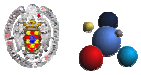
4 de mayo de 2006
E. T. S. Ingenieros Industriales
Universidad de Castilla-La Mancha

Computación de Altas Prestaciones en Ciencia e Ingeniería

Ignacio Martín Llorente
asds.dacya.ucm.es/nacho



Grupo de Arquitectura de Sistemas Distribuidos
Departamento de Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

Laboratorio de Computación Avanzada
Simulación y Aplicaciones Telemáticas
Centro de Astrobiología CSIC/INTA
Asociado al NASA Astrobiology Institute

Objetivo de la Presentación

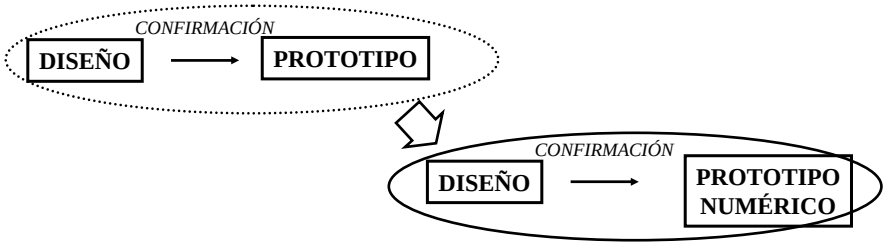
- Proporcionar una visión global de la **aplicación de la computación de altas prestaciones a la resolución de problemas numéricos**

Contenidos

Ingeniería: Disminuye el Uso de Prototipos en el Proceso de Diseño

- Disminuye **costes**
- Aumenta la productividad al disminuir el **tiempo de desarrollo**
- Aumenta la **seguridad**

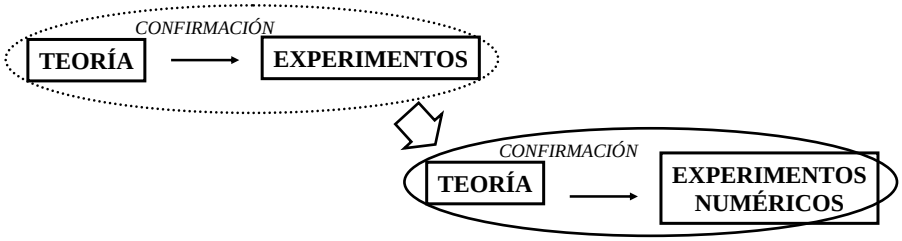
Nuevo Paradigma en Ingeniería



Ciencia: Herramienta Fundamental de Investigación

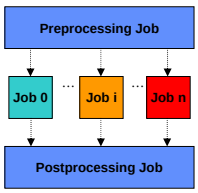
- Simulación de sistemas a **gran escala**
- Simulación de sistemas a **pequeña escala**
- Análisis de la **validez de un modelo matemático**

Nuevo Paradigma en Ciencia

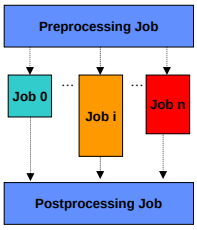


Aplicaciones HTC (High Throughput Computing)

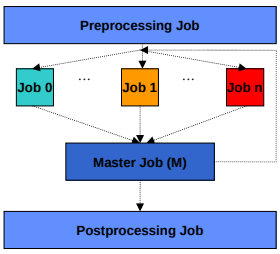
- Su **objetivo** es aumentar el número de ejecuciones por unidad de tiempo
- Su **rendimiento** se mide en número de trabajos ejecutados por segundo
- **Áreas de aplicación**: HEP, bioinformática, finanzas…



HTC Síncrono



HTC Asíncrono



Master-slave

Aplicaciones HPC (High Performance Computing)

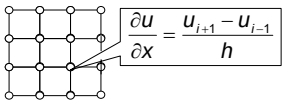
- Su **objetivo** es reducir el tiempo de ejecución de una única aplicación paralela
- Su **rendimiento** se mide en número de operaciones en punto flotante por segundo
- **Áreas de aplicación**:
  - **Estudio de fenómenos a escala microscópica** (*dinámica de partículas*)
    - Resolución limitada por la potencia de cálculo del computador
    - Cuantos más grados de libertad (puntos), mejor reflejo de la realidad
  - **Estudio de fenómenos a escala macroscópica** (*sistemas descritos por ecuaciones diferenciales fundamentales*)
    - Precisión limitada por la potencia de cálculo del computador
    - Cuantos más puntos, más se acerca la solución discreta a la continua

