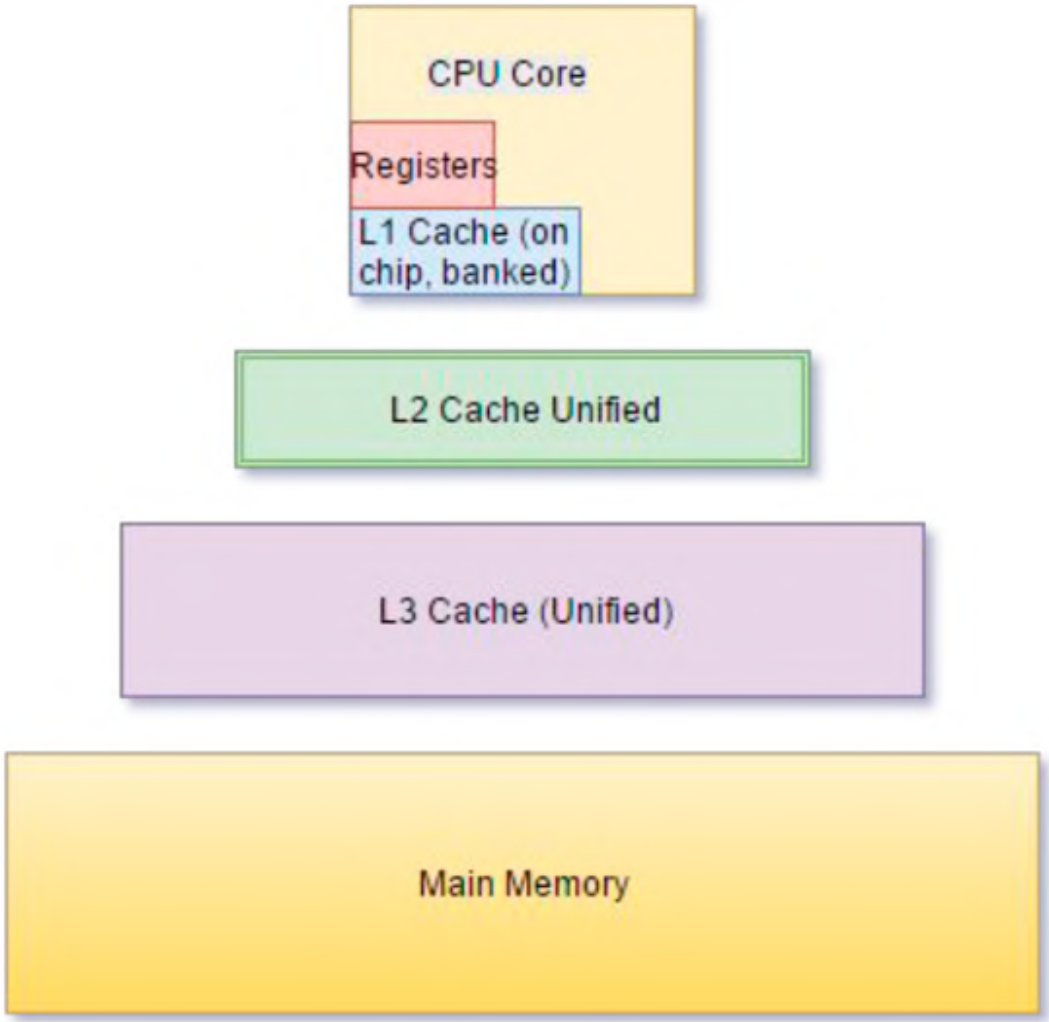
¿Cuáles son las memorias del procesador?



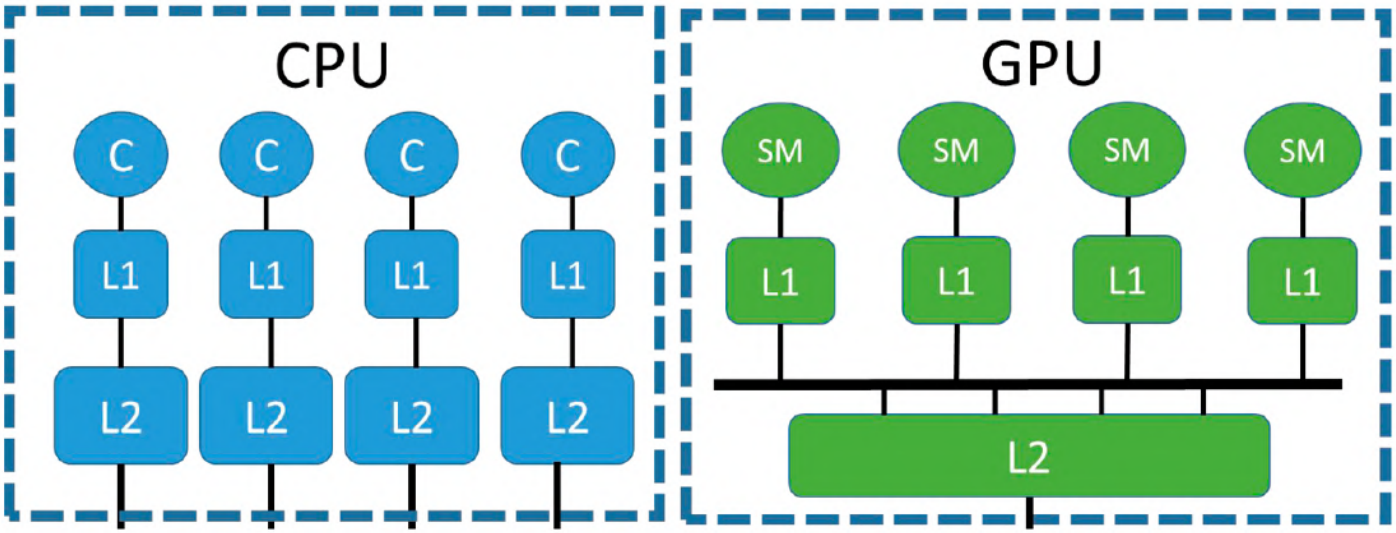
¿Qué tipos existen?

Memory technology	Typical access time	$ per GB in 2004
SRAM	0.5–5 ns	$4000–$10,000
DRAM	50–70 ns	$100–$200
Magnetic disk	5,000,000–20,000,000 ns	$0.50–$2

