Programación y Análisis de Algoritmos

Dr. Norberto Alejandro Hernández Leandro

CIMAT-Unidad Monterrey

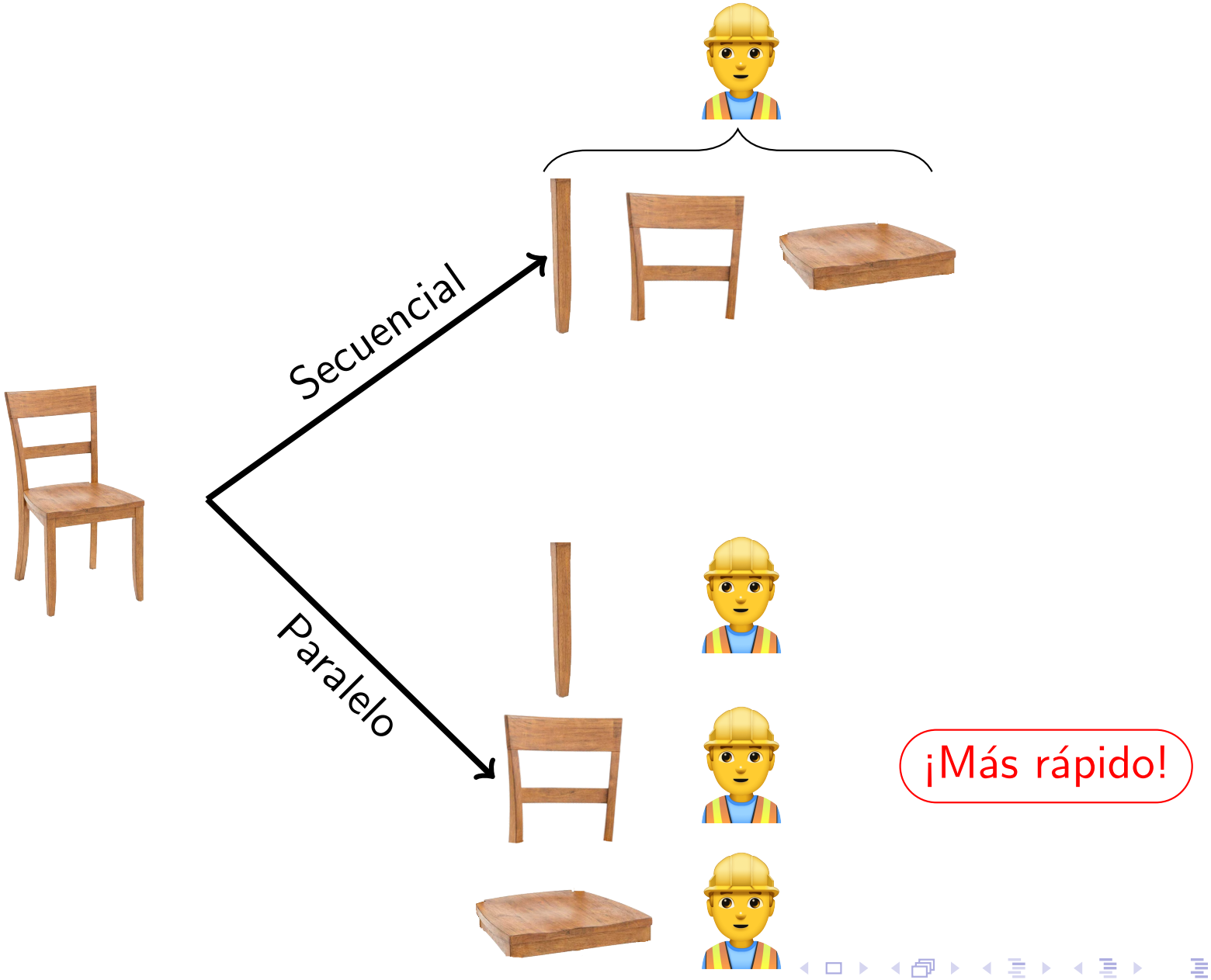
Noviembre 2020

Programación paralela

La programación paralela se define a partir de la ejecución simultánea de secciones de código sobre las unidades de procesamiento disponibles.

- **Computadoras**: a partir de una conexión local de red, las diferentes tareas a ejecutar se distribuye en las diferentes pc's conectadas en la red, y se ejecutan de manera simultánea.
- **CPU's**: las tareas se distribuye en los diferentes procesadores disponibles en la pc.
- **Núcleos**: las tareas son distribuidas sobre los núcleos de un procesador.
- **GPU's**: se utilizan los núcleos de la unidad de procesamiento gráfica para distribuir el trabajo.

Programación paralela



Programación paralela

Algunos datos relevantes:

- En 1958 se comenzó a hablar de la programación paralela para acelerar procesos pesados.
- En 1962, Burroughs Corporation, creo la primera computador con más de un procesador.
- En la decada de los 60's, 70's y 80's, se desarrollaron y crearon diferentes proyectos de procesamiento masivo, estableciendo las bases de la programación paralela.
- En los 90's se desarrollan las principales librerías para la codificación de algoritmos paralelizados (OpenMP y pthreads).

