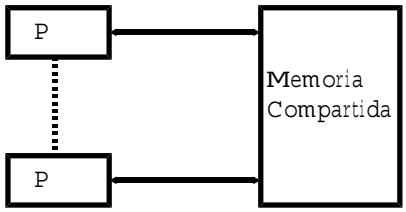
4 Parallel Architectures

Clasificación de Flynn: SISD - MISD - SIMD - MIMD

- ¿Pipeline? ¿Superescalares? ¿Arquitecturas Harvard?
- ¿Por qué se considera que MIMD es la más general? Aplicable a una amplia gama de problemas (al menos más amplia) que las demás arquitecturas
- ¿Por qué se considera que MIMD es la más escalable? Escalabilidad: capacidad de aumentar la cantidad de recursos para resolver problemas mayores (en datos y/o en procesamiento) "Más escalable": no necesita sincronismo al nivel de clock

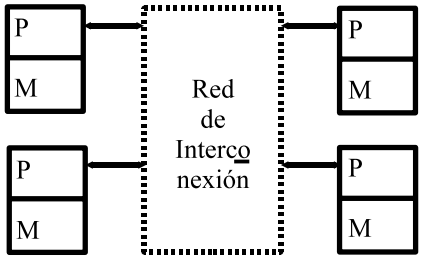
Dos clases de MIMD

- Memoria compartida (multiprocesadores)



  - Máquina única-muchos procesadores
  - Una única visión de la memoria (mapa de memoria)

- Memoria distribuida (multicomputadoras)



  - DMPC: lo mismo
  - Multicomputadora: muchas computadoras
  - Losely coupled: independencia, asincronismo
  - Pasaje de mensajes: CSP

MIMD - Multiprocesadores

1. Con toda la memoria compartida y en un solo bloque
   - Ventajas: Sincronización por acceso a memoria, Comunicación de procesos por acceso a memoria, "Historia" de la concurrencia: (Conc. - par.).
   - Desventajas: t de acceso a memoria (ya es un problema con 1 procesador), Accesos simultáneos a memoria: Memoria en bancos, buses.
   - Tiempo de acceso a memoria: $t_{mem} + t_{col(frec, \#proc)}$.
2. Agregando Cache a los procesadores
   - Reducir los requerimientos de acceso a memoria: frecuencia de accesos a la memoria compartida.
   - Problema: Falta de coherencia de memorias cache $\Longrightarrow$ Hardware (tiempo y transparencia): protocolos de coherencia de cache (snoop o aviso).
   - Tiempo de acceso: $t_{cache} + t_{mem} + t_{col(frec(cache), \#proc)}$.
   - SMP: caso particular, énfasis en el acceso a todos los recursos.
3. Agregando memoria local independiente de memoria compartida ("local data")
   - Ventaja: independencia de accesos a la memoria local.
   - Desventajas: Hardware-"discriminación" de accesos a memoria.
   - Tiempo de acceso: $t_{cache} + t_{meml} + t_{mem} + t_{col(frec(cache,ml), \#proc)}$.
4. Memoria compartida físicamente distribuida
   - SGI Origin.
   - La visión de la memoria sigue siendo única.
   - Sigue habiendo problemas con coherencia de caches.
   - Tiempo de acceso: $t_{cache} + t_{meml} + t_{memr} + t_{col(frec(ml,mr), \#proc)}$.

Las últimas dos son NUMA por construcción, por su misma arquitectura.

