Modelos de Programación y herramientas

Jesús Labarta

CEPBA-UPC

Technology Transfer | Research | Training | Mobility of Researchers
User Support | Education | HPC Facilities | Parallel Expertise

Supercomputadores

- **Qué son**
  - Máquinas mas rápidas del momento
  - http://www.top500.org
  - Necesidad?
    - ✓ Capacity computing
    - ✓ Capability computing
- **Cómo son**
  - Multiprocesadores
- **Cómo se programan**
  - Modelos de programación
  - Herramientas de análisis del rendimiento

Supercomputadores

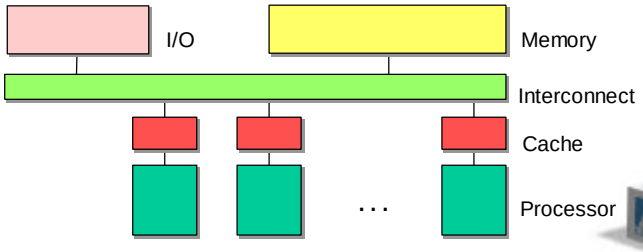
- **Temas**
  - Modelos de programación
    - ✓ OpenMP
    - ✓ MPI
  - Herramientas
    - ✓ Paraver
    - ✓ Dimemas
- **Previo**
  - Arquitectura

Architectures

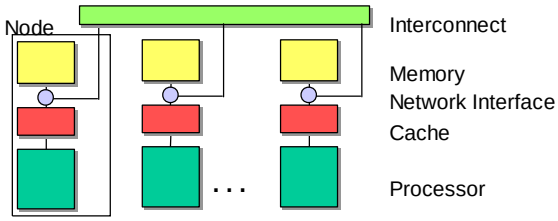
- **Shared Memory**
  - SMP
  - NUMA
- **Distributed Memory**
- **Hierarchical mix**
  - SMP nodes @ Distributed memory

Shared Memory

- **SMP (symmetric multiprocessor):**
  - UMA
  - Issues
    - ✓ Interconnection network:
      - – Aggregated bandwidth: bus, crossbar.
      - – Latency
    - ✓ Cache: coherency, consistency

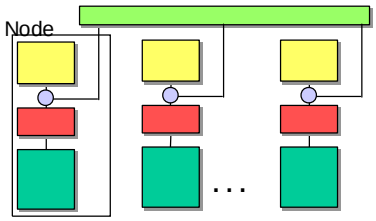


Distributed Shared Memory



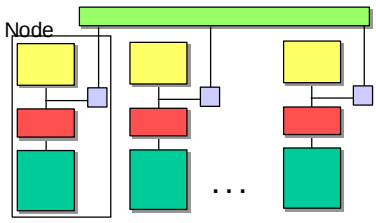
- **Load/stores: local/remote depending on physical address**
- **NUMA (non-uniform memory access): remote accesses may take 5-10 times longer**

Distributed Shared Memory



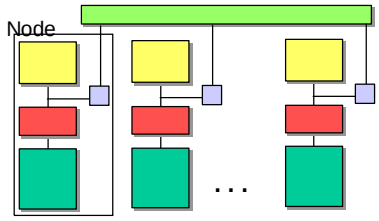
- **Issues:**
  - Interconnection network: latency, topology, bisection bandwidth
  - Cache: coherency, consistency !!
  - Locality of access

Distributed Memory



- **Load/stores: local**
- **I/O: send/receive (local address + node addressing mechanism)**
- **Private / commodity networks**