Programación paralela

Ventajas y desventajas

Ventajas:

- Permite ejecutar diferentes tareas de manera simultánea.
- Resuelve problemas en los que no es posible obtener una solución en un tiempo razonable.
- Acelera la ejecución de los algoritmos.
- Se pueden resolver problemas más complejos.

Desventajas:

- Mayor consumo de energía
- La codificación es más compleja.
- Pueden existir retardos causados por la sincronización y comunicación entre tareas.
- Es más complejo el control de errores.
- Hay que tener especial cuidado con la distribución de trabajo.

Programación paralela

Conceptos básicos

- **Tarea**: es una sección de código que se ejecutará en la unidad de procesamiento.
- **Hilo**: está definido por los procesos que se ejecutarán de manera simultánea.
- **Sincronización**: es un mecanismo que controla el orden de ejecución de las tareas, así como el acceso a la memoria compartida entre ellas.
- **Granularidad**: es un concepto que hace referencia al tamaño de las tareas y a la comunicación que existe entre ellas:
  - **Gruesa**: tareas pesadas, mucha independencia y poca sincronización.
  - **Fina**: tareas ligeras, poca independencia y mucha sincronización.
- **Scheduling**: representa un estrategia para definir el orden de la lista de hilos que se pasarán a las unidades de procesamiento.

Programación paralela

Diseño

Consideraciones:

- Los tiempos de las comunicaciones.
- Maximizar el tiempo de procesamiento de cada hilo.
- Los costes de implementar el algoritmo.
- Los tiempos del scheduling.

Existe una estrategia general para la paralelización de algoritmos por medio de la métodología Foster. Sin embargo, el llevarlo a cabo requiere de creatividad, experiencia y un alto nivel de conocimiento del algoritmo que se desea implementar.

Programación paralela

Tipos de paralelización

- **Datos**: cada unidad de procesamiento ejecuta la misma tarea sobre conjuntos de datos independientes.
- **Tareas**: cada unidad de procesamiento ejecuta una tarea distinta e independiente.

Programación paralela

Ley de Amdahl

Si α es la porción de tiempo que un programa gasta en ejecutar código no paralelizable, entonces

$$S_\alpha(P) = \frac{1}{\alpha + \frac{1-\alpha}{P}} \leq \frac{1}{\alpha}$$

representa la máxima aceleración que se puede alcanzar con la paralelización del algoritmo, donde P representa la cantidad de unidades de procesamiento. Por ejemplo, si $\alpha = 0.1$, entonces se logra una velocidad 10 veces superior.

Programación paralela

OpenMP vs Pthread

- OpenMp realiza una paralelización automática (generalmente sobre ciclos).
  - Los procedimiento identifican de manera automática las secciones paralelizables.
  - Menor control en la sincronización.
  - Puede no mejorar el rendimiento en la ejecución del código.
- Pthread se utilza para realizar una paralelización manual.
  - Se tiene mayor control sobre las tareas que se van a ejecutar de manera paralela.
  - Indica de manera explícita cómo se tiene que realizar la sincronización.
  - Si el algoritmo está bien diseñado, se logra una aceleración en la ejecución.

Programación paralela

pthread

- pthread es una librería para C/C++ que recopila un conjunto de procedimiento y tipos de datos que permiten la paralelización de código.
- Las hilos se definen mediante la palabra reservada pthread_t.

```
1       pthread_t thr;
```

- Las tareas que se desean ejecutar se definen mediante funciones que tienen como argumentos y retornos datos tipo void*

```
1       void* tarea(void *args){
2           Código de la tarea.
3       }
```

- La asignación de tareas a los hilos se realiza mediante las siguiente líneas de código:

```
1       pthread_attr_t attr;
2       pthread_attr_init(&attr);
3       pthread_create(&thr,&attr,tarea,args);
```

Programación paralela

pthread

- Si no se indica al código que tiene que esperar a que los hilos se terminen su ejecución, el programa terminará su ejecución y en segundo plano se seguiran ejecutando los hilos correspondientes.
- Así mismo, si no se indica la espera, tampoco se puede utilizar la información obtenida en cada tarea.
- Para indicar esta espera, se hace uso de la siguiente línea de código:

```
1       pthread_join(&thr,NULL);
```

Programación paralela

pthread

- Si existen **secciones críticas** (bloques de código que acceden a memoria compartida) en las tareas, se pueden generar conflictos al realizar modificaciones en sus valores.
- Dichas modificaciones se pueden controlar mediante el tipo de dato pthread_mutex_t, que permite bloquear y desbloquear estás secciones críticas para que sean ejecutadas por un sólo hilo a la vez, y evitar conflictos al realizar modificaciones.

```
1       pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
2       void* tarea(void *args){
3           ...
4               pthread_mutex_lock(&mutex);
5               Sección crítica
6               pthread_mutex_unlock(&mutex);
7           ...
8       }
```