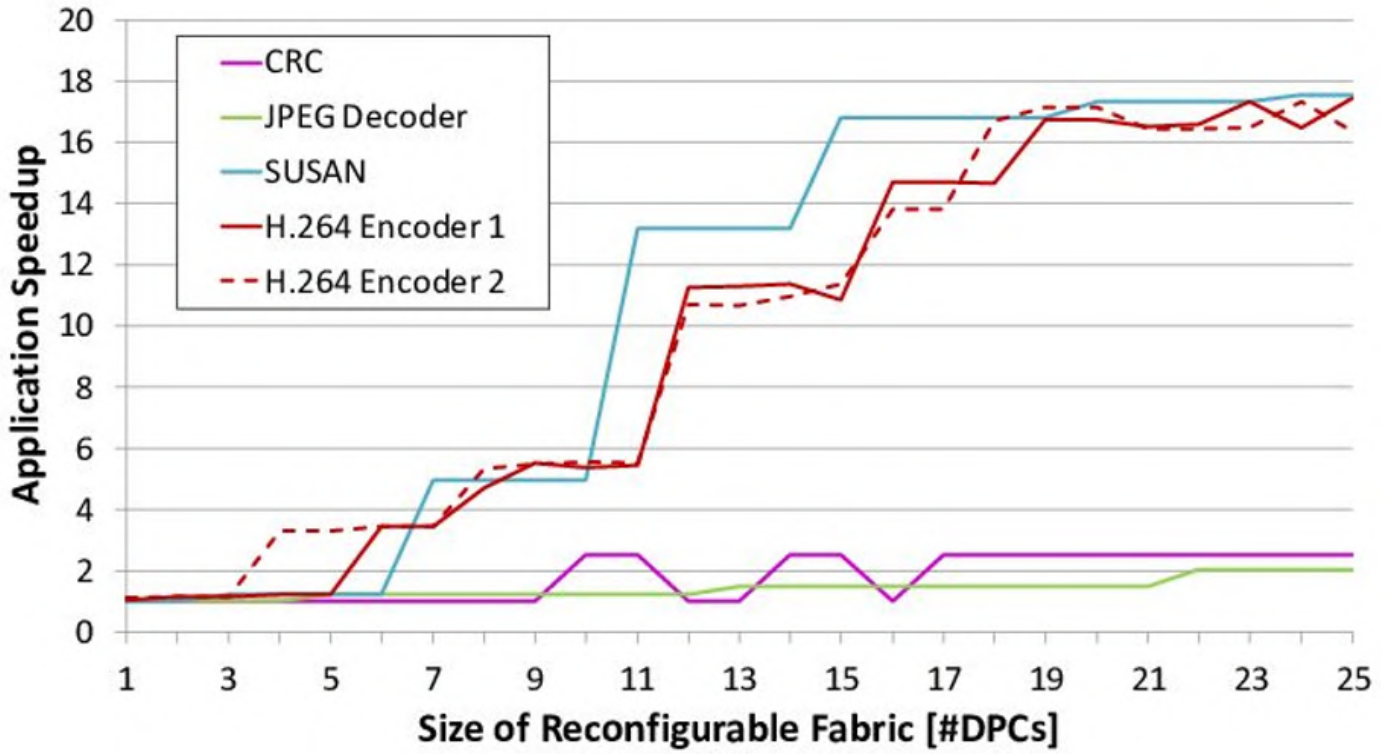
Niveles de memoria



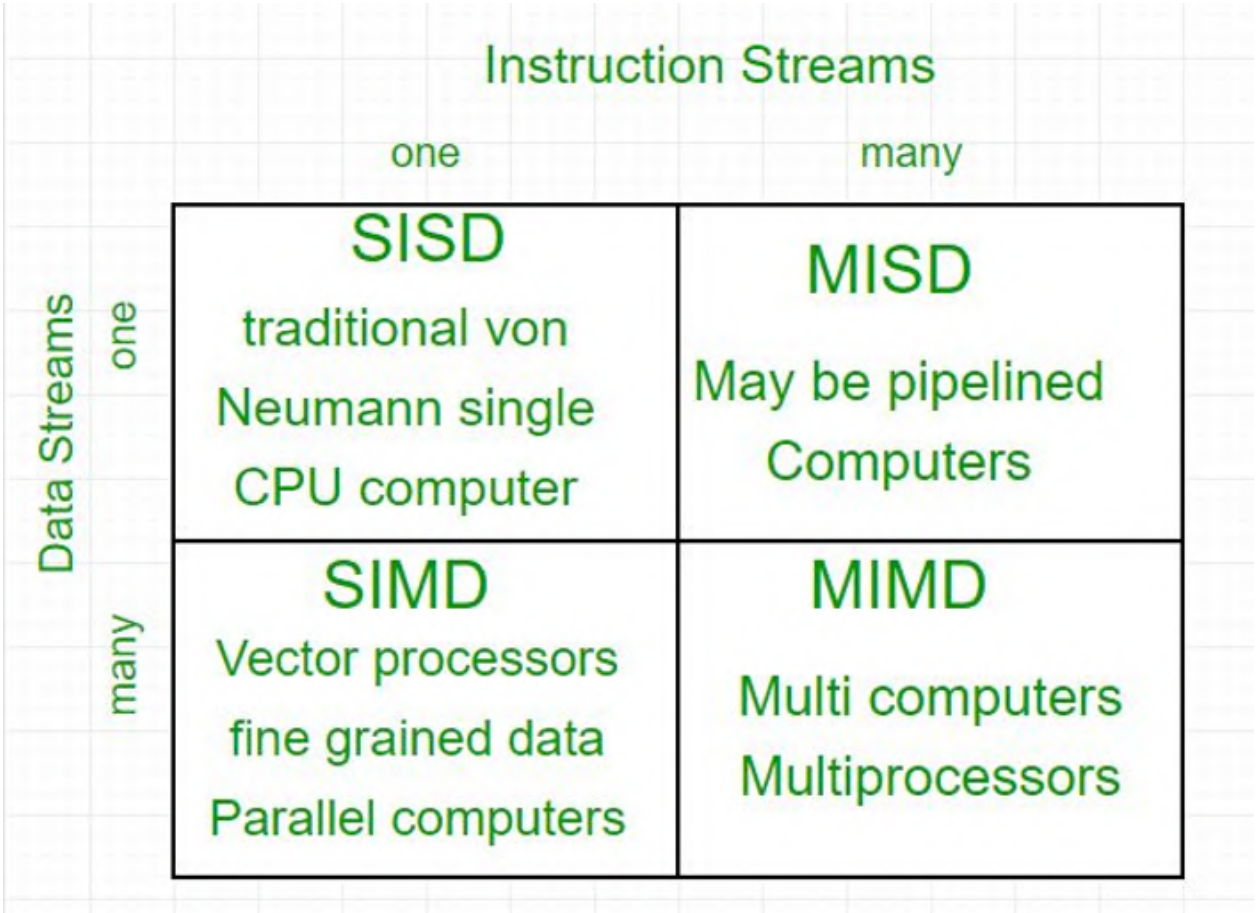
Jerarquía de memoria en la GPU



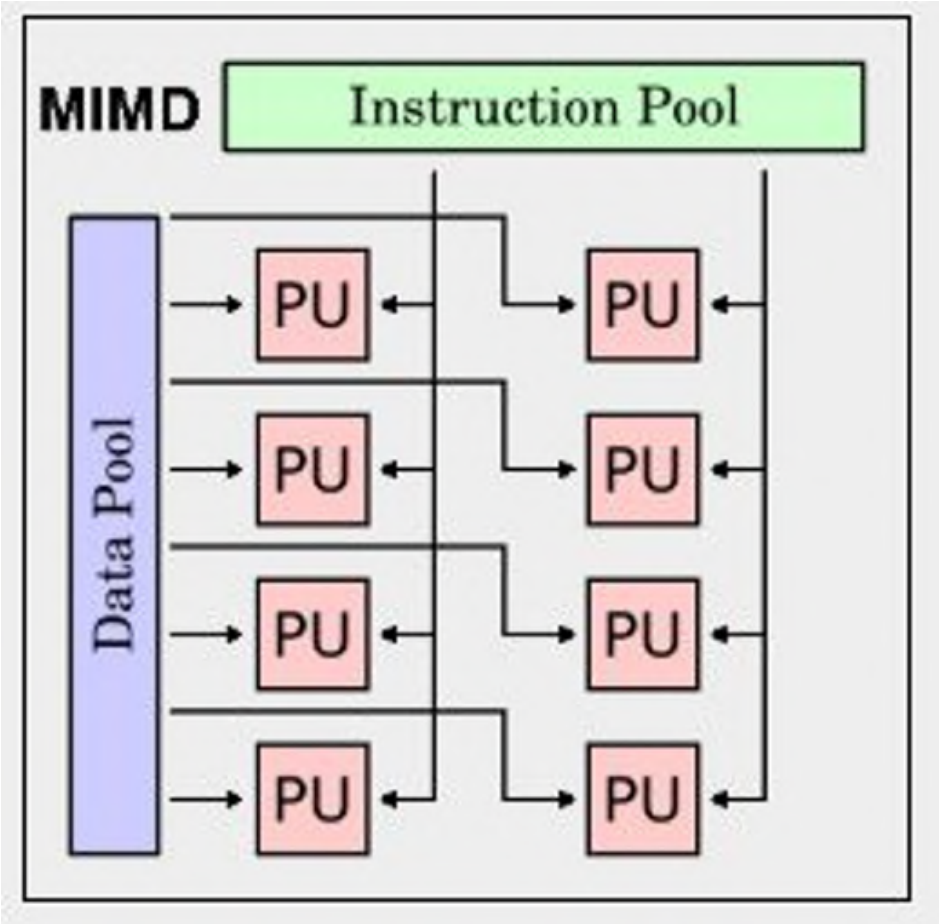
