Universidad Católica San Pablo

Parallel Computing

Introduction

No existe ninguna computadora que no sea capaz de paralelismo

Evolucion de las computadoras

- Desde 1986 hasta 2002 el rendimiento de los procesadores incrementa del 50% cada año
- En 2011 solamente incrementa del 20%

- Solución ?
  - **Múltiples núcleos**
- **Problema :** Agregar más procesadores no mejora mágicamente los programas secuenciales

Porque necesitamos sistemas paralelos ?

- 20 % por año no es suficiente ?
- Los últimos años han conocidos avances muy importantes en ciencias
  - Procesamiento de imágenes/mallas
  - Análisis de datos (Big data)
  - Inteligencia Artificial
  - Simulaciones (previsiones climáticas)

Porque no podemos mejorar estos 20% ?

- Se incrementa la cantidad de transistores
  - Más pequeños
  - Más veloces
  - Mas energia -> mas calientes
- Entre más calientes menos fiables (problema de fiabilidad en supercomputadores)
- Lo que podemos incrementar (por el momento), es la cantidad de procesadores
- Procesadores multi-núcleos

Los programas no son directamente paralelizables ?

- Los programas han sido pensado de forma secuencial
- Hubo intentos para paralelizar automáticamente programas sin mucho éxito
- Pasar un algoritmo de forma secuencial -> paralelo puede resultar ineficientes
    - **Entonces tenemos que pensarlo un poco más**

Ejemplo de la suma global

Codigo serial :

```
sum = 0;

for (i = 0; i < n; i++) {
        x = Compute next value(. . .);
        sum += x;
 }
```

```
my sum = 0;
my first i = . . . ;
my last i = . . . ;
for (my i = my first i; my i < my last i; my i++) {
        my x = Compute next value(. . .);
        my sum += my x;
 }
```

1,4,3, 9,2,8, 5,1,1, 6,2,7, 2,5,0, 4,1,8, 6,5,1, 2,3,9,

Core	0	1	2	3	4	5	6	7
my_sum	8	19	7	15	7	13	12	14

Ejemplo de la suma global

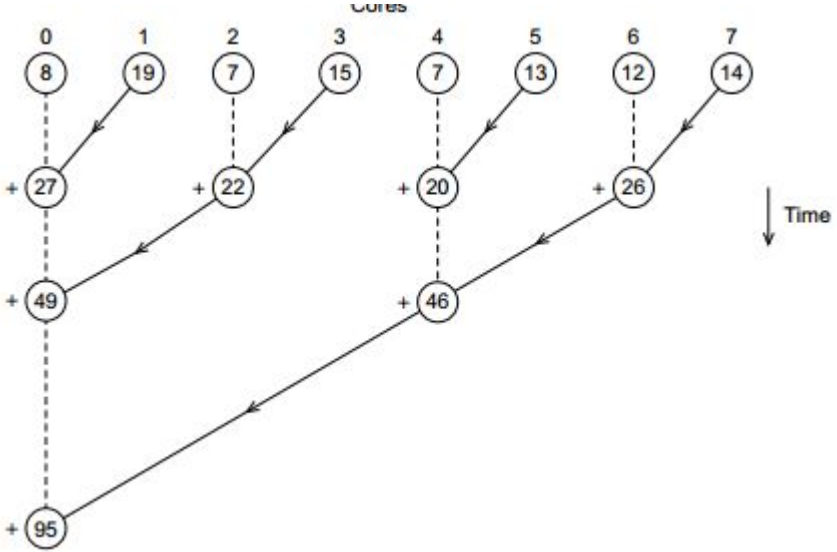
```
if (I'm the master core) {
        sum = my x;
        for each core other than myself {
                receive value from core;
                sum += value;
         }
} else {
        send my x to the master;
}
```

1,4,3, 9,2,8, 5,1,1, 6,2,7, 2,5,0, 4,1,8, 6,5,1, 2,3,9,

Core	0	1	2	3	4	5	6	7
my_sum	8	19	7	15	7	13	12	14

Ejemplo de la suma global

- Si la cantidad de núcleos es muy grande
  - el núcleo principal hará mucho trabajo