Ecuación no lineal de Schrödinger
Ecuaciones de Maxwell-Bloch

$$a\frac{\partial u(\vec{x})}{\partial x}+b\frac{\partial u(\vec{x})}{\partial y}=0 \quad \forall \vec{x}\in\Omega$$

$$u(\vec{x})= f(\vec{x}) \quad \forall \vec{x}\in\partial\Omega$$

Esquemas numéricos

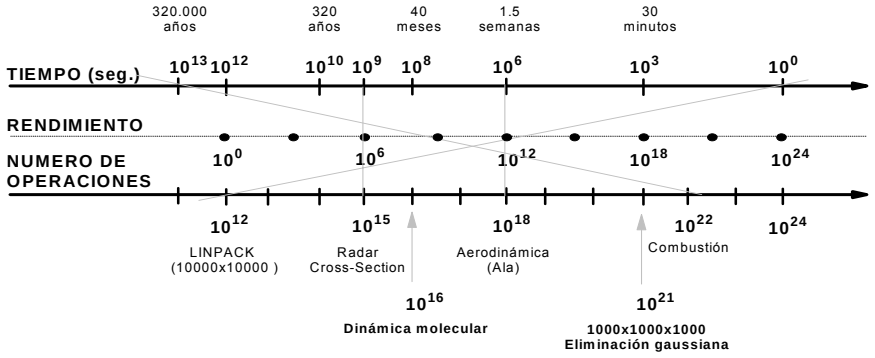


¿Qué Buscamos al Aumentar el Número de Procesadores?

- Resolución de problemas en menor **tiempo de ejecución**, usando mas procesadores (**crítico en T**)
- Resolución de problemas con **mayor precisión**, usando mas memoria (**crítico en P**)
- Resolución de **problemas más reales**, usando modelos matemáticos más complejos (**crítico en C**)

- **Ejemplo:**
  - Ecuación diferencial en derivadas parciales como modelo matemático que describe un sistema

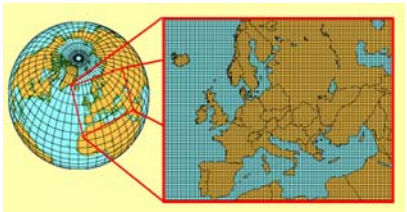
Ejemplos de la Necesidad de Potencia de Cálculo



NUMERO DE OPERACIONES = RENDIMIENTO x TIEMPO

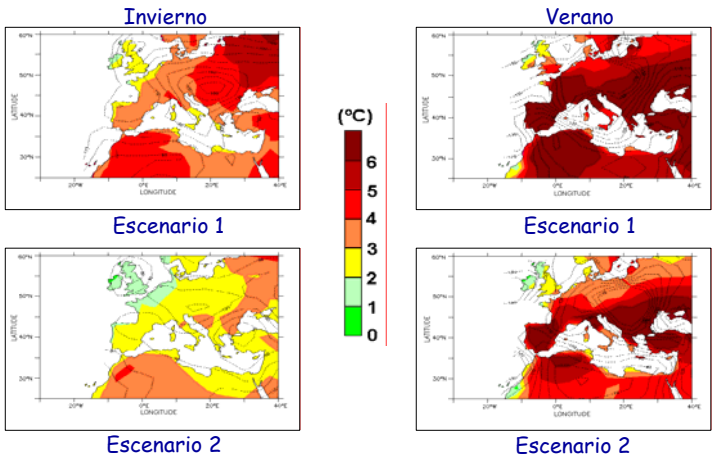
Predicción meteorológica. Necesidades de memoria y cálculo

- El clima es una función de la **longitud , latitud, altura, tiempo**
- Para cada uno de estos puntos se debe calcular **temperatura, presión, humedad y velocidad del viento** (3 componentes)
- Conocido *clima*(i,j,k,t), el simulador debe proporcionar el valor *clima*(i,j,k,t+Δt)
  - **Predicción 1 Minuto**
    - Celdas de 1kmx1km y 10 celdas en altura
      - $5x10^9$ celdas (0.1 TB)
    - Δ t de un minuto: 100 flops por celda (estimación optimista)
      - $100x5x10^9$ operaciones en menos de un minuto: 8 GFlops
  - **Predicción del tiempo a 7dias en 24 horas: 56 Gflops**
  - **Predicción de clima a 50 años en 30 días: 4.8 Tflops**



Predicción meteorológica. Necesidades de memoria y cálculo

• **Cambios en las temperaturas máximas** en el periodo 2070-2100 con respecto al clima actual (1960-1990)



Límites de la Ley de Moore

"La densidad de circuitos en un chip se dobla cada 18 meses".
(Dr. Gordon Moore, Chairman, Intel, 1965)

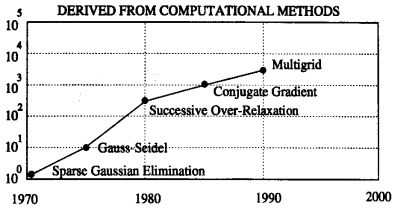
Velocidad de la luz
- Computador con 1Tflop y 1 TB (1 THz)
- La distancia *r* para recoger el dato de memoria $< c/10^{12} = 0.3$ mm: **Tamaño del computador 0.3x0.3**

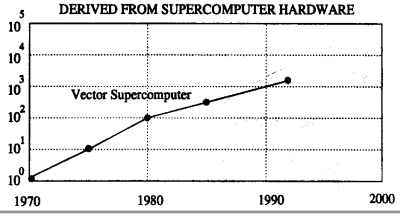
1 palabra de memoria ocupa 3 amstrons x 3 amstrons = tamaño de 1 átomo

Análisis Previo a la Paralelización

- **Complejidad numérica** del código/algoritmo: Escalabilidad
- **Grado de paralelismo** del código/algoritmo: Ley de Amdahl
- **Granularidad** de la implementación paralela: Grado de acoplamiento requerido en la arquitectura subyacente
- **Análisis del código**: ¿Por qué?, ¿Dónde? Y ¿Cómo?

