CLUSTER FING: ARQUITECTURA Y APLICACIONES

Gerardo Ares, Pablo Ezzatti

Centro de Cálculo, Instituto de Computación

FACULTAD DE INGENIERÍA, UNIVERSIDAD DE LA REPÚBLICA, URUGUAY

CONTENIDO

- Introducción
- Conceptos básicos de Procesos
- Arquitectura de memoria compartida
- Threads
- OpenMP, HPF, etc.
- Ejemplos de utilización

INTRODUCCIÓN

- Programación paralela
  - Varios procesos trabajan cooperativamente en la resolución de un problema (complejo).
  - Objetivos:
    - Mejorar el desempeño.
    - Escalabilidad incremental.
- Paradigmas de programación paralela:
  - **Paralelismo de memoria compartida**
    - **Comunicaciones y sincronizaciones mediante recurso común (memoria).**
  - Paralelismo de memoria distribuida
    - Comunicaciones y sincronizaciones mediante pasaje de mensajes explícitos.

INTRODUCCIÓN

- **Paralelismo de memoria compartida**
  - **Comunicaciones y sincronizaciones mediante recurso común (memoria).**
  - **Es necesario sincronizar el acceso y garantizar exclusión mutua a secciones compartidas.**
  - **Paralelismo multithreading:**
    - **Bibliotecas estándares (e.g., en lenguaje C).**
    - **Bibliotecas específicas.**

PROCESOS

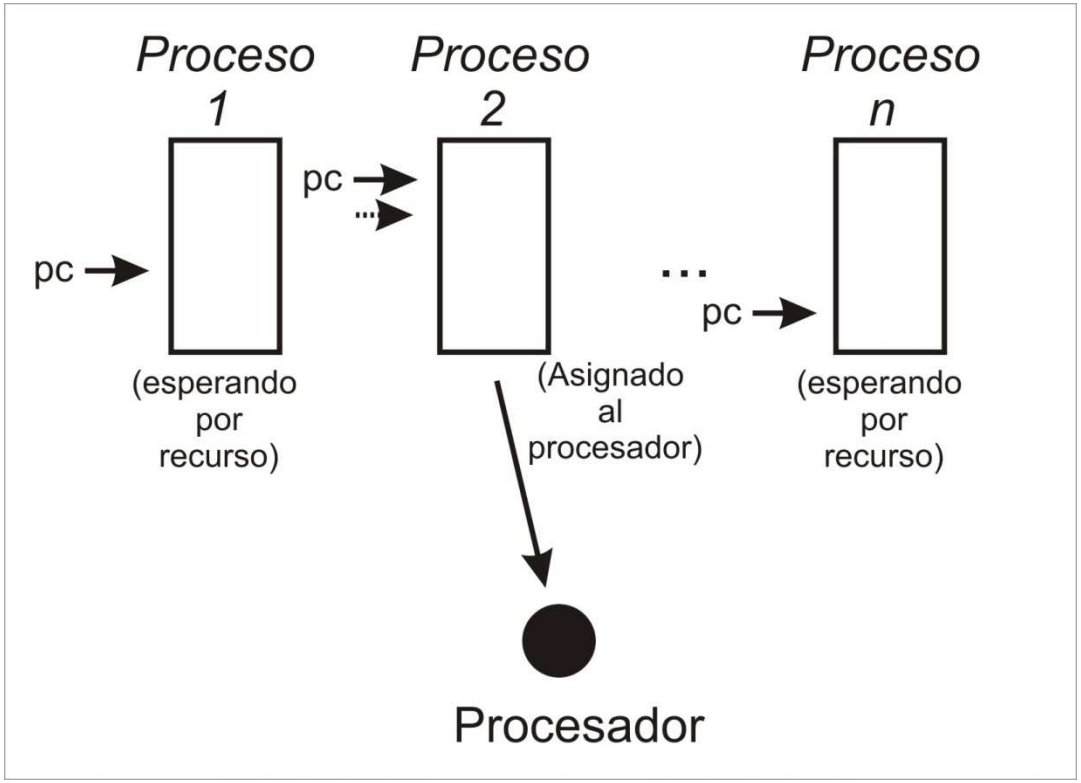
Que es ?

Los sistemas operativos definen el concepto de proceso como un programa en ejecución.