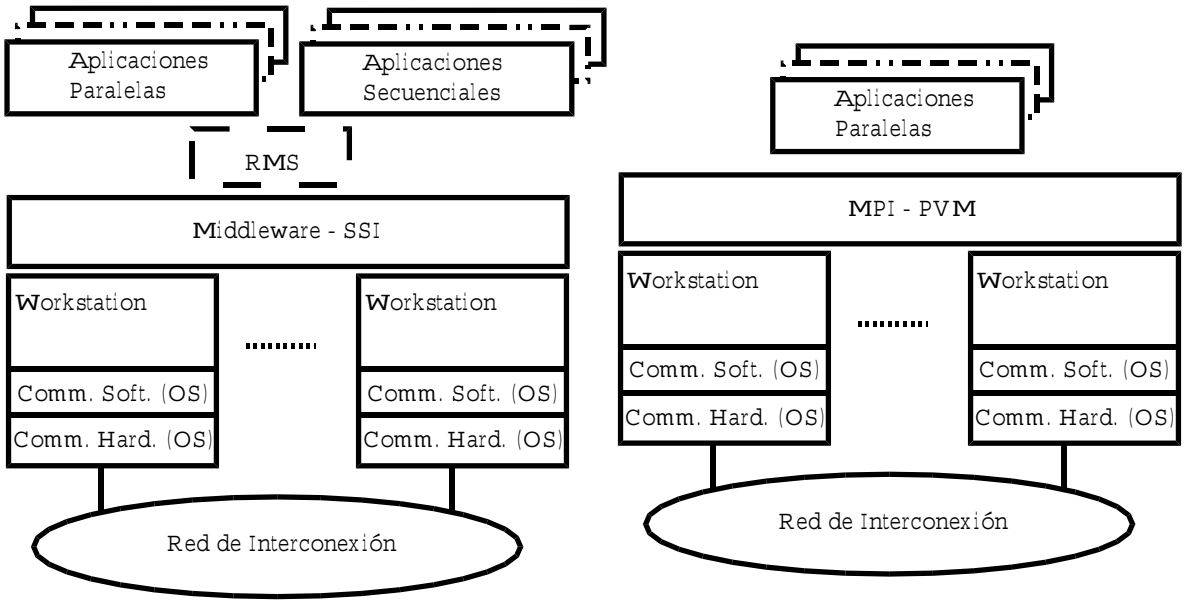
MIMD - Multicomputadoras

- La red de interconexión comunica computadoras o procesadores, no memoria.
- La comunicación entre procesadores: I/O, y de hecho no es acceso a memoria.
- Los bloques de P/M son "convencionales", las mayores variaciones se dan en:
  - Canales o links: hardware (a veces en el procesador) para interconexión.
    - Ej1: transputers: diseñados explícitamente con canales.
    - Ej2: DSP (Digital Signal Processors): con varios ports de I/O y canales de DMA más la documentación necesaria para utilizarlos.
  - Redes de interconexión de procesadores (entre los procesadores)
    - Estáticas: interconexiones fijas entre procesadores, vecindario.
    - Dinámicas: Conexiones pto. a pto. pueden variar en el tiempo.
- Caso muy particular de MIMD losely coupled: clusters
  - Algo más o menos nuevo.
  - Estaciones de trabajo y/o PCs conectadas a una red estándar.

4.1 Clusters como Computadoras Paralelas

Caracterización y terminología: "High Performance Cluster Computing (Architecture, Systems, and Applications)", ISCA-2000, The 27th Annual International Symposium on Computer Architecture, June 10-14, 2000, Vancouver, British Columbia, Canada, Sponsored by ACM SIGARCH, IEEE Computer Society TCCA.