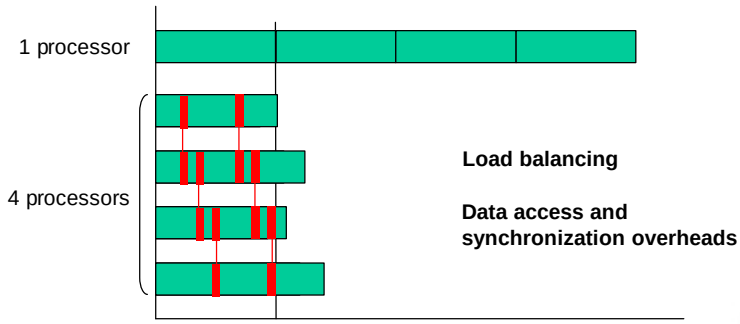
Distributed Memory



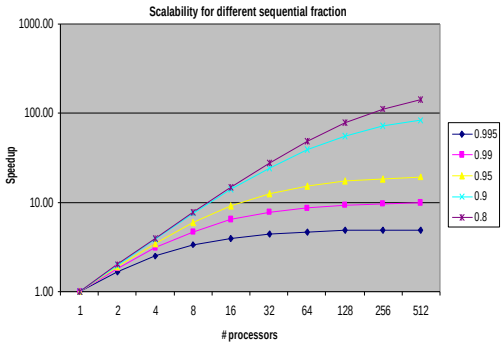
- **Issues**
  - Interconnection network:
    - ✓ latency, topology, bisection bandwidth
    - ✓ Injection mechanism
- **SMP nodes**

What is parallelization?

- **Objective: use of multiple processors to perform two or more tasks at the same time**
- **Benefit: faster turnaround time**

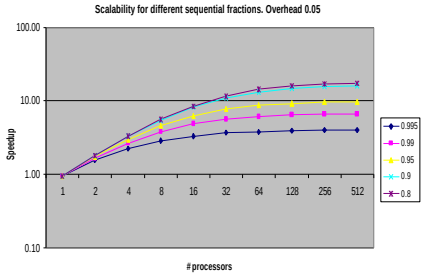


Speed-up: Amdahl's Law

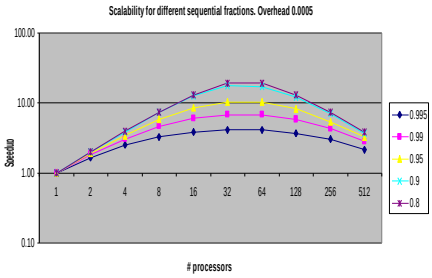


$$S(p) = 1 / (1 - f + (f / p))$$

Speed-up: Overheads



$$S(p) = 1 / (1 - f + (f / p)+o)$$



$$S(p) = 1 / (1 - f + (f / p)+o*p)$$

Modelos de programación

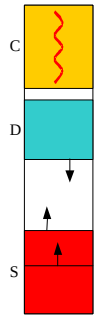
- **Mecanismos disponibles al programador para expresar la estructura lógica de un programa**

- **Influye**
  - Complejidad del programa
    - ✓ Costo de desarrollo
    - ✓ Legibilidad. Costo de mantenimiento
  - Rendimiento
    - ✓ Influenciado por el modelo
    - ✓ por la implementación del modelo
    - ✓ Por la estructura de paralelización

Modelos de programación

- **Componentes**
  - Datos
  - Procesos
  - Comunicación
  - Sincronización
  - Entrada/salida

Secuencial



Variable globales:

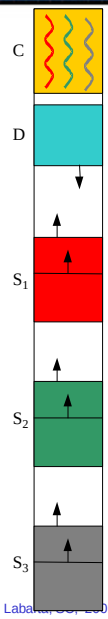
Variables dinámicas(locales):
- **Visibilidad local**
- **Se puede pasar puntero al llamar funciones**
- **Vida limitada**

Librerias:

malloc:
- **Visible global**

Problemas:
- **Desbordamiento pilas**
- **Errores difíciles detectar (efecto variable y lejano)**

Memoria compartida: espacio @



Variable globales:
- **Unica copia**

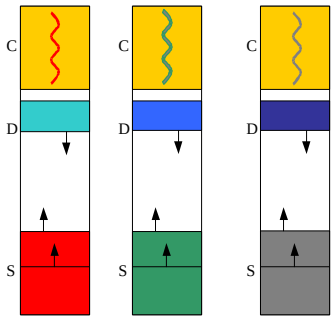
Variables dinámicas(locales):
- **Acceso normal: privadas**
- **Accesibles globalmente (pasar puntero)**

