Programación Sobre Redes

T3: Sistemas multiprocesador y paralelismo II

Nicolás Mastropasqua

April 6, 2020

Instituto Industrial Luis A. Huergo

Contenidos

1. Repaso
2. Ley de Amdahl
3. Scheduling con multiprocesador

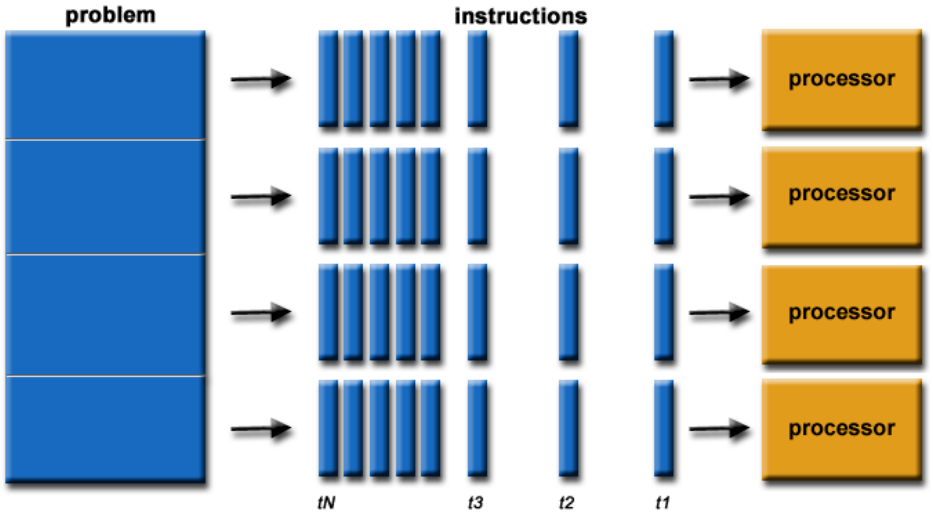
Repaso

Repaso

Previamente, en Programación sobre redes...

- Sistema de procesador único
- Sistemas multiprocesador
  - Asimétrico
  - Simétrico
- Computación paralela

Computación paralela



Idea

- Desarmar tareas en subtareas que ejecutan en unidades distintas
- Idealmente, con *n* CPUs tendríamos que la velocidad aumenta también con un factor *n*...
- Pero esto no siempre es posible! Recordar el problema de los pintores...

Ejercicio

Consideremos el algortmo conocido como ”La criba de Eratóstenes”, que dado un entero *n* obtiene todos los primos menores a *n*. ¿Podría paralelizarse? ¿Cómo?

```
Unmarked = {2,3,...,n}
for all a from 2 to sqrt(n):
    if a is in Unmarked then:
        for all m in muliplesLessThan(a,n):
            remove(Unmarked,m)
return unmarked
```

Ley de Amdahl

Pintores con habitaciones distintas

Pregunta: ¿Qué pasa si una de las habitaciones tiene el **doble de tamaño?**

- Intuitivamente, vemos que la ganancia es **solo 3 veces superior** a la de un solo pintor
- La **fracción** de la tarea que es paralelizable determinará la **ganancia** que obtendremos

Ley de Amdahl

- Se define S, el speedup de un proceso, como el ratio entre el tiempo que le toma a un procesador terminar un procesos y el tiempo que le toma a n procesadores terminar dicho proceso.
- Se define p, como la fracción del proceso que puede ser ejecutada en paralelo.
- Asumimos, por simplicidad, que le toma una unidad de tiempo terminar el procesos a un procesador.

$$S = \frac{1}{1 - p + \frac{p}{n}}$$

¿Tremendo galerazo?

Figure 1: ¿De dónde sale esta fórmula?

Demostración

Sea $T_s(W) = T$ el tiempo total de la ejecución serial para un tarea W y p la fracción paralelizable. Entonces

$$T_s(W) = T = T(1 - p + p) = (1 - p)T + pT$$

Por otro lado, sea $T_p(W)$ el tiempo total de la ejecución paralelizado de la tarea W. Sabemos que solo la parte p se beneficia con esto y más aún, dicha fracción ahora toma $\frac{p}{n}$ pues tenemos n recursos para paralelizar. Entonces

$$T_p(W) = (1 - p)T + \frac{p}{n}T$$

Finalmente, el speedup es el ratio entre estos valores

$$S = \frac{T_s(W)}{T_p(W)} = \frac{T}{(1 - p)T + \frac{p}{n}T} = \frac{1}{(1 - p) + \frac{p}{n}}$$

