



Programación Paralela

2020-II

Prof. José Fiestas

November 10, 2020

Universidad Nacional de Ingeniería
jose.fiestas@uni.edu.pe

Unidad 1: Fundamentos de paralelismo y arquitecturas paralelas

Al finalizar la unidad, los alumnos conocen:

1. Transición del procesamiento secuencial al paralelo
2. Taxonomía de Flynn
3. Memoria compartida vs memoria distribuida.
4. Metas del Paralelismo: velocidad y precisión

Taxonomia de Flynn

Problemas de gran escala solucionables con computación paralela:

Kilo: 10^3 , 2^{10}

Mega: 10^6 , 2^{20}

Giga: 10^9 , 2^{30}

Tera: 10^{12} , 2^{40} (BigData)

Peta: 10^{15} , 2^{50} (BigData)

Exa: 10^{18} , 2^{60} (BigData)

¿Cuánto tiempo demorará resolver un sistema lineal de n ecuaciones?

Un sistema $Ax=b$ con n ecuaciones lineales necesita aproximadamente $n^3/3$ FLOPs (Floating Point Operations) I.e., en un procesador de 100 MFLOPs, se necesitarían:

n	FLOPs	tiempo (sec)
10	3.3×10^2	0.0000033
100	3.3×10^5	0.0033
1000	3.3×10^8	3.33
10000	3.3×10^{11}	3333 (55.5 minutos)
100000	3.3×10^{14}	333333 (926 horas)
1000000	3.3×10^{17}	926000 horas (105 años)

Además:

La matriz de $10^6 \times 10^6 = 10^{12}$ elementos, ocupa 8 TB de memoria

Paradigmas de programación en paralelo

De acuerdo al manejo del flujo de data se clasifican, según FLynn:

SIMD: Single Instruction Multiple Data, una única instrucción (tarea) se ejecuta en uno o más núcleos en múltiples **data streams**, en forma simultánea. E.g. GPUs

MIMD: Multiple Instruction Multiple Data, múltiples tareas en múltiples procesos se ejecutan en distintos **data streams** en forma simultánea . E.g. arquitecturas de memoria compartida o distribuída.

Se complementa con la definición de **SISD** (Single Instruction Single Data) de procesadores secuenciales, y **MISD** (Multiple Instruction Single Data). Este último no es de uso adecuado en paralelismo.

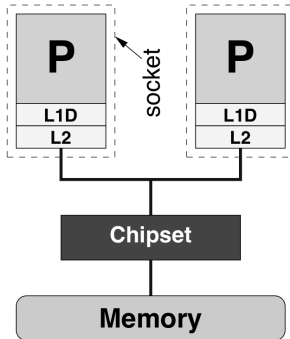
Memoria compartida vs memoria distribuida.

Computadoras de memoria compartida

Es un sistema en el que los CPUs comparten un mismo espacio físico (dirección). Dependiendo del acceso a memoria, pueden ser:

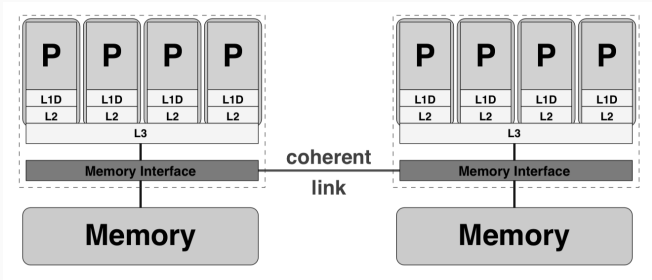
UMA: Uniform

Memory Access, es un modelo 'plano' de acceso a memoria (latencia = ancho de banda para todos los procesadores). También llamado Simmetric Multiprocessing (SMP).
E.g. procesadores multicore.



Computadoras de memoria compartida

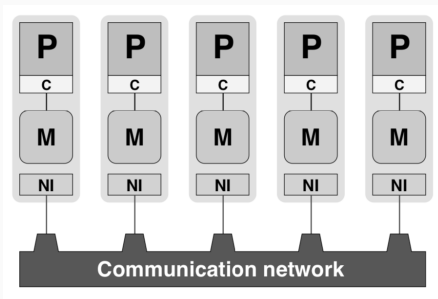
ccNUMA: Cache Coherent Non-uniform Memory Access, la memoria esta fisicamente distribuida, pero una red lógica permite que funcione como memoria en un mismo espacio físico (dirección)



En ambos casos la **coherencia de Caché** permite mantener consistencia entre las cachés y la memoria en caso se realicen modificaciones por algún CPU

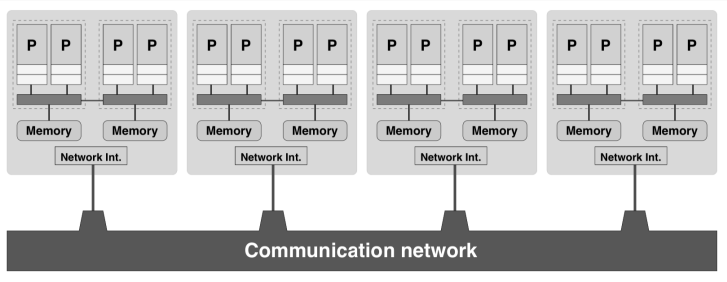
Computadoras de memoria distribuída

En este sistema, cada procesador está conectado a una memoria local, y ningún otro tiene acceso a esta memoria. Los procesadores se comunican a través de una red enviando a, y recibiendo mensajes de los otros procesadores.



