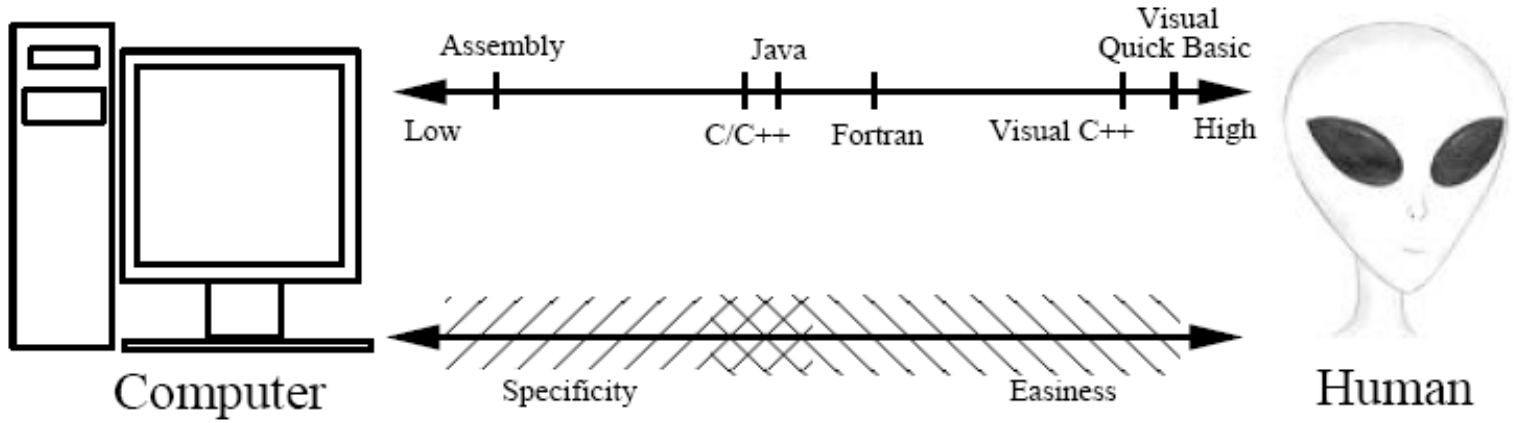
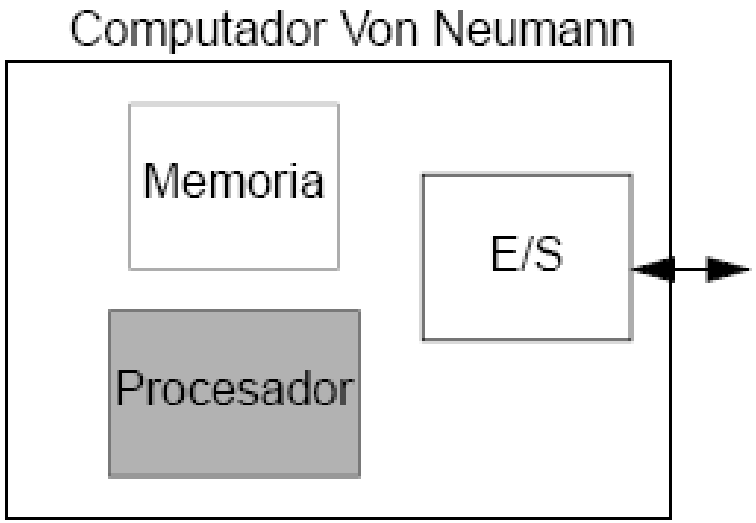
Multiprocesadores