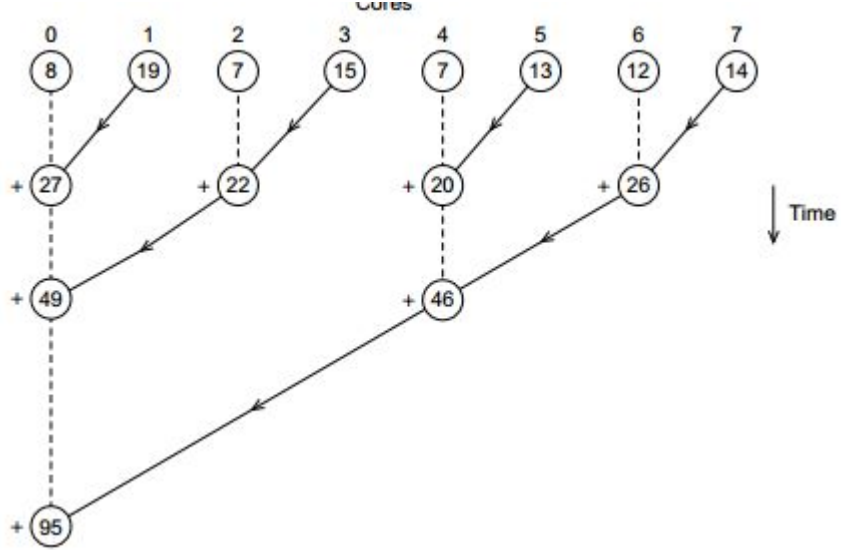


Cómo escribir programas paralelos ?

- Task parallelism : distribuir diferentes tareas a varios núcleos
- Data parallelism : distribuir los datos a varios núcleos
- Ej.: Prof princial tiene 100 alumnos, 5 profesores asistentes, un examen a 5 preguntas
  - Cada profesor asistente corrige 20 alumnos (data parallelism)
    - Cada núcleo hace los mismo
  - Cada profesor asistente corrige 1 pregunta para los 100 alumnos (task paraellism)
    - Cada uno de los núcleos hace algo distinto

Cómo escribir programas paralelos ?

- En el ejemplo de la suma la primera parte puede ser considerada como
  - data parallelism
- La segunda parte sería
  - Task parallelism
  - El núcleo principal recibe y agrega las sumas
  - Los otros núcleos mandan su suma parcial
- Cuando los núcleos pueden hacer un trabajo independiente entonces no hay dificultad
- Mientras que cuando necesitan coordinación las cosas se pueden poner más complejas



Cómo escribir programas paralelos ?

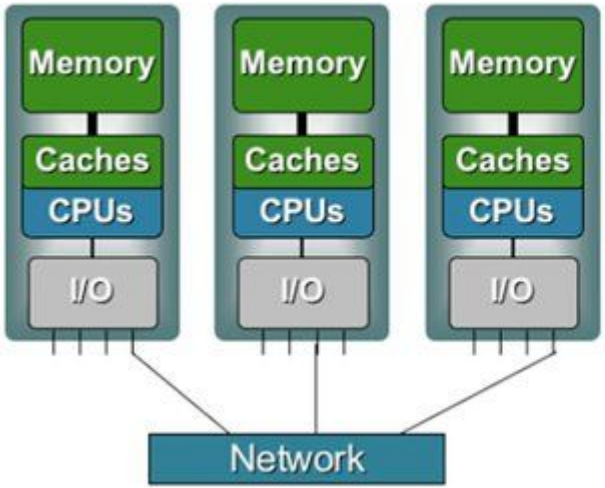
- Comunicación entre núcleos
- Load balancing
  - Queremos aprovechar al máximo cada núcleo
  - sobrecargar un núcleo significa malgastar los demás
- Sincronización
  - Generalmente los programas más eficientes utilizan una sincronización explícita
    - Ej. core 1 task 1 + core 2 task 2 -> synchronize ....
    - Esto puede generar programas complejos
    - Existen métodos de más alto nivel donde se sacrifica rendimiento contra simplicidad (OpenMP)

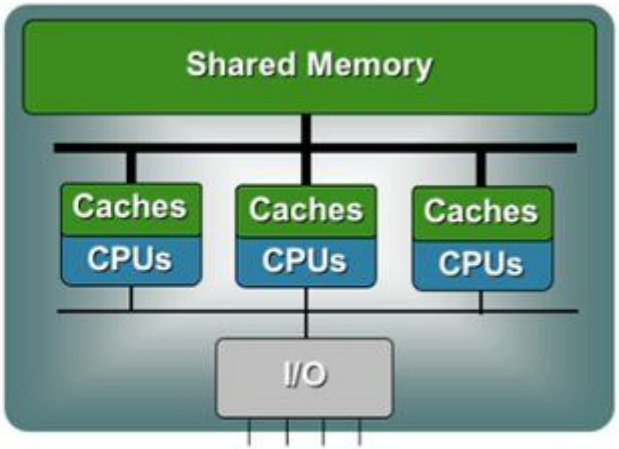
Herramientas

- OpenMP
- Thread C++ 11 o Pthread
- MPI Message Passing Interface

- Porque 3 extensiones del C++:
- Existen 2 tipos de sistemas paralelos
  - **Shared memory**
  - **Distributed memory**