Ganancia Debida al Algoritmo y al Computador





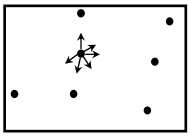
Grand Challenge: High Performance Computing and Communications (NSF)

Ejemplo de la Importancia de la Complejidad

Problema:

•Conjunto de *N* moléculas sometidas a la ley de Newton,
El potencial es la suma de potenciales de corto y largo alcance

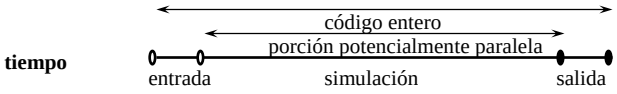
P	$O(N^2)$ MOLMEC 7.000	$O(NlogN)$ MEGADYN 550.000
1	8152	
2	4481	6305
3	3956	
4	2427	3295
6	1769	
8		1849



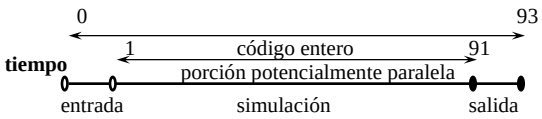
- Ambos tienen speedups lineales
- La simulación de 550.000 partículas consumiría más de 18.000 procesadores con MOLMEC

¿Vamos a Ganar Algo Paralelizando la Aplicación?

- Se debe medir el tiempo de ejecución (*wall-time*) de la **parte del programa que se va a paralelizar** (*código potencialmente paralelo*) usando las rutinas de tiempos que nos aporte el sistema operativo:
  - Se quitan las partes secuenciales, como las sumas globales y entrada/salida de datos
- Se debe medir el **tiempo del programa entero**, desde la primera sentencia hasta la última



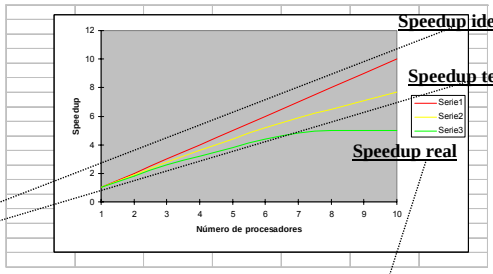
- El **contenido paralelo *c*** se calcula: $\frac{Tiempo\_parte\_paralela}{Tiempo\_total} = \frac{90}{93} = 0.9667$



Ganancia

$$speedup\_teorico = \frac{1}{(1-c)+\frac{c}{p}} = \frac{1}{0.0323+\frac{0.9677}{p}}$$

Número de CPUs	Speedup teórico
1	1
2	1,9
3	2,8
4	3,6
5	4,4
6	5,2
7	5,9
8	6,5
9	7,1
10	7,5



La diferencia creciente entre el speedup ideal y el teórico es debida al tanto por ciento serie que cada vez se hace más dominante

- Tiempo consumido en creación, sincronización y comunicación
- Competencia por recursos
- Desigualdad en la carga de los procesadores

Ley de Amdahl (1967)

*La eficiencia obtenida en una implementación paralela viene **limitada por la parte secuencial**, por ello el límite superior de mejora obtenida es independiente del número de procesadores*

$$Eficiencia: E(N,p) = \frac{S(N,P)}{P} = \frac{T(N,1)}{T(N,P)P} = \frac{T(N,1)}{T(N,1)+h(N,P)P}$$

Ejemplo:

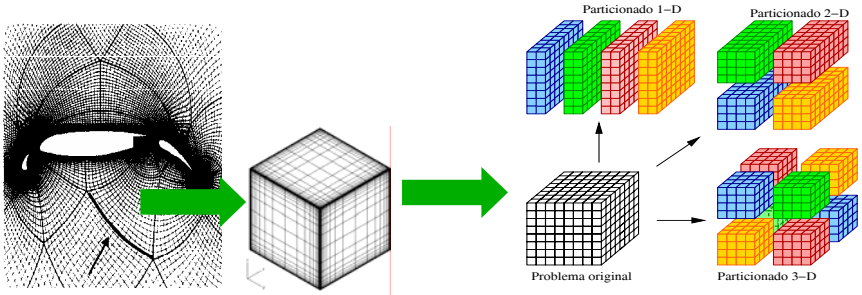
- Nunca se conseguirá un speedup 10 en un código 20% secuencial

Respuesta de Gustafson (1988)

*En la práctica, la mayoría de las aplicaciones son **críticas en precisión**, y por lo tanto el tamaño de problema crece con el número de procesadores*

Relación entre Computación y Comunicación

- El tamaño de **grano o granularidad** es una medida de la cantidad de computación de un proceso software
  - Se considera como el segmento de código escogido para su procesamiento paralelo
- Define la **arquitectura óptima**



Objetivo: Reducir el tiempo de ejecución de una aplicación

¿**Por qué** no se ejecuta eficientemente?