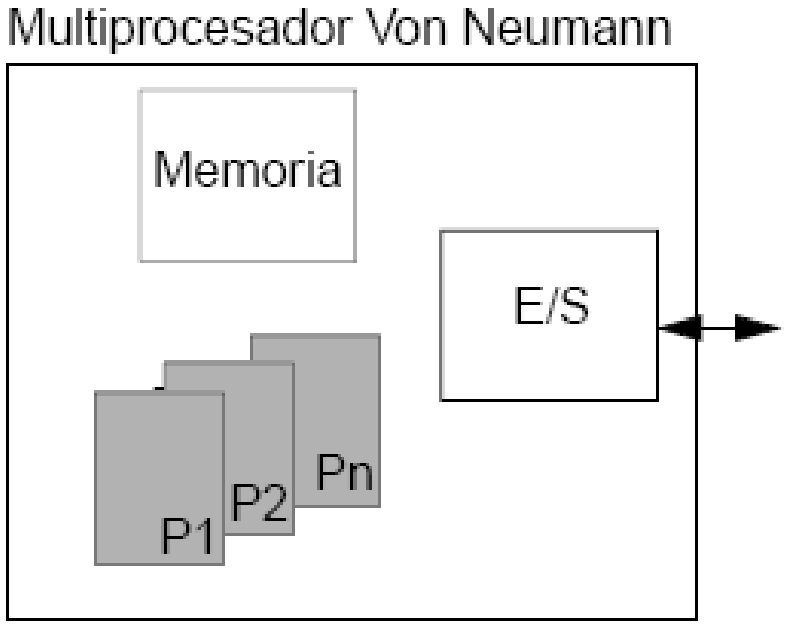
IEC-UTM



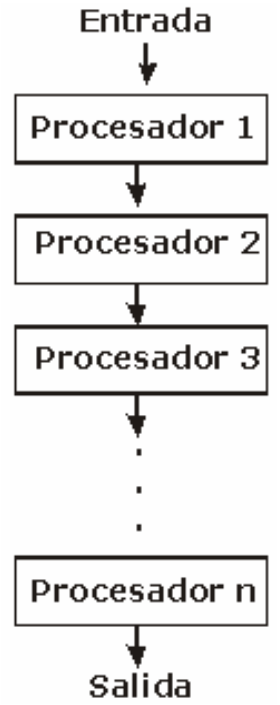
Introducción

- Arquitectura de un equipo paralelo

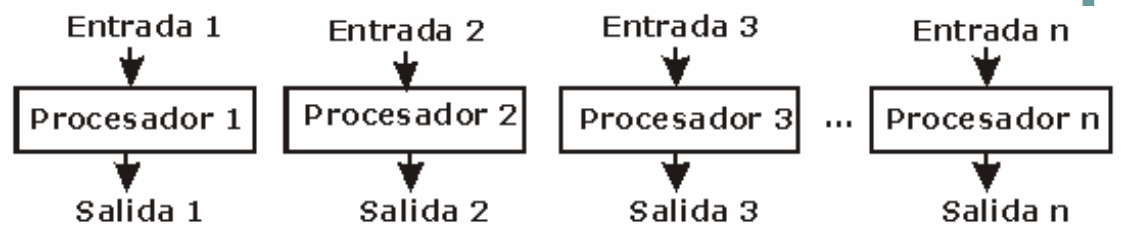




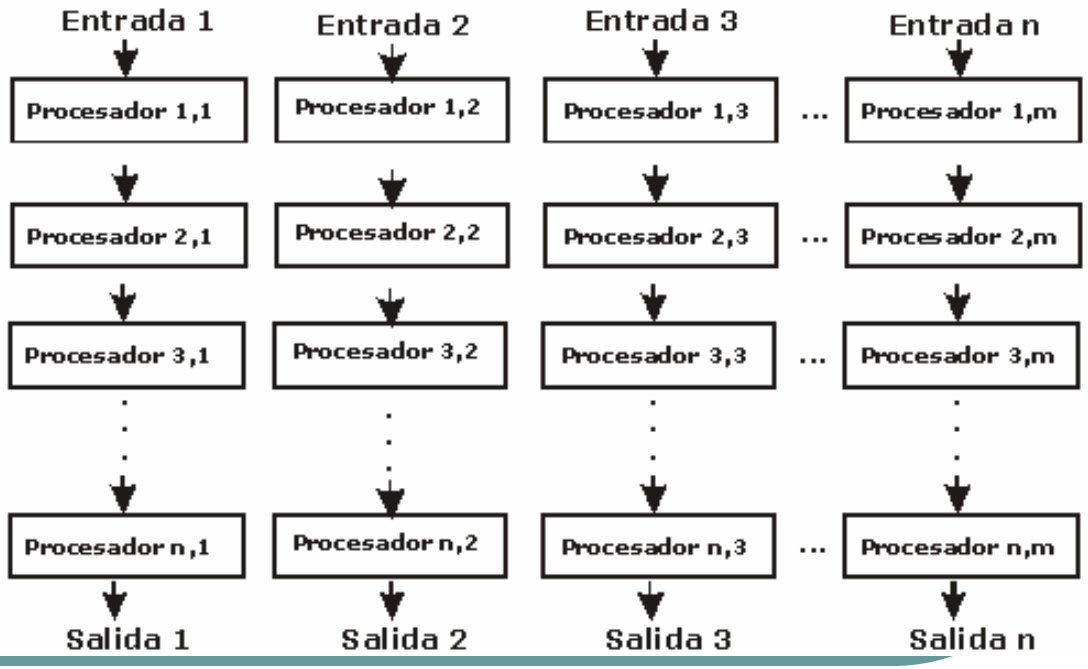
Paradigmas de paralelismo



Paradigma Vector/Array



Paradigma SIMD



Paradigma Sistólico

Clasificación de Flynn

- Flujo único de instrucciones, flujo único de datos (SISD).
- Flujo único de instrucciones, flujo múltiple de datos (SIMD).
- Flujos múltiples de instrucciones, flujo único de datos (MISD).