Sistemas paralelos





Herramientas

- Shared memory
  - Núcleos pueden compartir el acceso la memoria principal de la computadora
  - Podemos coordinar los núcleos para observar y modificar la memoria
- Distributed memory
  - Cada núcleo tiene su propia memoria
  - Los núcleos deben de comunicar de forma explícita a través de red mandando “mensajes”
- Threads y OpenMP son diseñado para sistemas de shared-memory.
  - OpenMP hi-level, accesible
  - Threads coordinación entre núcleos
- MPI está diseñado para trabajar con sistemas distribuidos
  - Mandar mensajes

Parallel software
Parallel hardware

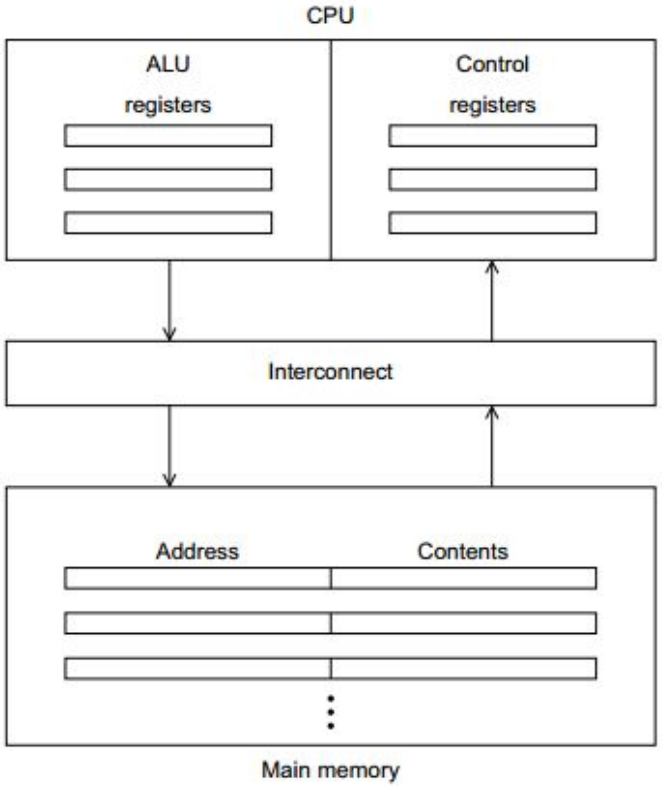
Software y Hardware paralelo - Objetivos

- Diferentes tipos de software
- Sistemas paralelos
- Evaluación de programas paralelos
- Método para el desarrollo de software paralelo

Sistema Serial - Arquitectura Von Neumann

- Main Memory
  - Localizaciones(datos, instrucciones)
  - Direcciones
- CPU
  - Control Unit (Que instrucción debe ejecutarse)
  - Arithmetic and Logic Unit (ALU) (Ejecutar la instrucción)
  - Registers (datos almacenados en el CPU)
    - Program counter (CU): almacena la dirección de la siguiente instrucción
    - Instrucción y datos
- Interconnection
  - Bus (parallel wires)

Sistema Serial - Arquitectura Von Neumann



Sistema Serial - Arquitectura Von Neumann

- Memorio transferida Memoria - CPU
  - Read/Fetch
- Memoria transferida CPU - Memoria
  - Write/stored
- Bus -> Von Neumann bottleneck
  - 2010 CPU puede ejecutar una instrucción más de 100 veces más rápido que leer un dato de la memoria
  - analogía a una fábrica que produce rápidamente y se encuentra esperando para materia prima
- **Veremos modificaciones hechas al modelo de Von Neumann para mejorar este problema**

Threads - multitasking

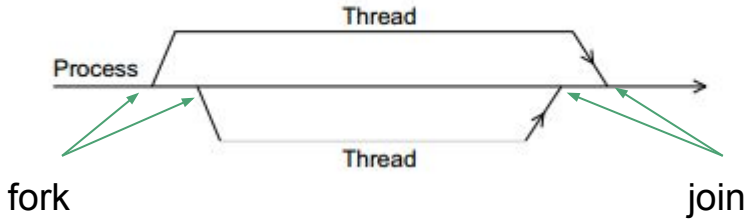
- Operating system
  - Software que maneja el hardware y recursos del software
- Process (Programa manejado por el OS)
  - Ejecutable (lenguaje máquina)
  - Bloque de memoria que contendrá
    - Código ejecutable
    - Call Stack (Variables temporales funciones)
    - Heap (asignación dinámica)
  - Información sobre el proceso (listo, esperando recursos etc.)