- Los procesos se componen de un conjunto de instrucciones a ejecutar que operan sobre su estado (memoria del proceso).
- Cada proceso tendrá un espacio de memoria asignado. En los sistemas operativos modernos este espacio de memoria es denominado espacio de direccionamiento virtual de un proceso (*virtual address space*).
- A su vez, cada proceso deberá contar con al menos un contador de programa (PC = *program counter*), denominado hilo de ejecución (*thread*).
- El contador de programa es un registro que indica cuál de todas las instrucciones del proceso es la siguiente a ejecutar por el proceso.

PROCESOS

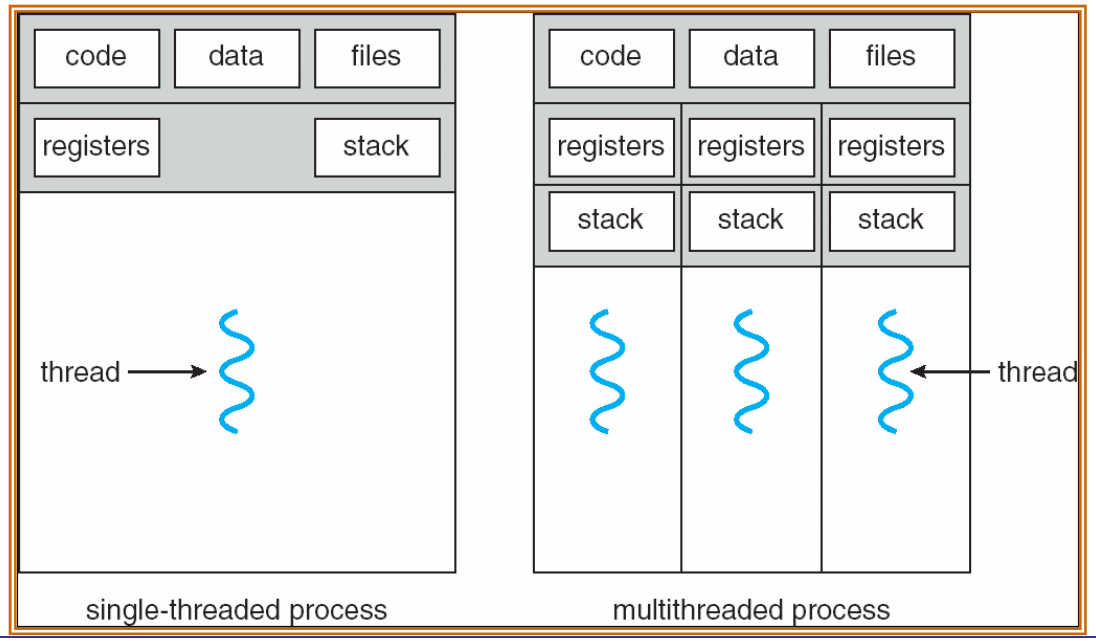
- El sistema operativo es el encargado de administrar las unidades de ejecución que cuente el hardware (procesadores).
- A lo largo de su existencia un proceso estará ejecutando sus instrucciones en un procesador o bloqueado (esperando por algún recurso).



PROCESOS

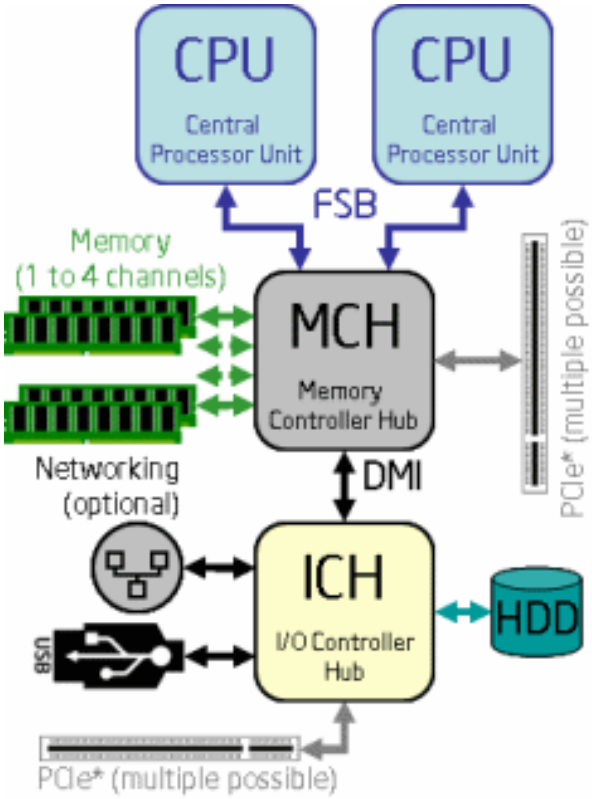
- En los años 90, con el advenimiento de los sistemas multiprocesadores, fueron surgiendo sistemas operativos que permitían a un proceso tener más de un contador de programa (*PC*) (hilo de ejecución).
- A un proceso se le permitía ejecutar en más de un recurso procesador en forma simultánea, logrando paralelismo dentro del proceso.
- Los hilos de un mismo proceso ejecutan sobre un mismo espacio de direccionamiento.