Sistemas híbridos

En la práctica, los sistemas no son puramente distribuidos o compartidos, sino una combinación de ambos (híbridos). I.e., nodos de memoria compartida conectados a través de una red, que añaden una complejidad mayor al sistema de comunicación entre procesadores. E.g. CPU-GPU clusters



Arquitecturas en paralelo:

Paralelismo de la data (del resultado)

Paralelismo funcional (de la tarea)

Paralelismo del proceso (de la agenda)

Metas del paralelismo

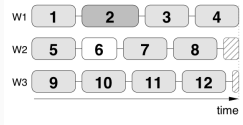
El problema de diseño de un algoritmo paralelo se refiere a

- Dado un algoritmo, buscar la arquitectura adecuada en paralelo
- Dada una arquitectura, buscar el algoritmo adecuado en paralelo. Es el caso clásico, que requiere :
 - Identificar **tareas** (tasks) con o por **procesos** (threads)
 - Diseñar un plan de ejecución (**scheduling**), de acuerdo a las dependencias y el I/O
 - Identificar los puntos de **comunicación** entre procesos y del I/O

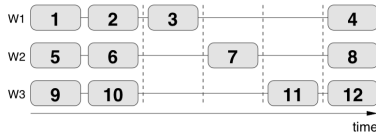
Problemas en el diseño en paralelo

Los principales problemas que aparecen son:

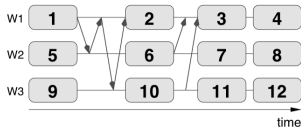
- Desbalance de cargas



- Cuellos de botella



- Comunicación



¿Cómo programar en paralelo?

- 1 - Escoger el paradigma (método) más cercano al problema
- 2 - Desarrollar código de acuerdo al paradigma elegido
- 3 - Si el código no es eficiente, retornar al paso 1

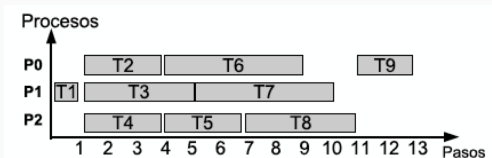
A tener en cuenta es el grado de **sincronización** y **comunicación** entre procesadores.

Un mal diseño de comunicación implica mayor costo computacional, mientras que una mala sincronización origina tiempos muertos (inactividad de procesadores)

Objetivos

Primer objetivo es *minimizar el tiempo total de ejecución*. I.e.

- **Tiempo de cálculo** propiamente dicho, se optimiza asignando tareas a procesadores distintos para maximizar la concurrencia
- **Tiempo de comunicación**, asignando tareas independientes en cada proceso
- **Tiempo de ocio**, evitar tiempos de inactividad.



Segundo objetivo es mantener (o mejorar) la *precisión del algoritmo* secuencial correspondiente.

Este no es un problema trivial

Ejercicios

Ejercicio 1

Dados dos procesadores y la tarea de actualizar la cuenta de un banco. Nombre dos problemas que hay que tener en cuenta al programar esta operación en paralelo en una arquitectura de memoria distribuída. Diseñe un pseudocódigo (enumere los pasos necesarios) para la solución de estos problemas

Proceso 0





```
x=leerCuenta(cuenta);  
x=x+500;  
escribirCuenta(x,cuenta)
```

Proceso 1

```
y=leerCuenta(cuenta);  
y=y+200;  
escribirCuenta(y,cuenta)
```

Ejercicio 2

1. Diagrame en un árbol, el proceso de la suma de n enteros en paralelo. Se tiene $n=16$ y disponibles 8 procesadores.
2. ¿Cuál será la forma más eficiente de hacer el cálculo si se tienen 7 procesadores?
3. ¿Qué método de paralelismo utiliza en cada caso?

-  David B. Kirk and Wen-mei W. Hwu *Programming Massively Parallel Processors: A Hands-on Approach*. 2nd. Morgan Kaufmann, 2013. isbn: 978-0-12-415992-1.
-  Norm Matloff. *Programming on Parallel Machines*. University of California, Davis, 2014.
-  Peter S. Pacheco. *An Introduction to Parallel Programming*. 1st. Morgan Kaufmann, 2011. isbn: 978-0-12-374260- 5.
-  Michael J. Quinn. *Parallel Programming in C with MPI and OpenMP*. 1st. McGraw-Hill Education Group, 2003. isbn: 0071232656.
-  Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Program- ming*. 1st. Addison-Wesley Professional, 2010. isbn: 0131387685, 9780131387683.