Threads - multitasking

- La mayoría de los sistemas son multitasking
  - Sistema maneja la ejecución de varios programas a la vez
  - Posible con un solo núcleo
    - **Time slice** : proceso corre sobre una pequeña unidad de tiempo
    - El OS puede cambiar muchas veces de proceso en un solo minuto
  - Un sistema operativo puede **bloquear** un proceso cuando le falta un recurso
    - Deja de ejecutarse
    - Hay una forma de seguir ejecutando un programa que espera un recurso
      - **Threading**

Threads - multitasking

- Threading
  - Divide un programa en múltiples tareas independientes
  - Un thread bloqueado mientras otro puede correr
  - **Es más rápido cambiar de thread** (lighter weight) que de proceso
    - Pertenecen al mismo proceso
    - Comparten la mayoría de los recursos del proceso
    - Call stack / Program counter independiente



Arquitectura Von Neumann - Modificaciones

- **Caching**
- **Virtual Memory**
- **Low level parallelism**

Caching

- Ej. Fábrica que produce rapido:
  - mejoramos el acceso a la fábrica
  - movemos el almacenamiento junto a la fábrica
- Mejor interconexión que permite transportar más datos en un solo acceso
- Dejamos de utilizar solo la main memory sino bloques de memoria cerca los registros del procesador
- Cache
  - Colección de localización de memoria que pueden ser accedidos rápidamente

Caching

- **Que datos poner en caché ?**
  - Datos e instrucciones físicamente cerca
  - Ej.: Después de ejecutar una instrucción generalmente ejecutamos la instrucción siguiente
  - Ej.: Después de acceder a una localización en memoria accedimos a la localización siguiente (v.at(i)<v.at(i+1))
- **Los arrays son asignados como bloques consecutivos en memoria**
  - **Locality (spatial/temporal locality)**
  - **Cache block 8x o 16x el tamaño de una locación en memoria**
  - **v[0] -> v[15]**

Caching

- Arquitectura
  - L1 - pequeño pero rápido
  - L2/L3 - grande y más lentos
- Cache hit
  - Cuando se encontró una variable en el caché
- Cache miss
  - Primero L1, después L2 etc. hasta la main memory (jerarquía)
- Inconsistency
  - CPU escribe en el cache -> cache != main memory
  - **write through** caches (escribimos en el caché + main memory)
  - **write back** caches (localización en el cache es marcada “dirty” así que antes de reemplazarla se escribirá en la main memory)

Cache mapping

- Fully associative
- Direct mapped
- n-way set associative

- Como saber cual reemplazar (evicted)
  - Se escoge la localización con el último acceso más antiguo

Memory Index	Cache Location		
	Fully Assoc	*Direct Mapped*	*2-way*
0	0, 1, 2, or 3	0	0 or 1
1	0, 1, 2, or 3	1	2 or 3
2	0, 1, 2, or 3	2	0 or 1
3	0, 1, 2, or 3	3	2 or 3
4	0, 1, 2, or 3	0	0 or 1
5	0, 1, 2, or 3	1	2 or 3
6	0, 1, 2, or 3	2	0 or 1
7	0, 1, 2, or 3	3	2 or 3
8	0, 1, 2, or 3	0	0 or 1
9	0, 1, 2, or 3	1	2 or 3
10	0, 1, 2, or 3	2	0 or 1
11	0, 1, 2, or 3	3	2 or 3
12	0, 1, 2, or 3	0	0 or 1
13	0, 1, 2, or 3	1	2 or 3
14	0, 1, 2, or 3	2	0 or 1
15	0, 1, 2, or 3	3	2 or 3

Cache - Consecuencias en programación

```
double A[MAX][MAX], x[MAX], y[MAX];
/* 1*/
for (i = 0; i < MAX; i++)
    for (j = 0; j < MAX; j++)
        y[i] += A[i][j]*x[j];

 /* 2*/
for (j = 0; j < MAX; j++)
      for (i = 0; i < MAX; i++)
          y[i] += A[i][j]*x[j];
```