¿**Donde** se encuentran los puntos críticos? → Regla 80/20

¿Cómo?

CPU

optimización

paralelización

2. Antes de Desarrollar un Código Paralelo
- Análisis del Código

¿Por Qué?

¿**Por qué** el código no se ejecuta eficientemente?

- Procesador
- Entrada/salida
- Memoria

time
conocimiento del código

Tiempos de sistema altos pueden ser debidos a:
- Entrada/salida
- Fallos de página en el acceso a memoria virtual
- Llamadas al sistema

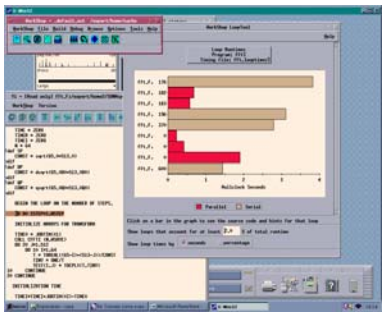
2. Antes de Desarrollar un Código Paralelo
- Análisis del Código

¿Dónde?

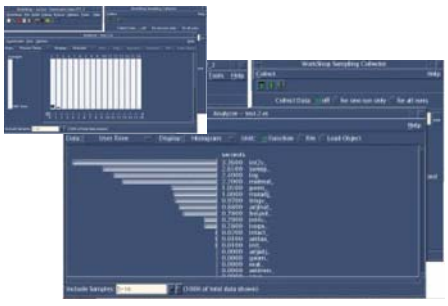
Herramientas de *profiling* orientadas a terminal

`gprof`, `tconv`, `dtime`, `etime`, ...

Herramientas de *profiling* orientadas a entorno de ventanas



Looptool (solaris)



cvd (SGI)

2. Antes de Desarrollar un Código Paralelo
• Análisis del Código

¿Cómo?

Procesador:

- Opciones de compilación para optimización secuencial
- Técnicas de optimización para mejorar la explotación de la arquitectura superescalar
- Técnicas de optimización para mejorar la explotación de la memoria cache
- Opciones de compilación para paralelización automática
- Inclusión de directivas de paralelización en el código

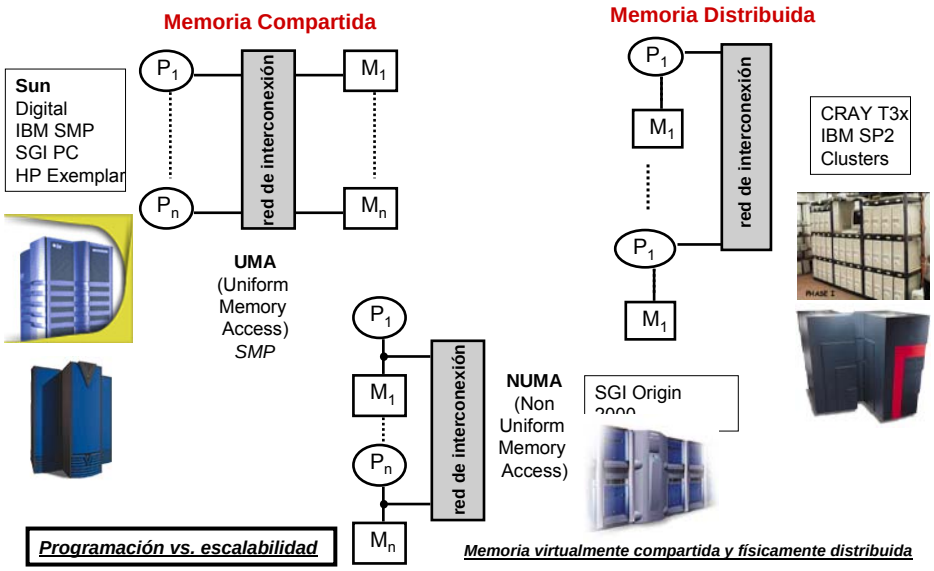
Entrada/salida:

- Reorganizar E/S para evitar muchas peticiones cortas (convertir a pocas y largas)
- Funciones `mmap` para mapear ficheros en memoria
- Funciones `madvise` para indicar al sistema el uso de las páginas de fichero en memoria

Memoria virtual:

- Técnicas de optimización para mejorar la explotación de la memoria virtual
- Funciones `madvise` para indicar al sistema el uso de las páginas de fichero en memoria

3. Arquitecturas de Altas Prestaciones
• Arquitecturas Multiprocesador



TOP500 LIST FOR NOVEMBER 2005

R_{max} and R_{peak} values are in GFlops. For more details about other fields, please click on the button "Explanation of the Fields"