Efectos del cache en los resultados



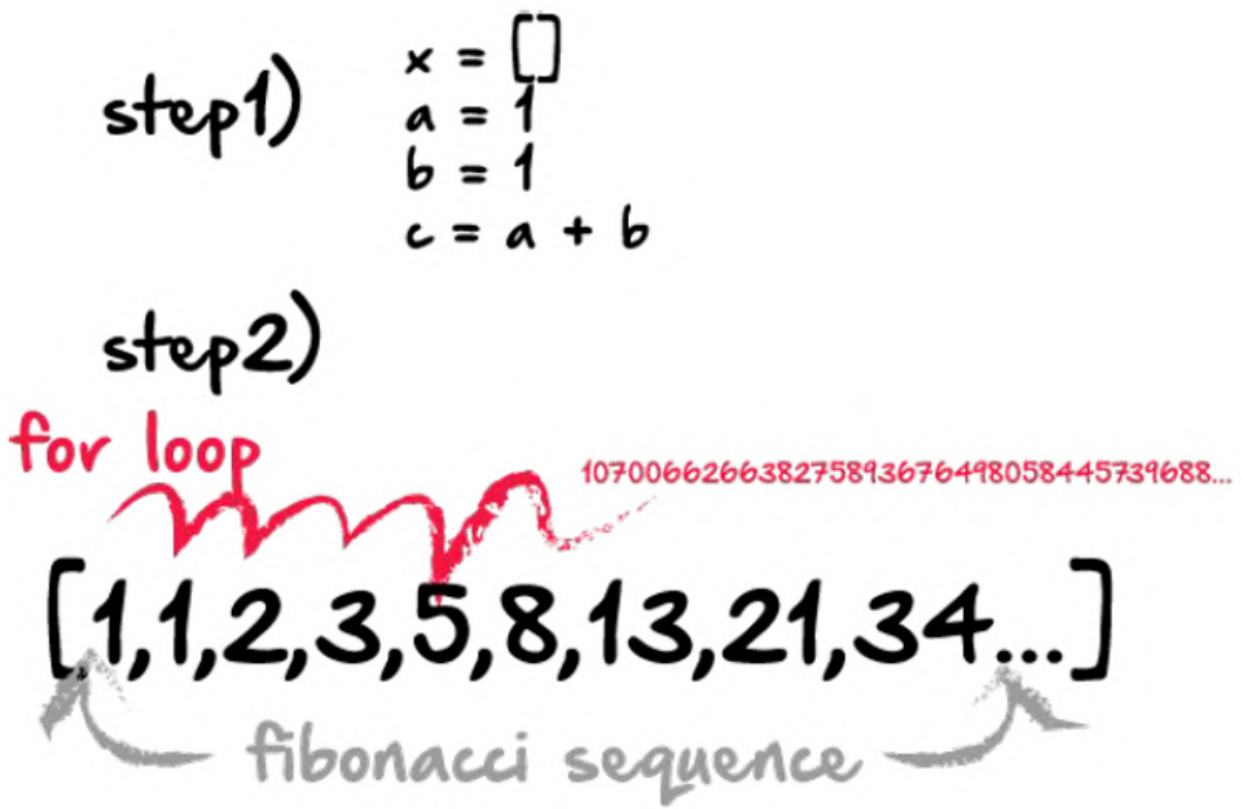
Flynn's taxonomy of processors



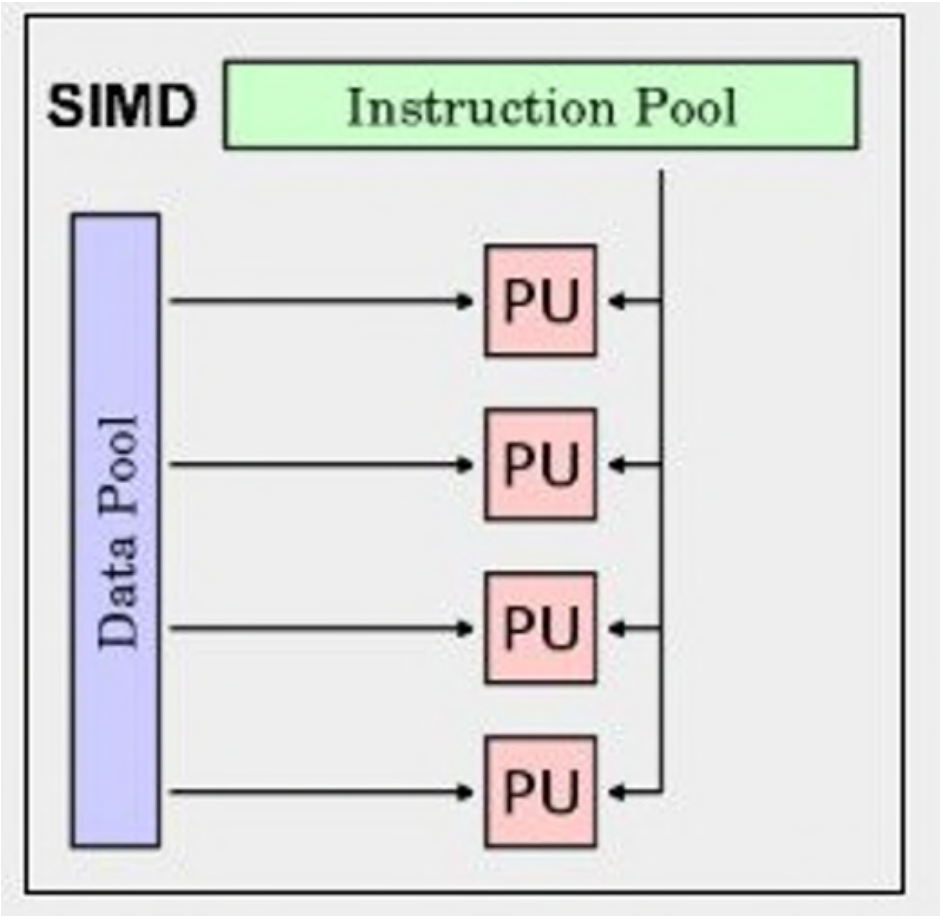
Task Parallel