- Con el surgimiento de hardware multi-core de los últimos años y el salto a arquitecturas de 64bits, la utilización de hilos de ejecución en un mismo proceso a reflotado.



ARQUITECTURA

- El hardware de los equipos se componen de unidades de ejecución (procesadores), memoria (RAM) y dispositivos de entrada/salida (discos, red, etc.).
- Con el objetivo de lograr un mejor desempeño, se diponen de una estructura jerárquica de la memoria:
  - Cache de 1er nivel. – Procesador.
  - Cache de 2no. Nivel – Compartida entre cores.
  - Memoria RAM.



- La interconexión de los procesadores a la memoria y los dispositivos de entrada/salida es a través de dos chip: el MCH y el ICH.

- El FSB es el bus de interconexión entre los procesadores y el controlador de memoria.

ARQUITECTURA

- Los equipos de cómputo se componen de dos procesadores Quad-Core con las siguientes especificaciones:

Característica	Valor
Procesador	E5430
Velocidad del reloj	2.66 GHz
Cache de 1er nivel	32 KB + 32 KB
Cache de 2do nivel	2x6MB – 12MB/CPU
FSB	1333 MHz

THREADs

- Una de los paradigmas de programación paralela es la programación con hilos (*threads*).
- Los sistemas operativos modernos brindan una interfaz de programación que permite la utilización de varios hilos de ejecución dentro de un mismo proceso.
- Brindan herramientas básicas para el manejo de hilos: creación, destrucción y sincronización.
- La programación con threads permite tener un mayor control sobre el paralelismo, pero requiere que el paralelismo sea implementado en su totalidad por el programador.
- Por lo tanto, requiere de un mayor conocimiento y dedicación que otras herramientas de programación de alto nivel.

THREADs

- A diferencia de otros paradigmas, la programación con hilos no necesita herramientas de intercambio de información, ya sea a través de mensajes o primitivas a nivel del sistema operativo, ya que cuenta los procesos comparten el espacio de direccionamiento.
- Esto permite una mayor eficiencia frente a la programación con procesos que no comparten el espacio de direccionamiento.
- La gran desventaja del uso exclusivo de esta programación es que no escalan más allá de una máquina. Para esto se deben utilizar otros paradigmas de la programación paralela distribuida.
- Con el uso en conjunto de los dos paradigmas se logra obtener el mayor rendimiento en los clusters HPC de hoy día.

THREADs

- Existen varias bibliotecas de programación con threads. Por lo general, cada sistema operativo brinda una propia.
- Un estándar POSIX es la librería de programación C con hilos pthread.
- Esta librería es implementada por la mayoría de los sistemas UNIX (Linux, Solaris, AIX). Por lo que hace que la programación sea portable.
- Para la creación de threads la librería brinda la primitiva en `pthread_create.`
- Ejemplo:

```
static thread_esclavo(void * args) {…}

pthread_t   thread;

if (pthread_create(&thread,&args,thread_esclavo,&params) < 0)

     error();
```

THREADs

- Para la destrucción de un thread la librería brinda la primitiva en C pthread_destroy.

```
pthread_destroy(NULL);
```

- La librería brinda mecanismos para la sincronización entre los threads:
  - Variable de mutuo exclusión: pthread_mutex_t.
  - Variables de condición: pthread_cond_t.