EXPLANATION OF THE FIELDS

1-100 | 101-200 | 201-300 | 301-400 | 401-500

Rank	Site	Computer	Processors	Year	R_{max}	R_{peak}
1	DOE/NNSA/LLNL United States	BlueGene/L - eServer Blue Gene Solution IBM	131072	2005	280600	367000
2	IBM Thomas J. Watson Research Center United States	BGW - eServer Blue Gene Solution IBM	40960	2005	91290	114688
3	DOE/NNSA/LLNL United States	ASC Purple - eServer pSeries p5 575 1.9 GHz IBM	10240	2005	63390	77824
4	NASA/Ames Research Center/NAS United States	Columbia - SGI Altix 1.5 GHz, Voltaire Infiniband SGI	10160	2004	51870	60960
5	Sandia National Laboratories United States	Thunderbird - PowerEdge 1850, 3.6 GHz, Infiniband Dell	8000	2005	38270	64512
6	Sandia National Laboratories United States	Red Storm Cray XT3, 2.0 GHz Cray Inc.	10880	2005	36190	43520
7	The Earth Simulator Center Japan	Earth-Simulator NEC	5120	2002	35860	40960
8	Barcelona Supercomputer Center Spain	MareNostrum - JS20 Cluster, PPC 970, 2.2 GHz, Myrinet IBM	4800	2005	27910	42144

Servidores HPC (*High Performance Computing Servers*)

• Arquitecturas de memoria compartida (SMP) o distribuida (MPP)

Perfil de Aplicación

• Ejecución eficiente de aplicaciones HPC y HTC

Ventajas

• Interconexión con ancho de banda alto y latencia baja
• Acceso uniforme al sistema gracias a una única copia del sistema operativo

Inconvenientes

• Baja escalabilidad (para SMPs)
• Modelos complejos de programación (para HPC en MPPs)
• Precio alto

Sistema de Colas Batch

NQE

Clusters Dedicados

- Cluster dedicado y homogéneo de PCs o estaciones interconectados por medio de una red de área de sistema (Giganet, Myrinet…)

Perfil de Aplicación

- Ejecución eficiente de aplicaciones HTC y HPC de grano grueso

Ventajas

- Mejor relación coste/rendimiento para aplicaciones HTC
- Mayor escalabilidad

Inconvenientes

- Requieren modelos de programación de memoria distribuida (librerías de paso de mensajes como MPI) para aplicaciones HPC

Sistema de Gestión de Recursos

PBS

Clusters no Dedicados

- Cluster no dedicado y heterogéneo de PCs o estaciones interconectados por medio de una red de área local (Fast ethernet…)

Perfil de Aplicación

- Únicamente ejecuta aplicaciones HTC

Ventajas

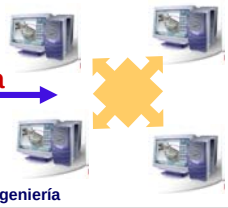
- Mínima relación coste/rendimiento para aplicaciones HTC
- Mayor escalabilidad

Inconvenientes

- Interconexión con ancho de banda bajo y latencia alta
- Requiere capacidades de gestión adaptativa para usar los tiempos ociosos de los recursos dinámicos

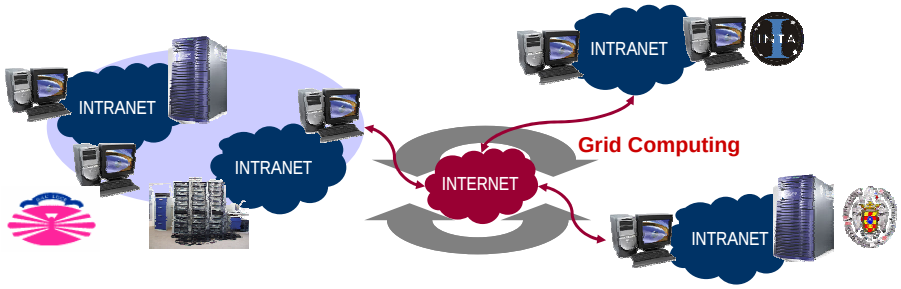
Sistema de Gestión de Carga

Condor



La Tecnología Grid es Complementaria a las Anteriores

- Interconecta recursos en **diferentes dominios de administración** respetando sus políticas internas de seguridad y su software de gestión de recursos en la Intranet
- Una nueva tecnología dentro del área global de Computación de Altas Prestaciones, para satisfacer las demandas de **determinados perfiles de aplicación**

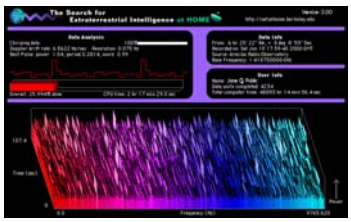


Iniciativas de Computación en Internet

- Plataformas software para usar **recursos cedidos voluntariamente**
- **Tecnología complementaria** a las anteriores que permiten **interconectar recursos individuales** en lugar de servidores y clusters con diferentes DRMs