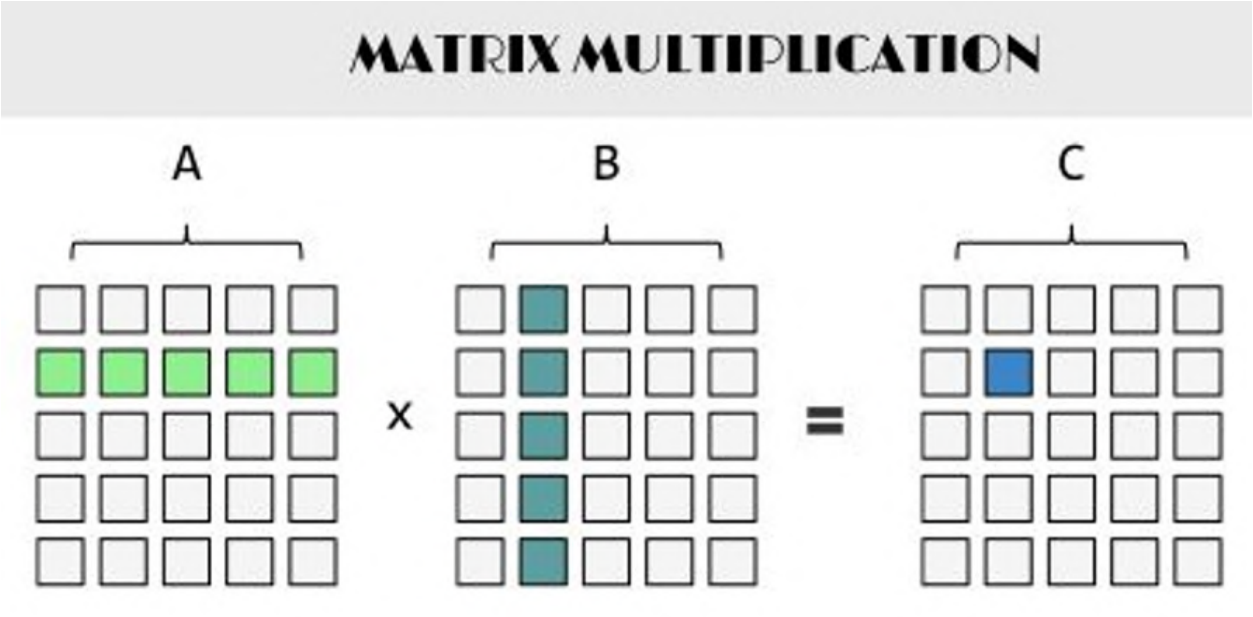
Task Parallel – Fibonacci



Data Parallel



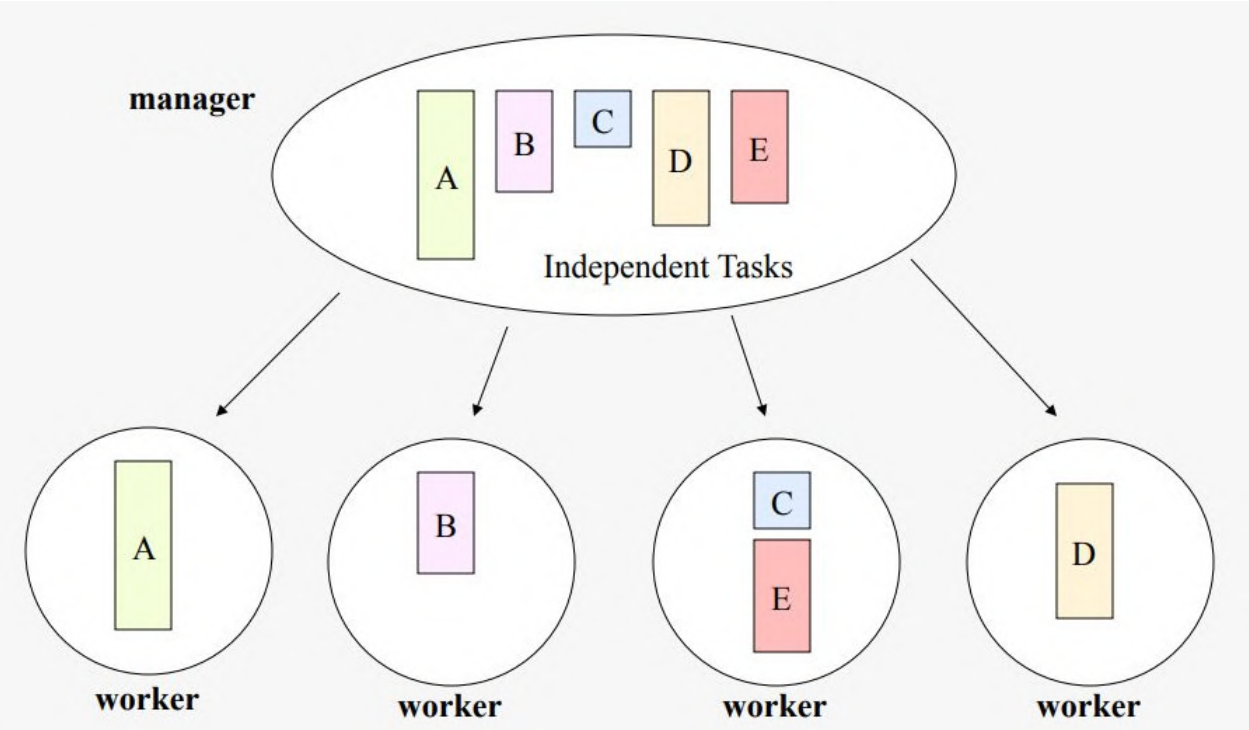
Data Parallel – Multiplicación de matrices