- Flujos múltiples de instrucciones, flujos múltiples de datos (MIMD).

SISD

- Las computadoras que entran en la clasificación SISD tienen un único procesador y ejecutan una sola instrucción a la vez.

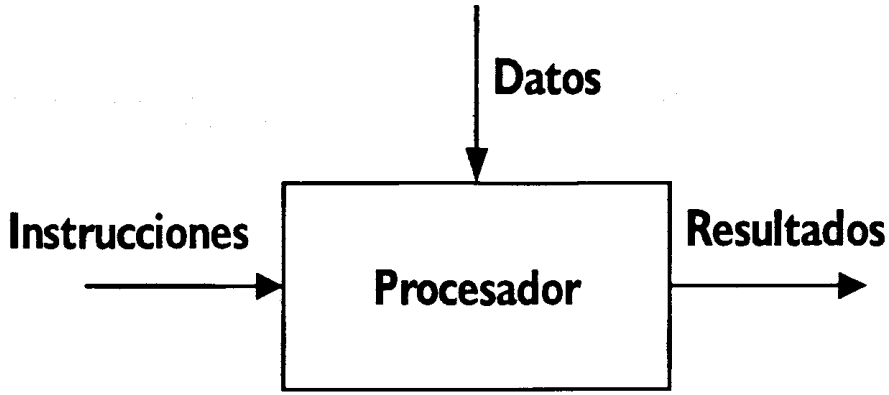


Figura A.4. Arquitectura SISD

SIMD

- En esta arquitectura se tienen *p* procesadores idénticos, los cuales poseen una memoria local. Trabajan bajo un solo flujo de instrucciones originado por una unidad central de control, por lo que se tienen *p* flujos de datos .