Objetivo: Análisis de datos de telescopio (Arecibo, Puerto Rico) en búsqueda de señales

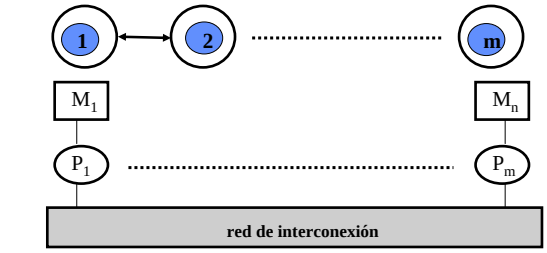
`setiathome.ssl.berkeley.edu`



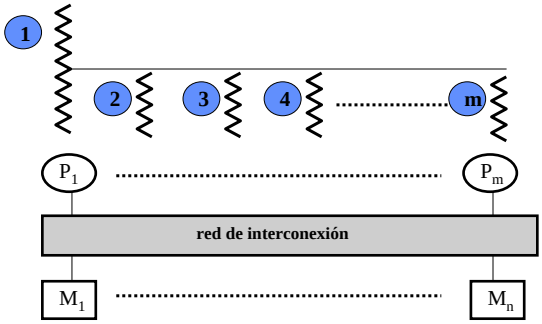
Estadísticas Octubre de 2005
Users: 234.746; Hosts: 498.876; Countries: 216
Average floating point operations per second: 127,491 TeraFLOPS

4. Paradigmas de Programación
• Visiones del Sistema

Memoria Distribuida



Memoria Compartida



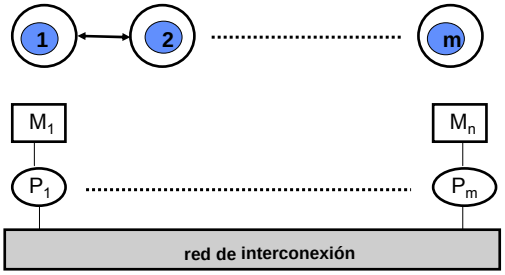
4. Paradigmas de Programación
• Modelos de Programación

PARADIGMAS (visión del sistema por parte del usuario)	MODELOS DE PROGRAMACIÓN (tipos de lenguaje)	ARQUITECTURA (computador donde aparece)
MC	• **PARALELISMO EN CONTROL** CON DIRECTIVAS • MEMORIA COMPARTIDA ESTÁNDAR NIVEL DE ABSTRACCIÓN / FLEXIBILIDAD / PORTABILIDAD	UMA y NUMA
MD	• **PARALELISMO EN DATOS** CON DIRECTIVAS • **PASO DE MENSAJES** ESTÁNDAR NIVEL DE ABSTRACCIÓN / FLEXIBILIDAD / PORTABILIDAD	UMA, NUMA y MD

Descripción

- Comunicación por paso de mensajes
- Primitivas de gestión de comunicaciones
  - Envío
  - Recepción
  - Sincronización

MPI
PVM
Librerías nativas



Nivel superior:
(directivas de distribución de datos)
HPF

MPI (*Message Passing Interface*)

Estructura de un Programa (Solo 6 rutinas)

```
#include <stdio.h>
#include <mpi.h>

main(int argc, char **argv)
  {
    int mi_rango, numero_procesos, etiqueta=50, destino=0, origen;
    char mens[100];
    MPI_Status est;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &mi_rango);
    MPI_Comm_size(MPI_COMM_WORLD, &numero_procesos);

    if (mi_rango !=0) {
      sprintf(mensaje,"¡Saludos desde el proceso %d!", mi_rango);
      MPI_Send(mens, strlen(mensaje)+1, MPI_CHAR, destino, etiqueta, MPI_COMM_WORLD);
    } else {
      for (origen = 1; origen < numero_procesos; origen++) {
        MPI_Recv(mens, strlen(mensaje)+1, MPI_CHAR, origen, etiqueta, MPI_COMM_WORLD, &est);
        printf("%s\n", mens);
      }
    }
    MPI_Finalize();
  }
```

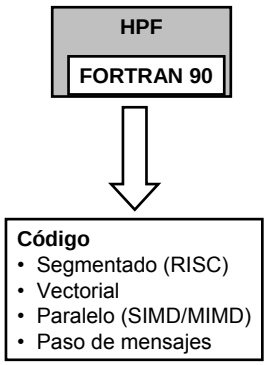
Descripción

HPF = FORTRAN 90 +
- Directivas para distribución de datos y mapeo
- Construcciones paralelas
- Nuevas funciones intrínsecas

Características
- La comunicación y distribución de datos la realiza el compilador
- Único flujo de control
- Memoria global
- Paralelismo implícito en las operaciones
- Uso de directivas de compilación

Ventajas
- Simple de escribir y depurar
- Portable, tanto los viejos como los nuevos códigos

Inconvenientes
- Solo válido para paralelismo en datos
- Solo tiene control en la distribución de los datos
- Difícil obtener muy buenos rendimientos
- Confía en el compilador