Capas - Visiones

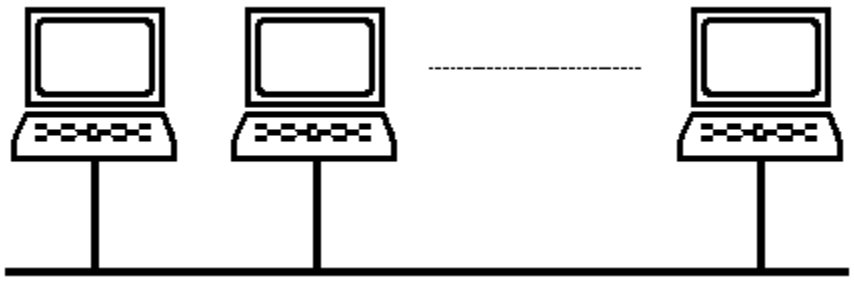


Características de Rendimiento

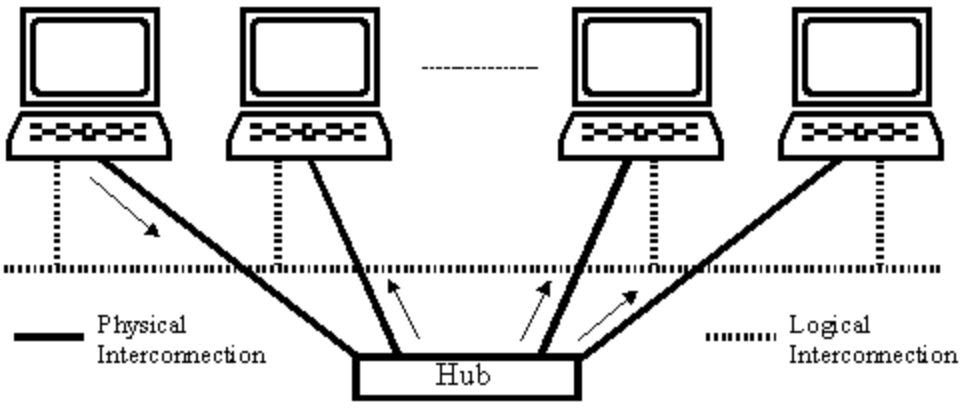
- Los clusters no nacen como máquinas paralelas.
- Las CPUs suelen ser "High Performance".
- Desfasaje entre Cómputo y Comunicación.
- LAN, WAN, MAN, SAN, diferentes objetivos.
- Gran énfasis actual en redes "ad hoc" (costo).
- Algoritmos: áreas nuevas, "trasladados".
- Caracterización de rendimiento *aceptable*...
- Cada capa agrega su overhead.
- Algunas capas no se pensaron para procesamiento paralelo.
- Algunas capas para procesamiento paralelo tienen más overhead del *aceptable*.
- Las capas dan una visión, no rendimiento.
- Overhead $\Longrightarrow$ Aumentar Granularidad.
- Heterogeneidad $\Longrightarrow$ Aplicaciones "aware".

4.2 Clusters Parallel Performance

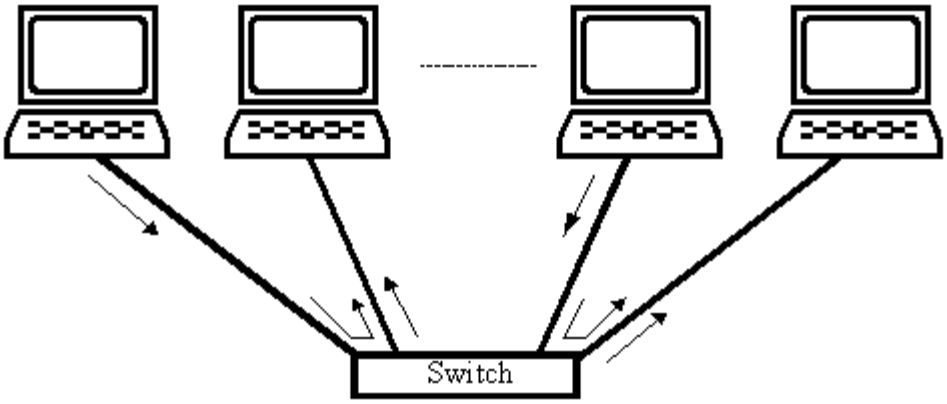
Clusters hardware: computers (PCs and/or workstations) and interconnection network.
Interconnection network: LAN, Ethernet.



Hubs: failures



Switches: performance



Clusters as parallel computers:

- Computing Nodes Performance
  - Sequential: very good
  - Relative: homogeneous and heterogeneous.
  - What is a computer/node in the previous picture?
- Interconnection Network Performance
  - Ethernet: startup and bandwidth
  - Standard Ethernet: bus based (CSMA/CD)
  - Switched Ethernet: switching does not reduce startup.
  - What about non Ethernet networks?
- Parallel Programming Model
  - Shared Memory
  - Message Passing
- Parallel Processing Model
  - SPMD
  - Other/s

4.2.1 Computing Nodes Performance

Homogeneous nodes $\Longrightarrow$ no new problems.

Heterogeneous nodes $\Longrightarrow$ two added tasks:

- Specific performance measures
- Processing workload balance

Specific Performance Measures

1) Specific programs: the same kind of processing but scaled to one processor/node. This does not mean the parallel program with only one task.

2) General benchmarks: generalization $\Longrightarrow$?

Linear algebra: given p computers, $comp_i$, $0 \leq i \leq p-1$, $Mflop/s(comp_i)$, relative computing power, $rp(comp_i)$,

$$rp(comp_i) = rp_i = \frac{Mflop/s(comp_i)}{\max_{j=0\ldots p-1}(Mflop/s(comp_j))}$$

Computing, usage, and underlying idea is analogous by using min or avg. Furthermore, the normalized computing power, $np(comp_i)$, can be defined as

$$np(comp_i) = np_i = \frac{Mflop/s(comp_i)}{\sum_{j=0}^{p-1}(Mflop/s(comp_j))}$$

where

$$0 < np_i < 1$$
$$\sum_{i=0}^{p-1}(np_i) = 1$$

Relationship with Speed Up and Efficiency

Processing Workload Balance

There are many ways: "automatic", dynamic and static processing workload balance. Master-slave could be considered automatic processing workload balance. Dynamic processing workload balance is relatively justified on "unknown" data-dependent systems. Static processing workload balance is strongly "suggested" for linear algebra operations and methods.