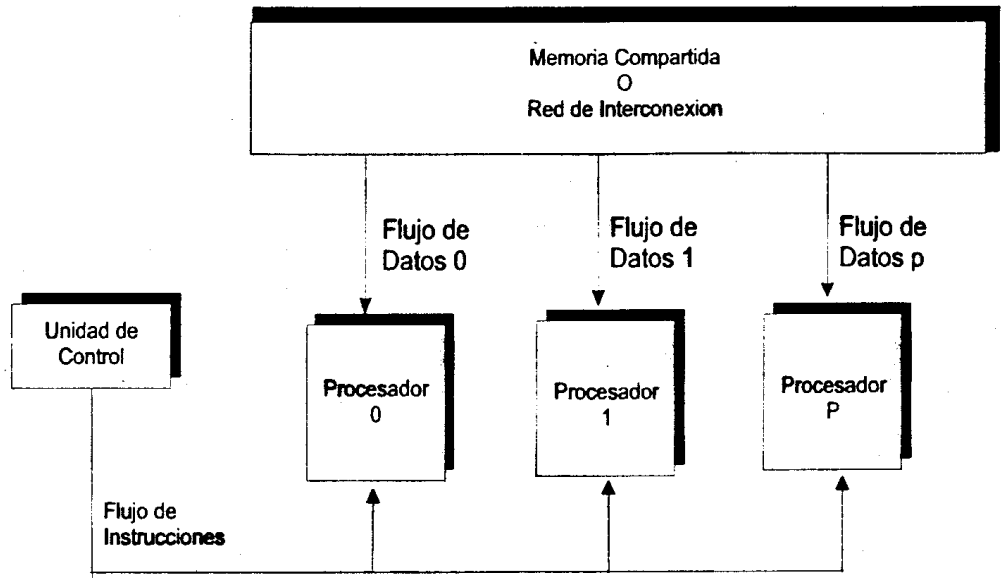
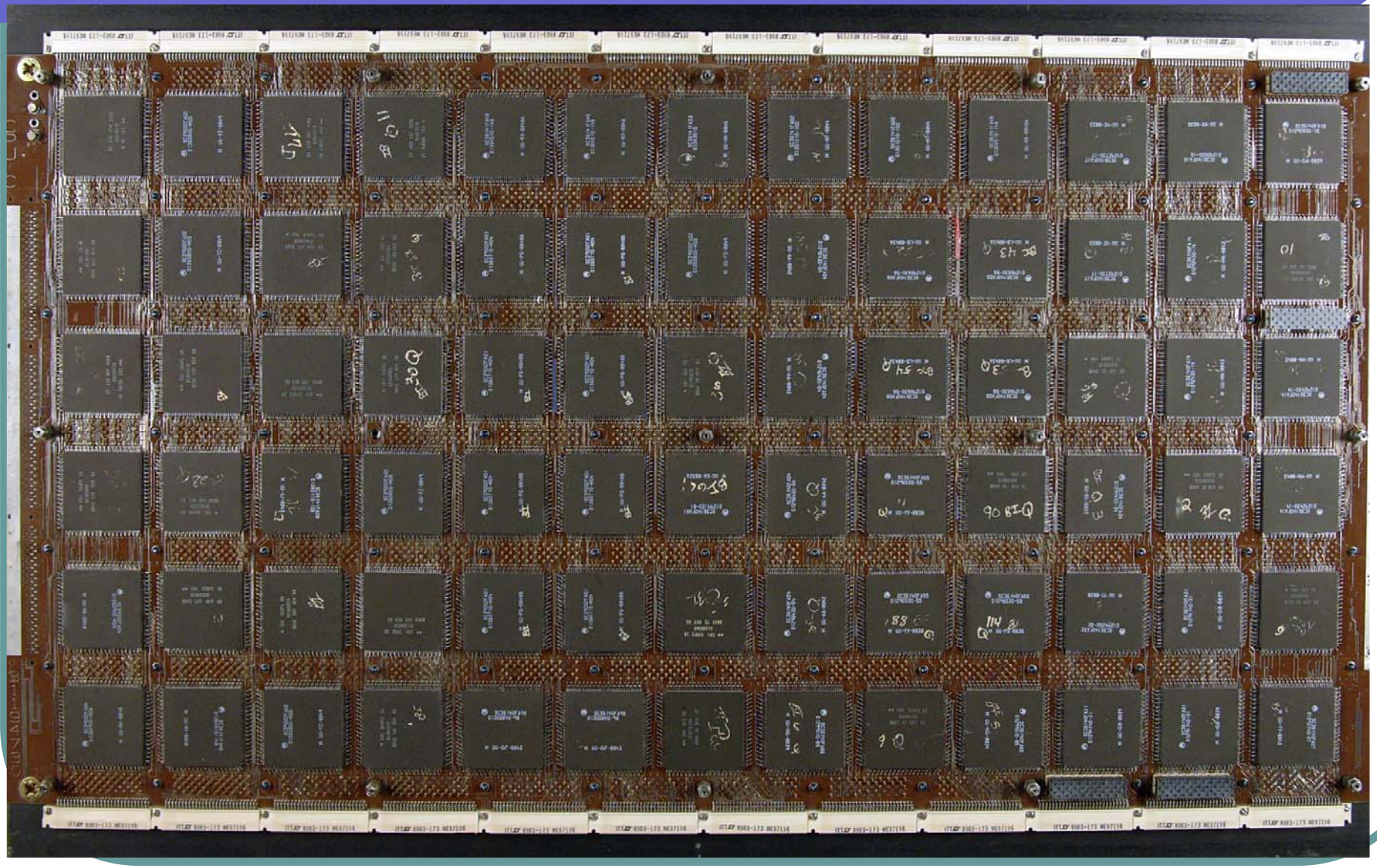


Figura A.5. Arquitectura SIMD

Processor board of a CRAY YMP vector computer (operational ca. 1992-2000). The board was liquid cooled and is one vector processor with shared memory (access to one central memory)



MISD

- Realizan múltiples instrucciones para un solo conjunto de datos.