HPF (*High Performance Fortran*)

Estructura de un Programa (Descomposición de datos)

```
REAL*8                  X(N,N), Y(N,N)
!HPF$ PROCESSORS        P(4)

!HPF$ ALIGN             X(i,j) with P(i)
!HPF$ ALIGN             Y(i,j) with P(j)

!HPF$ DISTRIBUTE        P(BLOCK)

FORALL (I=1:N,J=1:N)
       X(I,J) = Y(J,I) + I*J
```

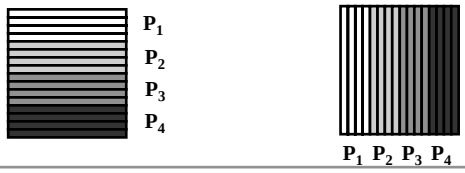
DECLARACIÓN DE LAS VARIABLES Y DE LA MALLA LÓGICA DE PROCESADORES

ALINEAMIENTO DE LAS MATRICES SOBRE LA MALLA DE PROCESADORES

DISTRIBUCIÓN DE LAS MATRICES SOBRE LA MALLA DE PROCESADORES

CONSTRUCCIÓN PARALELA

```
!HPF$ DISTRIBUTE      P(BLOCK)
```

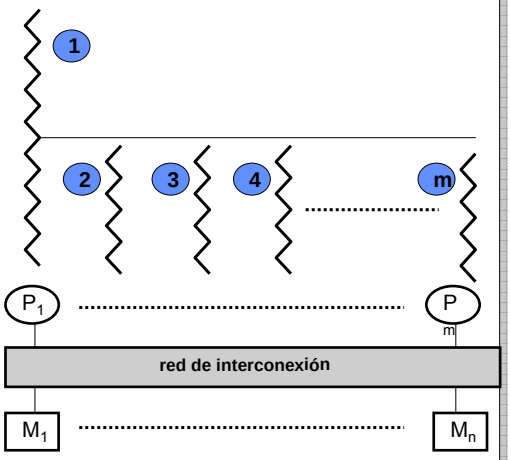


Descripción

- Comunicación memoria compartida
- Primitivas de gestión de threads
  - •Creación
  - •Espera
  - •Protección de datos

Posix Threads
Librerías nativas

Nivel superior:
(directivas de distribución de datos)
DOACROSS
DOALL
X3H5
OPENMP



¿Qué es un Thread?

- Un sistema multiprocesador donde todos los procesos pueden ejecutar el núcleo del sistema operativo con el mismo privilegio se denomina **multiprocesador simétrico** (SMP)
- Todos los recursos se **comparten los recursos** (memoria y dispositivos entrada/salida)
- Un sistema operativo soporta multiprocesamiento simétrico siempre que se puedan proteger **regiones críticas** (garantía de **exclusión mutua**)
- El **planificador** distribuye los procesadores disponibles entre los procesos (*threads*) preparados para ejecutarse
- El paralelismo aparece cuando un programa ha sido codificado (o compilado) de modo que fragmentos del mismo se ejecutan como procesos (*threads*) independientes.

***Thread* = proceso ligero (solo contexto hardware)**

Posix *Threads*

```
void *print_message_function( void *ptr );
pthread_mutex_t mutex;
main()
{
     pthread_t thread1, thread2;
     pthread_attr_t pthread_attr_default;
     pthread_mutexattr_t pthread_mutexattr_defa
     struct timespec delay;
     char *message1 = "Hello";
     char *message2 = "World\n";

     delay.tv_sec = 10;
     delay.tv_nsec = 0;

     pthread_attr_init(&pthread_attr_default);
     pthread_mutexattr_init(&pthread_mutexattr_default);

     pthread_mutex_init(&mutex, &pthread_mutexattr_default);
     pthread_mutex_lock(&mutex);

     pthread_create( &thread1, &pthread_attr_default,
                     (void *) print_message_function, (void *) message1);
     pthread_mutex_lock(&mutex);
     pthread_create(&thread2, &pthread_attr_default,
                     (void *) print_message_function, (void *) message2);
     pthread_mutex_lock(&mutex);
     exit(0);
}
```

```
void *print_message_function( void *ptr )
{
     char *message;
     message = (char *) ptr;
     printf("%s ", message);
     pthread_mutex_unlock(&mutex);
     pthread_exit(0);
}
```

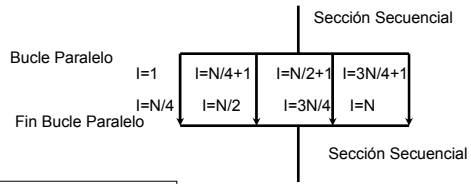

Descripción

Problemas de la programación por medio de threads
- •Análisis del código
- •Recodificación
- •Incluir primitivas *multithreading*

No cambiamos el lenguaje pero ...