Note the relationship among: number of floating points operations and np_i as defined above. Some questions:

- And block processing? (remember flop count is from sequential and *mathematically* defined operation or method).
- What about iterative methods (used on general sparse linear systems)? These methods end when computed solution is *close enough* to the real solution (*convergence*).

A *Little* Comment on Compilers, Code, and Performance

For non optimized code, the compiler, generated binary (Linux FC6 32 or 64 bits) and compiler optimizations become the most important factors related to performance. Just as an example: climate modeling program, AMD Athlon 64 3000+, 1.8 GHz

Binary	**ifort 9.0**	**ifort 10.0**	**Time diff.**
32 bits	01 h. : 37 min.	00 h. : 50 min.	47 min.
64 bits	00 h. : 52 min.	00 h. : 34 min.	18 min.
Time diff.	45 min.	16 min.	

A Comment on Block processing from the Practice with Matrix Multiplication

Remember the basic idea of block processing, specifically applied to multiply two 4×4 matrices: A, B, and C, are subdivided into 2×2 blocks ($bs = 2$). Each block/submatrix will have the *corresponding* row and column indexes

C_{00}		C_{01}	
c_{00}	c_{01}	c_{02}	c_{03}
c_{10}	c_{11}	c_{12}	c_{13}
c_{20}	c_{21}	c_{22}	c_{23}
c_{30}	c_{31}	c_{32}	c_{33}
C_{10}		C_{11}	

=

A_{00}		A_{01}	
a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}
A_{10}		A_{11}	

x

B_{00}		B_{01}	
b_{00}	b_{01}	b_{02}	b_{03}
b_{10}	b_{11}	b_{12}	b_{13}
b_{20}	b_{21}	b_{22}	b_{23}
b_{30}	b_{31}	b_{32}	b_{33}
B_{10}		B_{11}	

Partial Result: $C0_{00} = A_{00}$ x B_{00}
Partial Result: $C1_{00} = A_{01}$ x B_{10}

$$C0_{00} = \begin{array}{|cc|}\hline a_{00}b_{00} + a_{01}b_{10} & a_{00}b_{01} + a_{01}b_{11} \\ a_{10}b_{00} + a_{11}b_{10} & a_{10}b_{01} + a_{11}b_{11} \\ \hline\end{array}$$

$$C1_{00} = \begin{array}{|cc|}\hline a_{02}b_{20} + a_{03}b_{30} & a_{02}b_{21} + a_{03}b_{31} \\ a_{12}b_{20} + a_{13}b_{30} & a_{12}b_{21} + a_{13}b_{31} \\ \hline\end{array}$$

Thus, $C0_{00} + C1_{00}$ should be C_{00}, i.e. the four elements at the *upper left* corner of C...

$$\begin{array}{|cc|}\hline a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} + a_{03}b_{30} & a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} + a_{03}b_{31} \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} + a_{13}b_{30} & a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} \\ \hline\end{array}$$

And the idea is true in general. But computing is not alone in cluster parallel (*high performance*) computing ...

4.2.2 Interconnection Network Performance

Standard parameters/indexes: Latency (or startup) and Bandwidth (or data rate). Modeling message time:

$$t(n) = \alpha + \beta n$$

where n is the number of data items, α is the latency, and β is $bndw^{-1}$ where $bndw$ is the data bandwidth. Usually, latency and bandwidth are found experimentally.

If latency is not taken into account, β is the total cost (in time) per data item (and for short messages α is distributed on the data items), i.e.

$$t(n) = \beta n$$

which is almost true when n is large enough ($\alpha \ll \beta n$).

Latency

According to literature, on 100 Mb/s Ethernet the measured latency is about 0,5 ms. On computers with 1 Gflop/s (10^9 floating point operations per second) this means

$$flxlat = 10^9 \times 5 \times 10^{-4} = 5 \times 10^5 = 500000$$

i.e. waiting for *any* message completion implies a time equivalent to 500000 floating point operations. The *main* problem: latency is almost constant and computer performance is enhanced each year (Moore's Law).