Cache - Consecuencias en programación

```
double A[MAX][MAX], x[MAX], y[MAX];
/* 1*/
for (i = 0; i < MAX; i++)
    for (j = 0; j < MAX; j++)
        y[i] += A[i][j]*x[j];

 /* 2*/
for (j = 0; j < MAX; j++)
     for (i = 0; i < MAX; i++)
        y[i] += A[i][j]*x[j];
```

Cache Line	Elements of A			
0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	A[1][0]	A[1][1]	A[1][2]	A[1][3]
2	A[2][0]	A[2][1]	A[2][2]	A[2][3]
3	A[3][0]	A[3][1]	A[3][2]	A[3][3]

Ej. MAX = 1000 (1) es 3 veces más rápido

Memoria Virtual

- En sistema multi-tasking la main memory no podrá almacenar todos los datos e instrucciones de todos los programas
- Memoria Virtual
  - La main memory es como un caché para la memoria secundaria
  - Se guarda en la main memory solamente las partes activas de los programas en ejecución
  - Lo demás está almacenado en el **swap space** de la memoria secundaria
  - Trabaja sobre bloques en memoria llama **pages (tamaño fijo 8-16kb)**

Instruction level parallelism

- Mejorar el rendimiento varios **functional units** en el CPU
  - Ejecutar varias instrucciones simultáneamente
- Pipelining
  - functional units organizados en etapas
- Multiple Issue
  - Réplica funcional unit
  - **Static (compilation)**
  - **Dynamic (Super scalar) runtime**
- **ILP es ejecutado en casos donde el sistema puede predecir las posibilidades de paralelismo**
  - **Casos no son evidentes (números de Fibonacci)**

Hardware multithreading

- **El sistema debe ser capaz de cambiar de threads mucha mas rápido que de proceso**

- **Thread level parallelism (TLP)**
  - **Coarsed-grained parallelism vs finer-grained (ILP)**
  - **Partes del programa que seran ejecutado en paralelo**

Parallel Hardware

- Clasificación de Flynn
- Von Neumann : Single Instruction Stream, Single Data Stream SISD
  - Solamente ejecuta un instrucción a la vez
  - lee y escribe un dato a la vez

Parallel Hardware

- Single Instruction, Multiple Data SIMD (sistema paralelo)
  - Aplicar la misma instrucción sobre múltiples datos
  - Ej.: Un CPU con múltiples ALU, una instrucción llega los diferentes ALU

```
for (i = 0; i < n; i++)
        x[i] += y[i];
```

- Si tenemos n datos y m ALUs (m<n)
  - Podemos ejecutar bloques de m datos a la vez
  - Ej.: m=4 n=15, primero: 0-3,4-7,8-11, 12-14(uno de los ALU esperara)

Parallel Hardware

- Se requiere que todos los ALUs ejecuten las mismas instrucciones
    - Este puede degradar el rendimiento del sistema

```
for (i = 0; i < n; i++)
    if (y[i] > 0.0)
        x[i] += y[i];
```

- Si la condición no se cumple entonces el ALU esperara hasta que los demás hayan terminado
- SIMD -> ejecución sincronizada de las instrucciones entre los ALUs
- SIMD funcionan muy bien para procesar grandes array de datos
- **DATA-PARALLELISM**

Parallel Hardware

- Aspectos de SIMD es utilizado en los GPUs y CPUs actuales
- Vector processor
  - Procesadores que trabajan sobre array grandes
  - Permite leer, escribir, procesar grandes datos con bucles simples
- Compiladores informan porque un bucle no puedo ser vectorizado
- No es muy escalable, se incrementa la cantidad de procesadores, no el tamaño del vector
- Muy caros

High Performance Computing

- Arquitectura heterogénea CPU-GPU
- Computación paralela consiste en llevar a cabo cálculos al mismo tiempo
  - Arquitectura de la computadora
  - Programación paralela

Parallel Hardware

- Graphics processing units (GPU)
- Pipeline permite convertir geometría en un buffer de píxeles
- Utiliza tradicionalmente shaders
  - miniprogramas implícitamente paralelos
- SIMD, gran cantidad de ALUs
- No SIMD puro porque los GPUs actuales pueden ejecutar varios flujos de instrucciones a la vez.
- Varios lenguajes de programación han sido creados dedicados al GPGPU

Parallel Hardware

- Multiple Instruction, Multiple Data SIMD MIMD