Paralelización por medio de directivas
- •Ejecución de un **bucle** sobre múltiples procesos

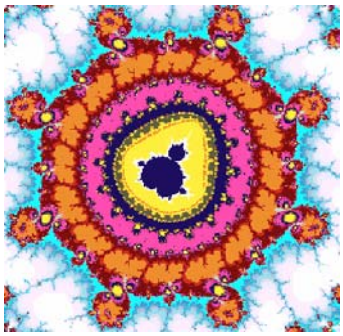
```
C$PAR DOALL
      DO I=1, N
        A(I) = B(I)
      END DO
```



Nivel de abstracción superior:
- •Menos flexible
- •Más portable
- •Menos eficiente
- •Más rápido

Planificación de Iteraciones sobre *Threads*

```
for m = 0 … 1023 do
  for n = 0 … 1023 do
    c = x+yj
    a = 0+0j
    while ((k<1000) and (abs(a)<2)) do
      a = a^2+c
      k = k+1
    end
    data(n,m) = k
  end
end
```



Paralelizador automático (planificación por bloques)

p	1	2	3	4
T(Ultra450)	41	30	35	27

Paralelizador manual (planificación cíclica)

p	1	2	3	4
T(Ultra450)	41	21	14	10

Paralelización Automática

```
PROGRAM prac10
      REAL A(1000),x
      x = 135.0
      DO I=1, 1000
        x = I*I+1
        A(I)= x*x
      END DO
      SUM = 0.0
      DO I=1, 1000
        SUM = SUM + A(I)
      END DO
      WRITE(*,*) SUM, x, I
      END
```

.l .m

```
4: PARALLEL (Auto) __mpdo_MAIN__1

9: Not Parallel
   Scalar dependence on SUM.
```

```
CSGI$ start 1
      PROGRAM MAIN
      IMPLICIT NONE
C      **** Variables and functions ****
      REAL*4 A(1000_8)
      REAL*4 x
      INTEGER*4 I
      REAL*4 SUM
C      **** statements ****
      x = 1.35E+02
C      PARALLEL DO will be converted to SUBROUTINE __mpdo_MAIN__1
CSGI$ start 2
C$OMP PARALLEL DO private(I), lastprivate(x), shared(A)
      DO I = 1, 1000, 1
        x = REAL(((I * I) + 1))
        A(I) = (x * x)
      END DO
CSGI$ end 2
      SUM = 0.0
CSGI$ start 3
      DO I = 1, 1000, 1
        SUM = (A(I) + SUM)
      END DO
CSGI$ end 3
      I = 1001
      WRITE(*, *) SUM, x, I
      STOP
      END ! MAIN
CSGI$ end 1
```

Directivas *OpenMP*

Problemas de Otros Modelos de Programación

Paso de Mensajes (MPI):
- Difícil de programar
- No soporta paralelización incremental de un código (partiendo del puro secuencial)
- Creado para las arquitecturas de memoria distribuida
- Los actuales sistemas cc-NUMA no tienen porqué arrastrar este tipo de programación

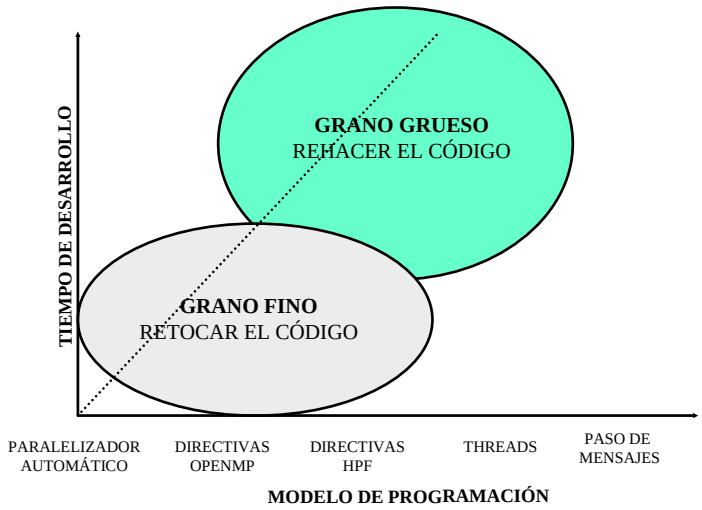
HPF:
- No es tan utilizado como inicialmente se estimó
- Tiene bastantes limitaciones (paralelismo en control)
- Los compiladores son todavía deficientes

Posix Threads:
- Complejos
- No son utilizados por la computación numérica
- Prácticamente sin soporte para Fortran 77
- Incluso en lenguaje C exige programar a demasiado bajo nivel

Directivas clásicas:
- Solo soportan paralelismo a nivel de bucle (solo grano fino)

OCCAM, BSP, …:
- No son portables

La Solución Depende de (Tiempo, Dinero, Rendimiento)



Resumen

1. Necesidades de Recursos en Simulación Numérica
2. Antes de Desarrollar un Código Paralelo
3. Arquitecturas de Altas Prestaciones
4. Paradigmas de Programación
5. Paradigma de Memoria Distribuida
6. Paradigma de Memoria Compartida
7. Conclusiones