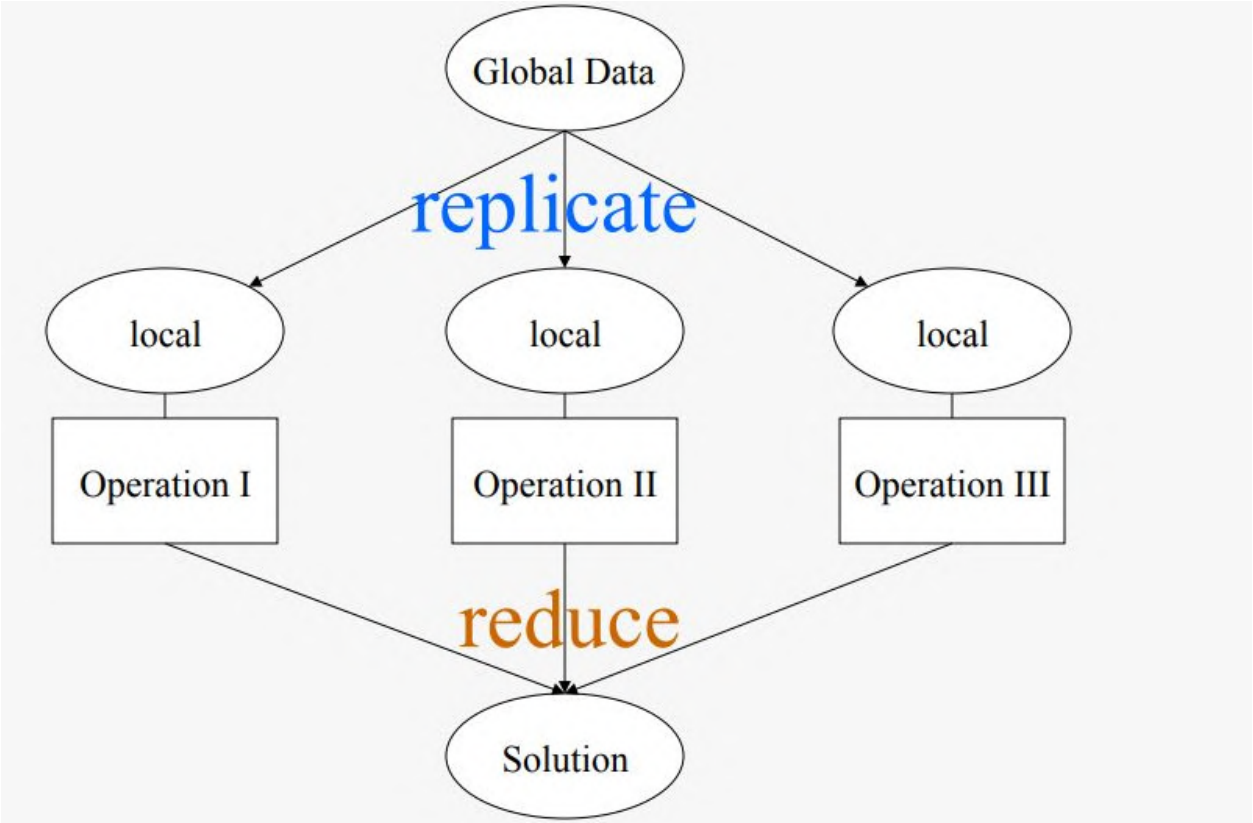
Patrones programación paralela

- Embarrassingly Parallel
- Replicable (reduce)
- Divide & Conquer
- Recursive Data

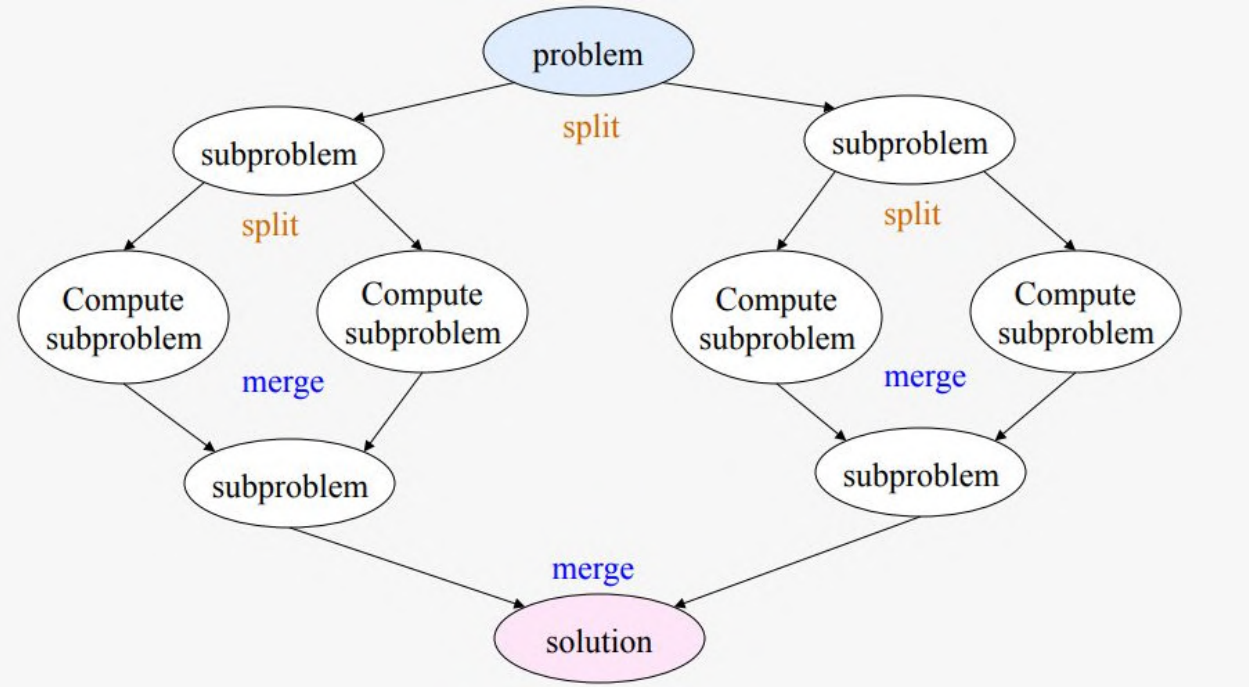
Embarrassingly Parallel



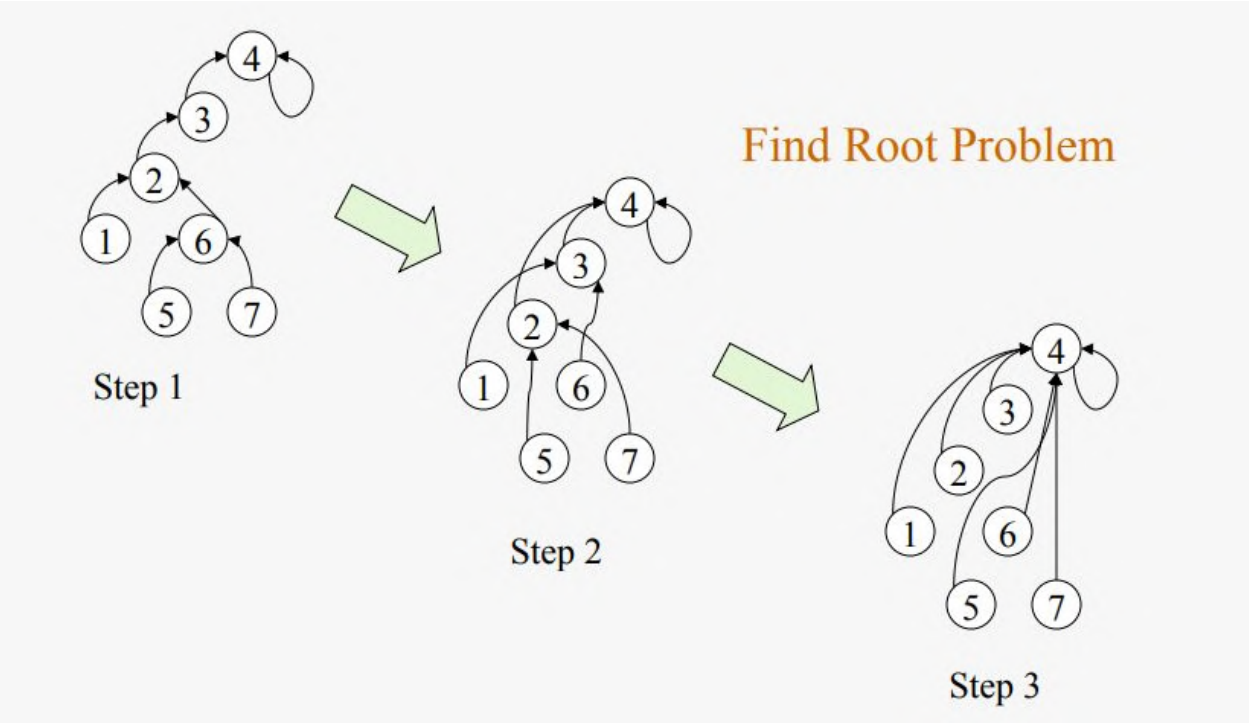
Replicable (reduce)