THREADs

- Las variables de mutuo exclusión permiten a los threads ejecutar secciones críticas en forma mutuoexcluyente (atomicidad).
- Las primitivas utilizadas para la mutuo exclusión son:
  - `pthread_mutex_lock(&mutex).`
  - `pthread_mutex_unlock(&mutex).`
- Las variables de condición permiten bloquear a un proceso mientras una condición no esté dada.
- Las primitivas de brindadas por la librería pthreads son:
  - `pthread_cond_wait(&cond, &mutex)`
  - `pthread_cond_signal(&cond).`

THREADs

- Ejemplo:

```
pthread_mutex_lock(&mutex);
while (contador != 10)
     pthread_cond_wait(&cond,&mutex);
…
pthread_mutex_unlock(&mutex);
```

OpenMP

Que es ?

- Es una API (Application Programming Interface) para aplicar paralelismo sobre memoria compartida utilizando multi-threads.
- Compuesta por:
  - Directivas para el compilador
  - Biblioteca de Rutinas
  - Variables de ambiente
- Portable: C/C++ y Fortran, multiplataforma.
- Se podrían utilizar herramientas de bajo nivel (threads, semaforos, etc).
- Se pierde en eficiencia y flexibilidad ←→ Se gana en portabilidad.

OpenMP

OpenMP utiliza el modelo de ejecución paralela fork-join:

- Un programa comienza su ejecución con un proceso único (thread maestro).
- Cuando se encuentra la primera construcción paralela crea un conjunto de threads.
- El trabajo se reparte entre todos los threads, incluido el maestro.
- Cuando termina la región paralela sólo el thread maestro continua la ejecución.

- Los diferentes threads se comunican a través de variables compartidas.
- Para evitar el acceso simultáneo se utilizan directivas de sincronización.
- Las sincronizaciones son costosas -> hay que tratar de evitarlas.
- Las directivas se aplican a bloques estructurados.

OpenMP

Sintaxis general de las directivas Fortran:

centinela nombre_directiva [cláusulas]

```
 Centinelas:
 C$OMP, *$OMP (en programas de formato fijo)
!$OMP (en programas de formato fijo o variable)
```

Sintaxis general de las directivas C y C++:

#pragma omp nombre_directiva [cláusulas]

OpenMP

Directivas:

- Regiones paralelas
- DO /for paralelos
- Secciones paralelas / Single
- Construcciones de sincronización

Biblioteca:

- `OMP_GET_NUM_THREADS`
- `OMP_GET_THREAD_NUM`
- `OMP_SET_NUM_THREADS`
- `OMP_GET_MAX_THREADS`
- `OMP_INIT_LOCK`

Variables.

OpenMP

Directiva ***PARALLEL:***

Fortran:

```
!$OMP PARALLEL[cláusulas]
 bloque estructurado
!$OMP END PARALLEL
```

C/C++:

```
#pragma omp parallel[cláusulas]
{
 bloque estructurado
}
```

OpenMP

Directiva ***Do/for:***

Fortran:

```
!$OMP DO [clause ...]
do_loop
!$OMP END DO [ NOWAIT ] :
```

C/C++:

```
#pragma omp for [clause ...] newline
for_loop
```

HPF

Extensión de FORTRAN 90 para incluir (más) paralelismo.

FORTRAN 90 ya incluía:

- Sintaxis libre.
- Operaciones vectoriales.
- Allocamiento dinámico.
- Permite definir tipos de datos estructurados.
- Recursión.
- Módulos y conceptos de programación orientada objetos.

HPF

Objetivos del HPF:

- Soportar paralelismo a nivel de datos.
- Soportar optimizaciones de códigos en diferentes arquitecturas.

Conceptos

- Cada procesador ejecuta el mismo programa (SPMD).
- Cada procesador opera sobre un sub-conjunto de los datos.
- HPF permite especificar que datos procesa cada procesador.

