Materia: Inf – 317 Sistemas paralelos y distribuidos

Nombre: Valle Fernandez Ariel Victor

CI: 14290216

Pregunta1

1. SISD (Single Instruction, Single Data)

En esta arquitectura, un único procesador ejecuta una sola instrucción sobre un único conjunto de datos a la vez. Esto significa que no existe paralelismo a nivel de datos ni de instrucciones. Es típico de las computadoras secuenciales tradicionales.

- **Ejemplo de uso**: Lenguajes de programación secuenciales como C, Python o Java pueden aplicarse en arquitecturas SISD.
- **Aplicación típica**: Computación de propósito general, donde se necesita un solo flujo de instrucciones y no se requiere paralelismo.

Ejemplo:

- **Descripción**: El procesador toma una sola instrucción, como sumar dos números, y la aplica a un único conjunto de datos.
- **Aplicación**: Un procesador SISD leería una operación de suma, tomaría dos números de entrada, realizaría la operación y devolvería un único resultado. Este tipo de arquitectura es común en las computadoras personales básicas que no requieren procesamiento paralelo.

```
# Python - Ejemplo básico de SISD para suma de dos números
a = 5
b = 10
resultado = a + b
print("Resultado:", resultado)
```

2. SIMD (Single Instruction, Multiple Data)

Esta arquitectura permite que un solo conjunto de instrucciones se aplique simultáneamente a múltiples conjuntos de datos. Es comúnmente utilizada en aplicaciones que requieren operaciones en vectores o matrices, como en procesamiento de gráficos y simulaciones científicas.

- **Ejemplo de uso**: Lenguajes con soporte para paralelismo de datos como CUDA y OpenCL para GPUs, así como extensiones en C++ (AVX, SSE) que permiten procesamiento en paralelo.
- **Aplicación típica**: Juegos, procesamiento de imágenes, y operaciones en matrices en ciencia de datos y aprendizaje automático.

Ejemplo:

- **Descripción**: Se utiliza una sola instrucción para aplicar la misma operación a múltiples datos en paralelo. Esto es común en GPUs (Unidades de Procesamiento Gráfico) para realizar operaciones masivas en vectores o matrices.
- **Aplicación**: Supongamos que se desea incrementar el valor de brillo en cada píxel de una imagen, y cada píxel puede procesarse de forma independiente pero usando la misma operación.

```
# Python - Ejemplo de SIMD para incrementar brillo en una imagen
usando NumPy
import numpy as np

# Imagen representada como una matriz (2D) de valores de brillo
imagen = np.array([[100, 150, 200], [125, 175, 225], [130, 180,
230]])
incremento = 20

# SIMD: Incrementar brillo en todos los píxeles
imagen_brillante = imagen + incremento
print("Imagen con brillo incrementado:\n", imagen_brillante)
```

3. MISD (Multiple Instruction, Single Data)

En esta arquitectura, múltiples procesadores ejecutan distintas instrucciones sobre el mismo conjunto de datos. Este enfoque es menos común en sistemas de computación debido a su complejidad y aplicación limitada.

- **Ejemplo de uso**: Esta arquitectura es inusual, pero algunas implementaciones especializadas, como Systolic Arrays, usan el concepto para tareas de procesamiento de señales.
- **Aplicación típica**: Sistemas de control redundante, donde es necesario verificar múltiples condiciones o realizar chequeos de integridad sobre los mismos datos (por ejemplo, en sistemas de control de vuelos o de seguridad crítica).

Ejemplo:

- **Descripción**: Varias instrucciones se aplican al mismo conjunto de datos, generalmente para redundancia y verificación. Este modelo es raro y se utiliza en sistemas donde la integridad de los datos es crítica.
- **Aplicación**: Supón que varios sistemas deben monitorear la altitud y cada uno ejecuta un cálculo diferente para confirmar que la altitud es correcta.

```
# Python - Ejemplo de MISD para verificar datos
dato_altitud = 30000  # Altitud en pies

# Múltiples verificaciones sobre el mismo dato
def verificar_minima(altitud):
```

```
    return altitud > 10000

def verificar_maxima(altitud):
    return altitud < 40000

def verificar_seguridad(altitud):
    return 20000 <= altitud <= 35000

# Aplicar las diferentes instrucciones al mismo dato
print("Verificación mínima:", verificar_minima(dato_altitud))
print("Verificación máxima:", verificar_maxima(dato_altitud))
print("Verificación de seguridad:",
verificar_seguridad(dato_altitud))
```

4. MIMD (Multiple Instruction, Multiple Data)

MIMD permite la ejecución de múltiples instrucciones sobre múltiples conjuntos de datos de manera simultánea. Este tipo de arquitectura es muy flexible y se utiliza comúnmente en sistemas de multiprocesamiento, donde cada procesador puede ejecutar diferentes tareas en paralelo.

- **Ejemplo de uso**: Lenguajes con soporte para concurrencia y multiprocesamiento como MPI (Message Passing Interface), OpenMP, Java (con hilos), y Python (con la biblioteca multiprocessing).
- **Aplicación típica**: Computación en la nube, servidores de alta capacidad, sistemas de supercomputación, y aplicaciones que requieren realizar múltiples tareas simultáneamente.

Ejemplo:

- **Descripción**: Cada procesador o núcleo puede ejecutar una instrucción diferente en un conjunto de datos diferente. Este modelo es común en sistemas de multiprocesamiento y servidores que manejan tareas diversas en paralelo.
- **Aplicación**: Supón que tienes un sistema de procesamiento de datos de sensores, donde cada procesador realiza cálculos específicos sobre datos de sensores diferentes.

```
# Python - Ejemplo de MIMD usando multiprocessing
import multiprocessing as mp

# Funciones diferentes aplicadas a diferentes datos
def calcular_promedio(datos):
    print("Promedio:", sum(datos) / len(datos))

def calcular_maximo(datos):
    print("Máximo:", max(datos))

def calcular_minimo(datos):
    print("Mínimo:", min(datos))
```

```
# Datos diferentes para cada proceso
datos1 = [5, 10, 15]
datos2 = [2, 8, 12]
datos3 = [20, 5, 10]

# Crear procesos para cada instrucción con conjuntos de datos
diferentes
procesos = [
    mp.Process(target=calcular_promedio, args=(datos1,)),
    mp.Process(target=calcular_maximo, args=(datos2,)),
    mp.Process(target=calcular_minimo, args=(datos3,))
]

# Iniciar y unir procesos
for proceso in procesos:
    proceso.start()
for proceso in procesos:
    proceso.join()
```