Divide & Conquer

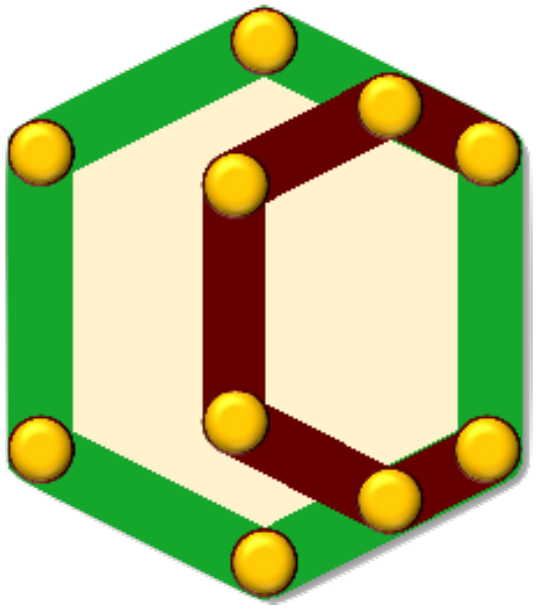


Recursive Data



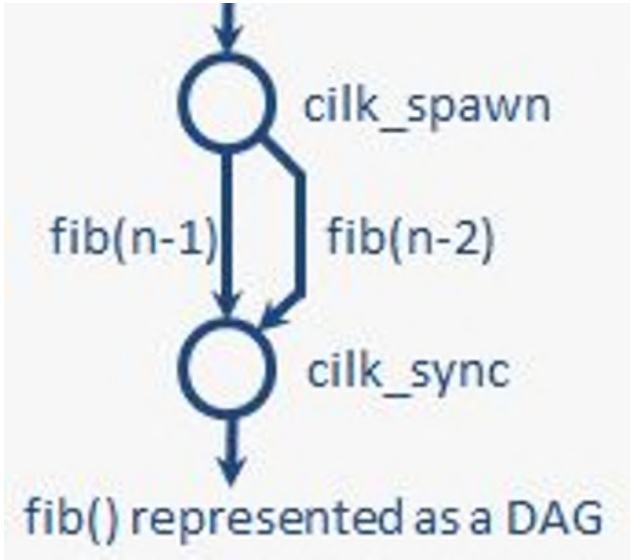
Multicore: OpenCilk

- Extensión de C y C++ para soportar programación paralela.
- Desarrollado inicialmente por Intel. Mantenido actualmente por el MIT.
- https://cilk.mit.edu
- https://github.com/opencilk



Keywords importantes

- parallel_for
- spawn
- sync

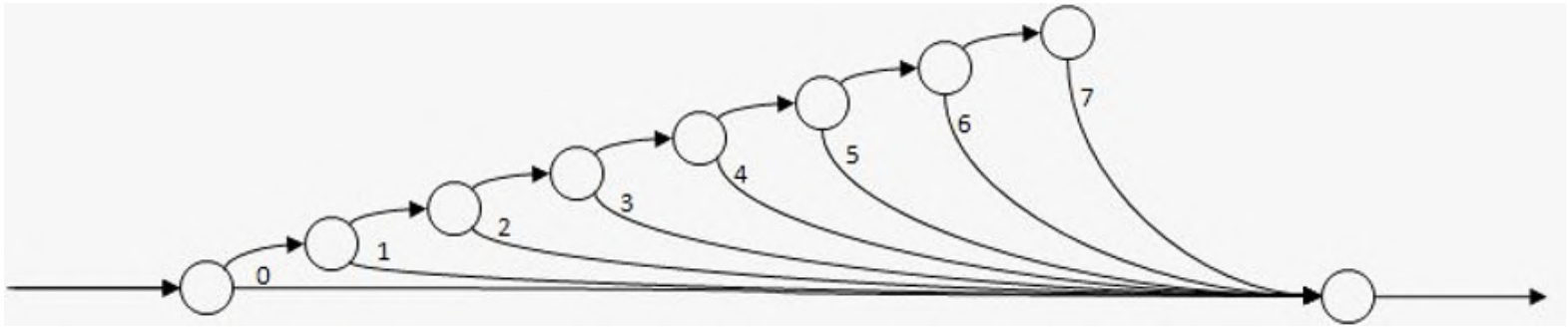


¿Cómo paralelizamos algo simple?

```
1 for (int i = 0; i < 8; ++i) {
2   do_work(i);
3 }
```

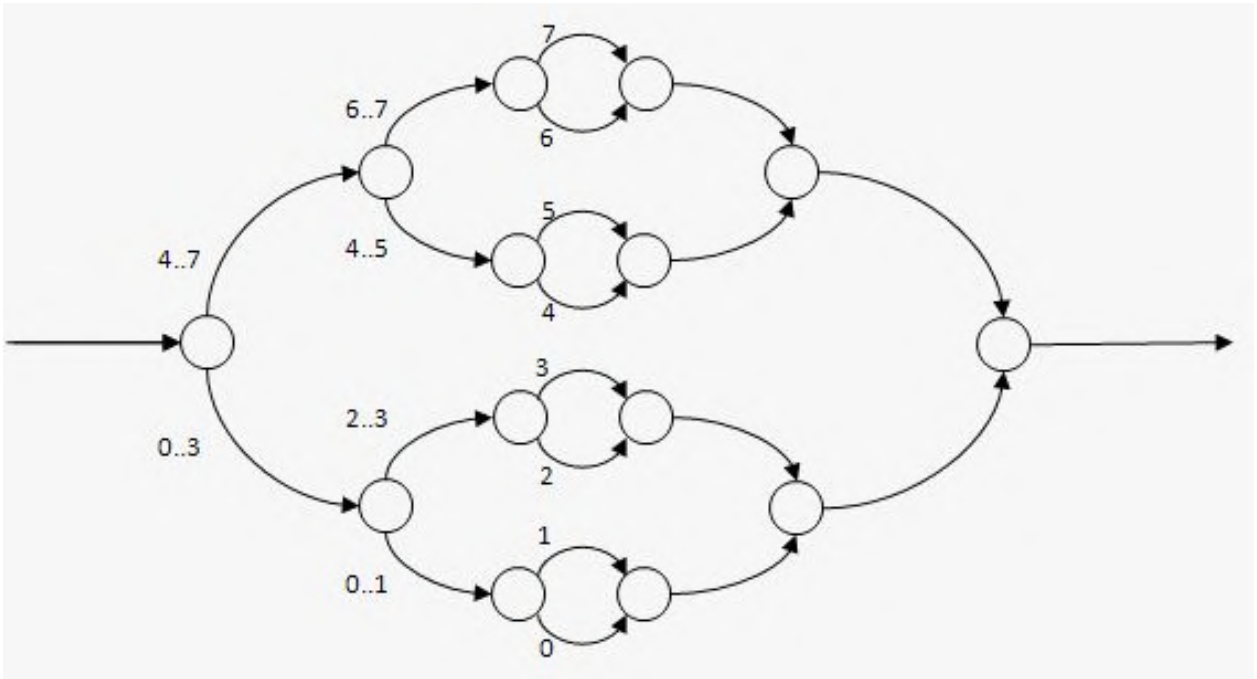
¿Cómo paralelizamos algo simple? – cilk_spawn

```
1 for (int i = 0; i < 8; ++i) {
2   cilk_spawn do_work(i);
3 }
4 cilk_sync;
```