Librerias:
- **Exclusión mutua**

malloc:
- **Visible global**

Problemas:
- **Desbordamiento pilas**
- **Errores difíciles detectar (efecto variable y lejano)**

Memoria Distribuida: espacio @



Variables globales:
Replicadas: misma dirección lógica
mantener consistencia
Distribuidas: misma dirección, distinto objeto

Variables dinámicas(locales):
privadas (no posibilidad compartir)
~ mismas direcciones lógicas

Librerías:
Locales.

malloc: local
~ mismas direcciones lógicas

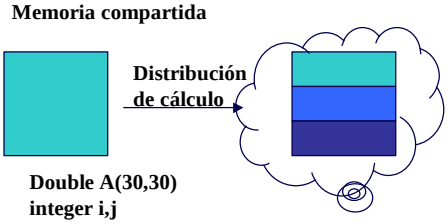
Modelos

- **Memoria compartida**
  - Pthreads
  - OpenMP
    - ✓ http://www.openmp.org
    - ✓ http://www.compunity.org
- **Memoria distribuida**
  - MPI
    - ✓ http://www.mpi.org
    - ✓ http://www-unix.mcs.anl.gov/mpi/

- Aspectos básicos de programación en los distintos modelos

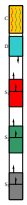
Pthreads: programación

Memoria compartida




```
Double A(30,30)
integer i,j

Do j = 1,30
  Do i =1,30
    A(i,j) = 2* A(i,j)
    A(i,j) = j* A(i,j)
    A(i,j) = i* A(i,j)
```



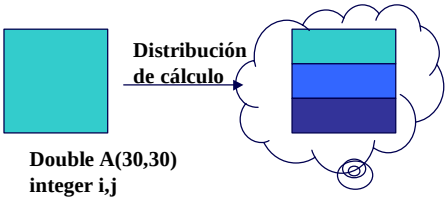
```
Double A(30,30)
interger thid, j

main() {
  Do j = 1,30
    Do thid=1,num_threads
      iinf = f(thid)
      isup = f(thid)
      create_thread(Loops_body,iinf,isup)
}

Loop_body(iinf,isup) {
  Do i =iinf,isup
    A(i,j) = 2* A(i,j)
    A(i,j) = j* A(i,j)
    A(i,j) = i* A(i,j)
}
```

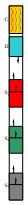

OpenMP: programación

Secuencial




```
Double A(30,30)
integer i,j

Do j = 1,30
  Do i =1,30
    A(i,j) = 2* A(i,j)
    A(i,j) = j* A(i,j)
    A(i,j) = i* A(i,j)
```



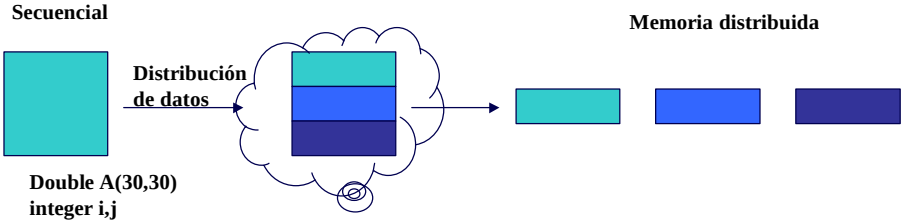
```
Double A(30,30)
interger i,j

  Do j = 1,30
C$OMP  PARALLEL DO
C$OMP  PRIVATE (i)
   Do i =1,30
      A(i,j) = 2* A(i,j)
      A(i,j) = j* A(i,j)
      A(i,j) = i* A(i,j)
```

MPI: programación

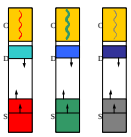
Secuencial

Memoria distribuida




```
Double A(30,30)
integer i,j

Do j = 1,30
  Do i =1,30
    A(i,j) = 2* A(i,j)
    A(i,j) = j* A(i,j)
    A(i,j) = i* A(i,j)
```



```
Double A(10,30)

myrank = quien_soy()
Do j = 1,30
  Do li =1,10
    i=li+myrank*10
    A(li,j) = 2* A(li,j)
    A(li,j) = j* A(li,j)
    A(li,j) = i* A(li,j)
```