HPF

Directivas

- **!HPF$ <directiva>**

- **Si se trabaja con un compilador HPF se interpretan las directivas.**
- **Sino, p. ej. compilador F90, se toman como comentario.**

HPF

Processor - Establece una grilla lógica de procesadores

- !HPF$ PROCESSOR, DIMENSION(4) :: P1
- !HPF$ PROCESSOR, DIMENSION(2,2) :: P2
- !HPF$ PROCESSOR, DIMENSION(2,1,2) :: P3

Distribute - Especifica como distribuir los datos entre las unidades de procesamiento

- !HPF$ DISTRIBUTE (BLOCK) ONTO P1 ::A
- !HPF$ DISTRIBUTE (CYCLIC) ONTO P1 ::B
- !HPF$ DISTRIBUTE (CYCLIC,BLOCK) ONTO P2 ::B

Variantes BLOCK(k) y CYCLIC(k)

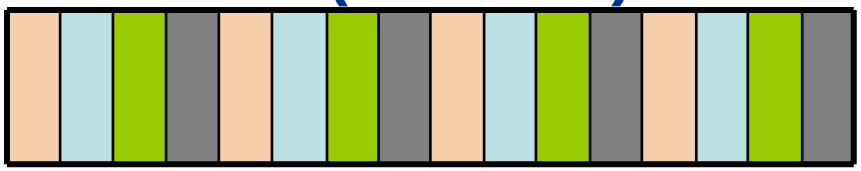
HPF

BLOCK

```
REAL, DIMENSION(16) :: A
!HPF$ PROCESSOR, DIMENSION(4) :: P1
!HPF$ DISTRIBUTE (BLOCK) ONTO P1 ::A
```

CYCLYC

```
REAL, DIMENSION(16) :: A
!HPF$ PROCESSOR, DIMENSION(4) :: P1
!HPF$ DISTRIBUTE (CYCLYC) ONTO P1 ::A
```



HPF

Quien calcula ?

El dueño del lado izquierdo de la asignación !!

```
DO i=1,n
  A(i) = B(i) – C(i)
END DO
```

Reglas

- **Más procesos, más comunicaciones.**
- **Buscar buen balance.**
- **Preservar la localidad de datos.**
- **Usar sintaxis vectorial.**

Ejemplo I

- Trabajos preliminares sobre radiosidad y paralelismo. Eduardo Fernández y Pablo Ezzatti
- Resolución (con pivoteo) de sistemas lineales completos de problemas de radiosidad utilizando OpenMP sobre un equipo de 8 procesadores.

Threads	Tiempo de ejecución (segundos)	Desv. est.
1	81,019	0,008
2	40,792	0,056
4	20,634	0,034
8	10,850	0,168

Tabla 4: Tiempos de ejecución para una matriz de 2048×2048.

- Buena escalabilidad con poco esfuerzo …

Ejemplo II

Caffa3d.MB. Gabriel Usera.

- Simulaciones de fluidos en 3 dimensiones (ecuaciones de Navier-Stokes) mediante la discretización de volúmenes finitos.
- Aplicado al solver lineal (SIP)
- Alcanza speed up de 1,7 en una máquina con doble procesador para 50^3 nodos.

```
c.....OpenMP : Start parallel section
c$OMP   PARALLEL DEFAULT(SHARED)
C
c.....OpenMP : Start a parallel loop
c$OMP   DO PRIVATE(M,K,LKK,I,LKI,IJK,NKMT,NIMT,NJMT,
c$OMP*   KSTT,ISTT,NJT,NIJT)
C
        DO M=1,NBLKS
C
C.....COMPUTE RESIDUAL VECTOR, SUM OF RESIDUALS AND AUXILIARY
C
        NKMT=NKBK(M)-1
        NIMT=NIBK(M)-1
        NJMT=NJBK(M)-1
        KSTT=KBK(M)
        ISTT=IBK(M)
        NJT=NJMT+1
        NIJT=(NIMT+1)*NJT
C
        DO K=2,NKMT
        LKK=LKBK(K+KSTT)
        DO I=2,NIMT
        LKI=LKK+LIBK(I+ISTT)
        JP=LKI+2
        JPL=LKI+NJMT
        RES(JP:JPL)=SU(JP:JPL)-AP(JP:JPL)*FI(JP:JPL)
     *             -AE(JP:JPL)*FI(JP+NJT :JPL+NJT )
     *             -AW(JP:JPL)*FI(JP-NJT :JPL-NJT )
     *             -AN(JP:JPL)*FI(JP+1   :JPL+1   )
     *             -AS(JP:JPL)*FI(JP-1   :JPL-1   )
     *             -AT(JP:JPL)*FI(JP+NIJT:JPL+NIJT)
     *             -AB(JP:JPL)*FI(JP-NIJT:JPL-NIJT)
        END DO
        END DO
C
        END DO
c.....OpenMP : Here ends this parallel loop
```