¿Cómo paralelizamos algo simple? – cilk_for

```
1 cilk_for for (int i = 0; i < 8; ++i) {
2   do_work(i);
3 }
```



Fibonacci

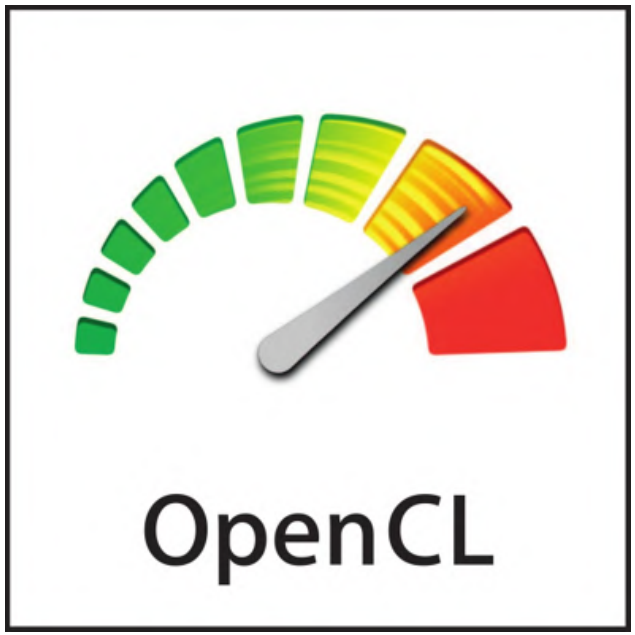
Código sin paralelizar:

```
1 int fib(int n) {
2   if (n < 2)
3   return n;
4   int x = fib(n-1);
5   int y = fib(n-2);
6   return x + y;
7 }
```

Código paralelizado:

```
1 int fib(int n) {
2   if (n < 2)
3   return n;
4   int x = cilk_spawn fib(n-1);
5   int y = cilk_spawn fib(n-2);
6   cilk_sync;
7   return x + y;
8 }
```