Ejemplos

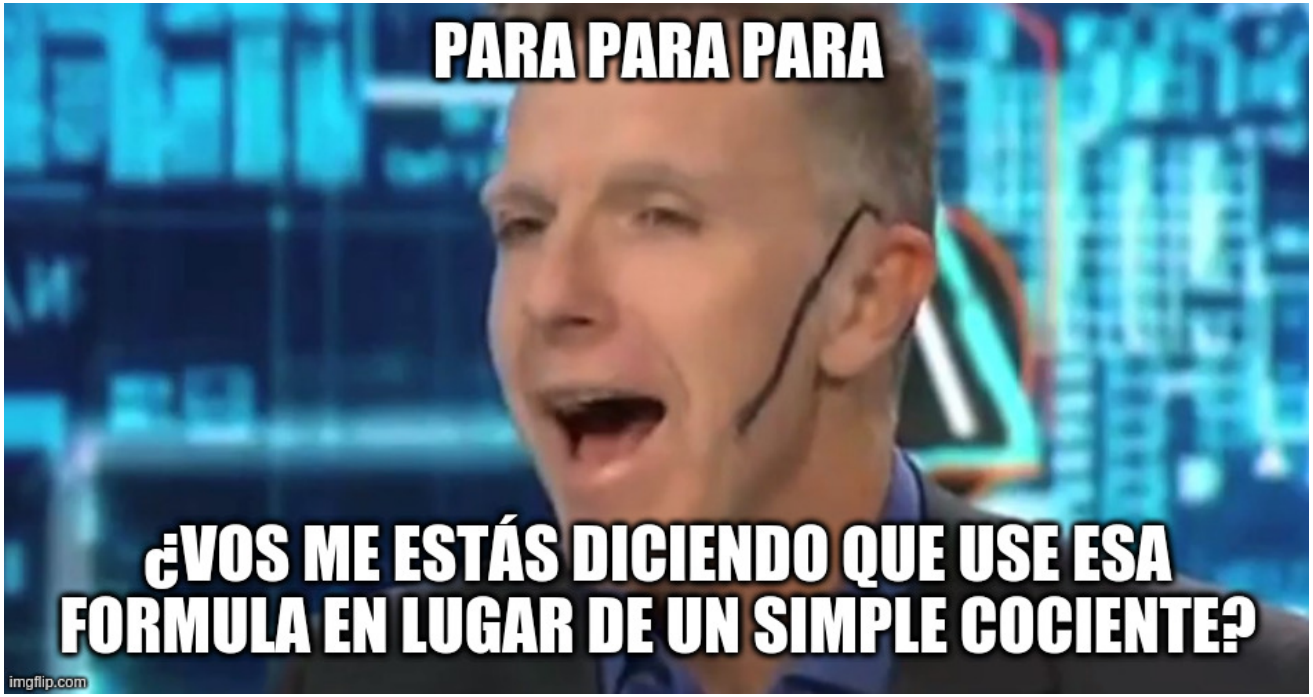
Veamos como aplicar esto al problema de los pintores

- Tenemos $p = \frac{5}{6}$ porque 5 de las 6 habitaciones (recordar que una tiene el doble de tamaño) pueden ser pintadas en paralelo.
- Por lo tanto $1 - p = \frac{1}{6}$
- La cantidad de recursos para paralelizar son $n = 5$