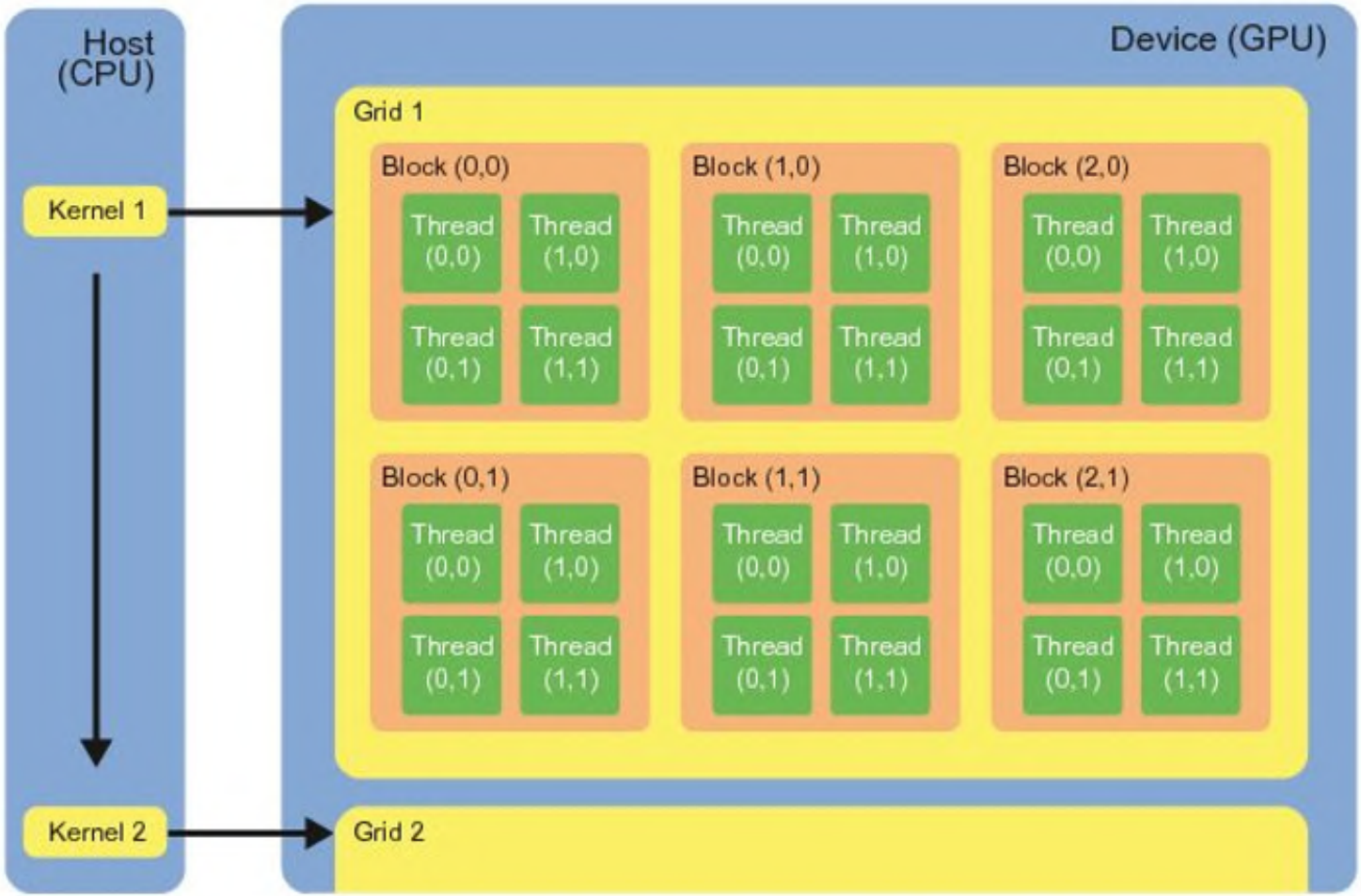
GPU: OpenCL y CUDA



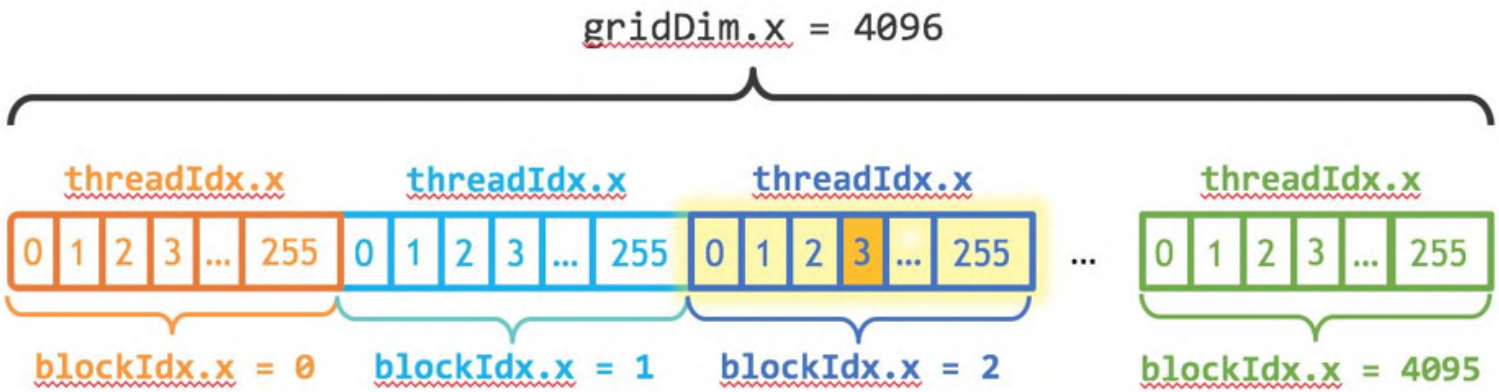
GPU: Keywords importantes

- Kernel
- Thread
- Block

Threads & Blocks



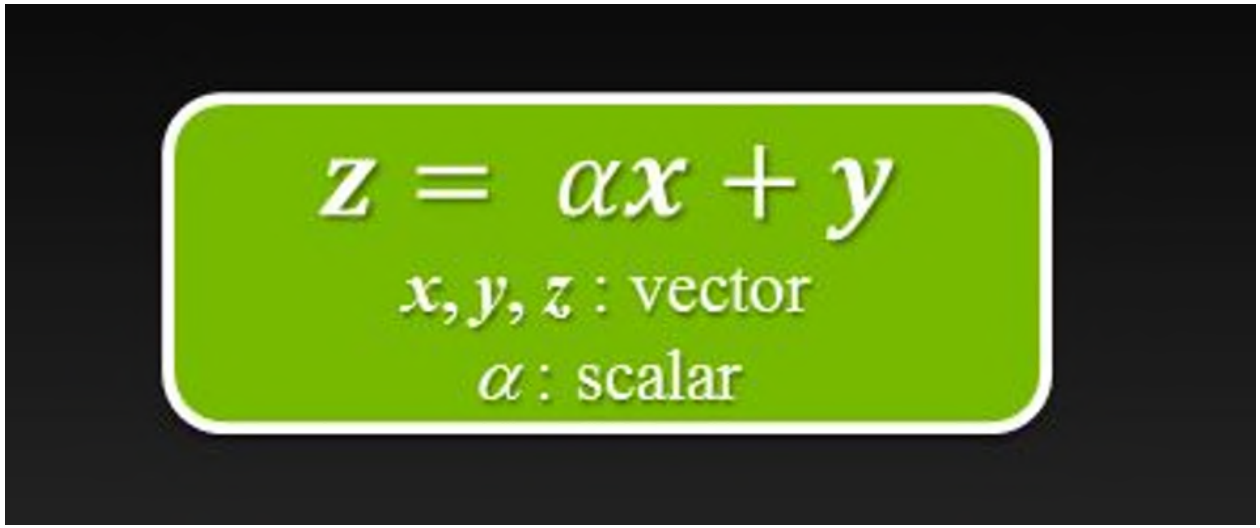
Threads & Blocks



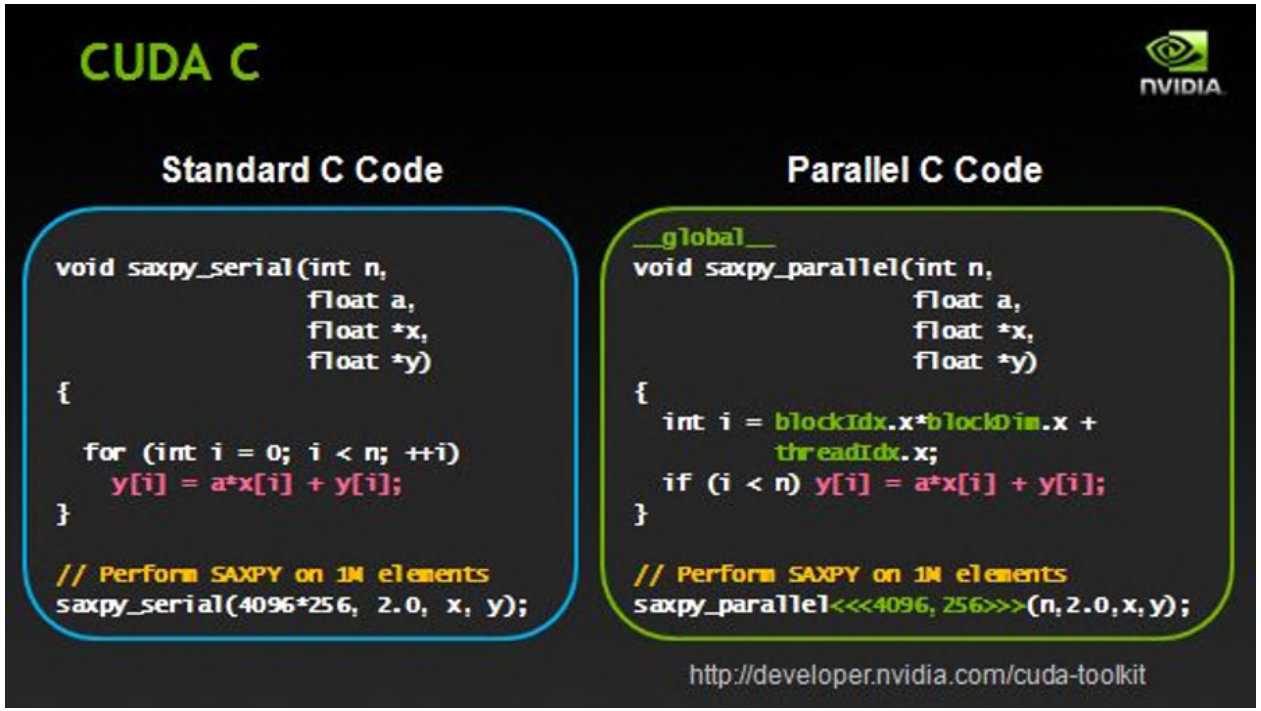

```
index = blockIdx.x  * blockDim.x + threadIdx.x

index =    (2)      *   (256)    +     (3)      = 515
```

Single Precision $\alpha A+B$



Single Precision $\alpha A+B$ – Ejemplo CUDA



Single Precision $\alpha A+B$ – Ejemplo Open-CL

```
1 __kernel void SAXPY (__global float* x, __global float* y, float a) {
2   const int i = get_global_id (0);
3   y [i] = a * x [i] + y [i];
4 }
```

Gracias por su atención