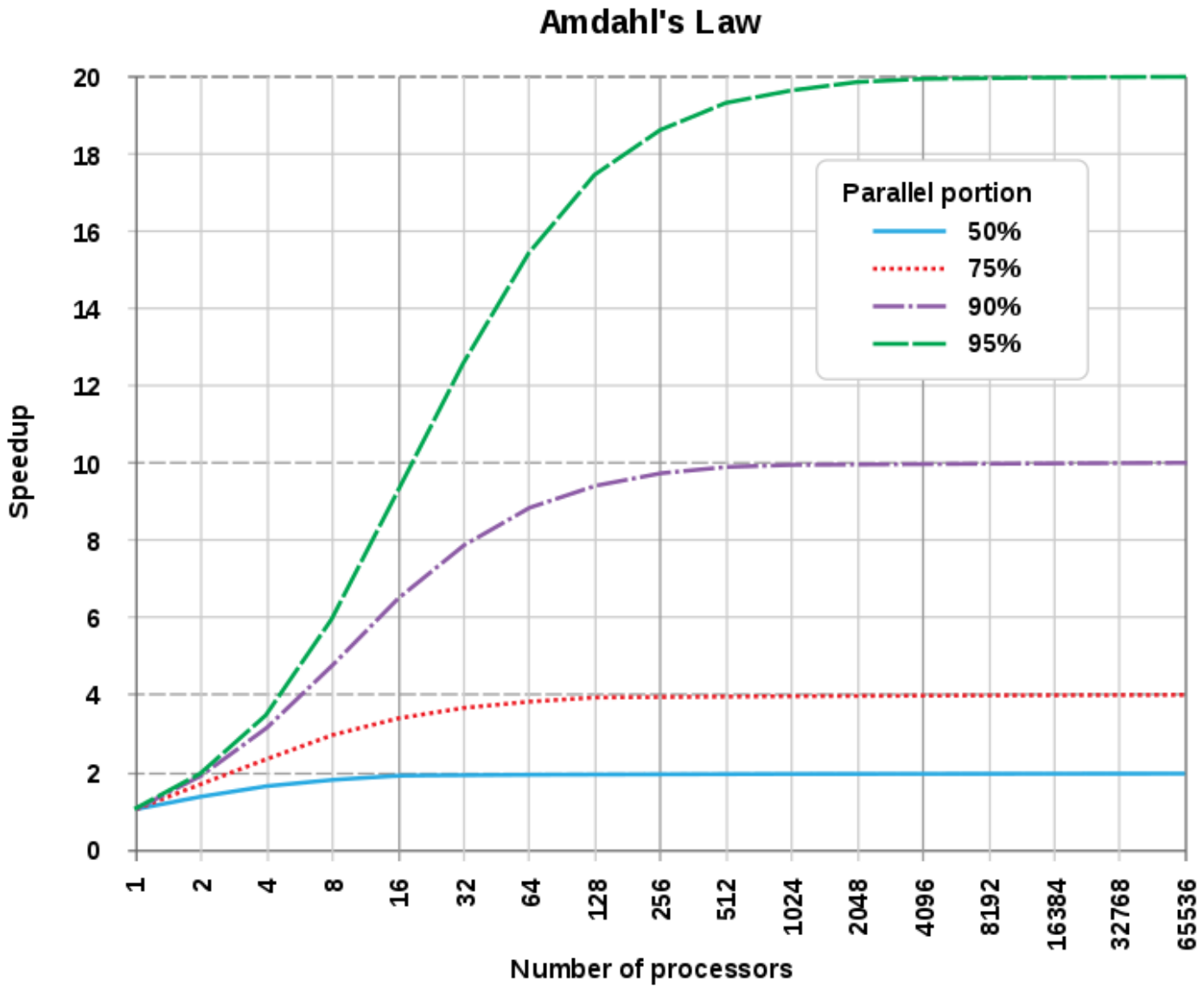
Resulta entonces

$$S = \frac{1}{1 - p + \frac{p}{n}} = \frac{1}{\frac{1}{6} + \frac{\frac{5}{6}}{5}} = \frac{1}{\frac{2}{6}} = 3$$

- Supongamos el mismo contexto, pero con 10 pintores. ¿Qué pasa con el speedup? S = 4 La situación empeora!
- En general si $n \to \inf$ entonces $S = \frac{1}{1-p}$


PARA PARA PARA
¿VOS ME ESTÁS DICIENDO QUE USE ESA FORMULA EN LUGAR DE UN SIMPLE COCIENTE?
imgflip.com

Análisis del speedup



Ejercicio

Enunciado

¿Cuántos procesadores son necesarios para obtener un speedup de 2 al paralelizar una tarea que requiere que al menos $\frac{1}{3}$ de su ejecución sea serial?

Scheduling con multiprocesador

Scheduling con multiprocesador

- Hasta ahora vimos distintas formas de hacer scheduling de procesos con **único procesador**
- ¿Cómo cambia la situación en el caso de tener **múltiples procesadores** y un esquema simétrico?
- Dos alternativas:
  - Todos los procesos están en única cola de listos
  - Cada procesador tiene su cola privada. En este caso, puede tener sentido implementar políticas de balanceo de carga
    - Push migration: Se monitorea la carga de cada CPU y en caso de desbalance, se redistribuyen las tareas
    - Pull migration: Si un CPU está IDLE, puede robar una tarea de otro ocupado

Afinidad

Memoria Cache

En general, se emplea esta memoria, más rápida que la RAM, para tratar de tener ”más a mano” las porciones de datos que ”más usa” un programa.

- Cuando desalojamos una tarea por una nueva, la Cache tiene que volver a llenarse según el contexto de la nueva
- Si hacemos esto muy seguido, estamos atacando la performance de cada tarea
- Por eso, el sistema operativo puede tener políticas de afinidad, más o menos duras, para evitar la migración de tareas entre procesadores

Ejercicio

Dados los siguientes procesos

Tarea	Duracion	Tiempo de llegada	E/S
T0	7	0	
T1	6	1	
T2	9	2	2-5

Dibujar el diagrama de GANTT para un scheduler RR con quantum = 2 y permitiendo la migración entre los tres núcleos (afinidad blanda)

¿Dudas?

Bibliografía

- *Abraham Silberschatz, Peter Baer Galvin, Greg Gagne.* ***Operating System Concepts Eigth Edition****, 5.5-5.5.4*
- Para leer más sobre la Ley de Amdahl