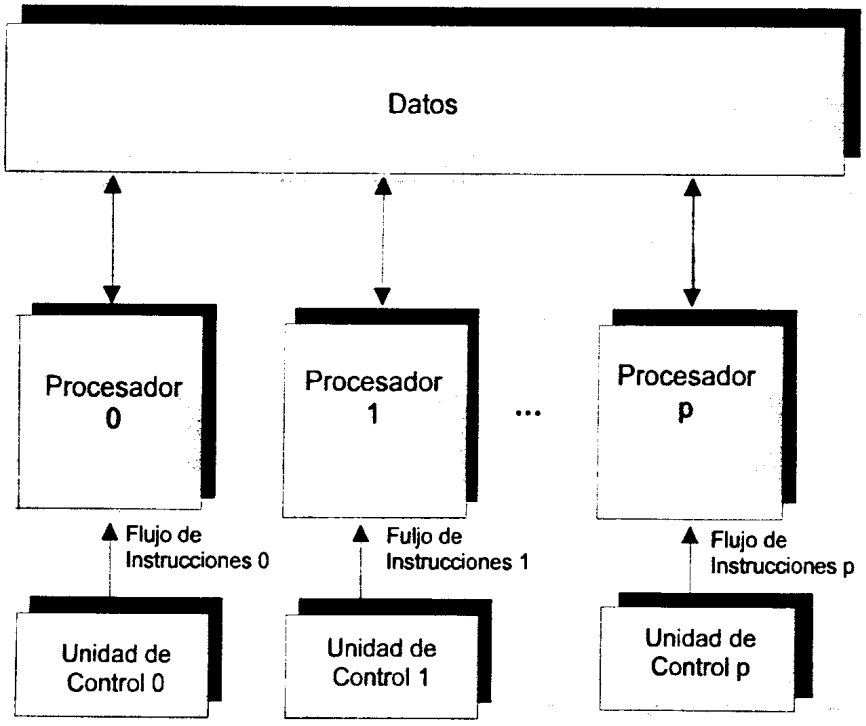


Figura A.6. Arquitectura MISD

MIMD

- La arquitectura MIMD está conformada por *p* procesadores, *p* flujos de instrucciones y *p* flujos de datos. Cada procesador trabaja de modo asíncrono bajo el control de un flujo de instrucciones proveniente de su propia unidad de control

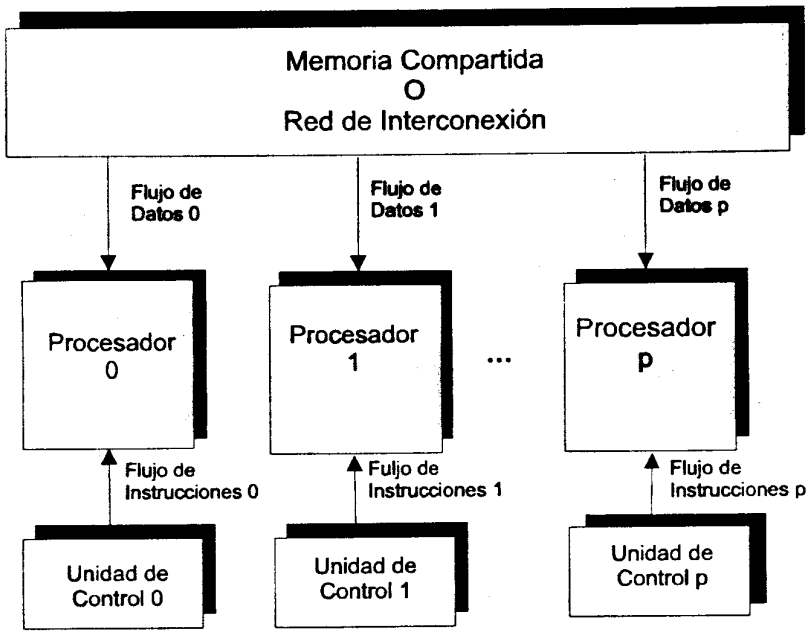
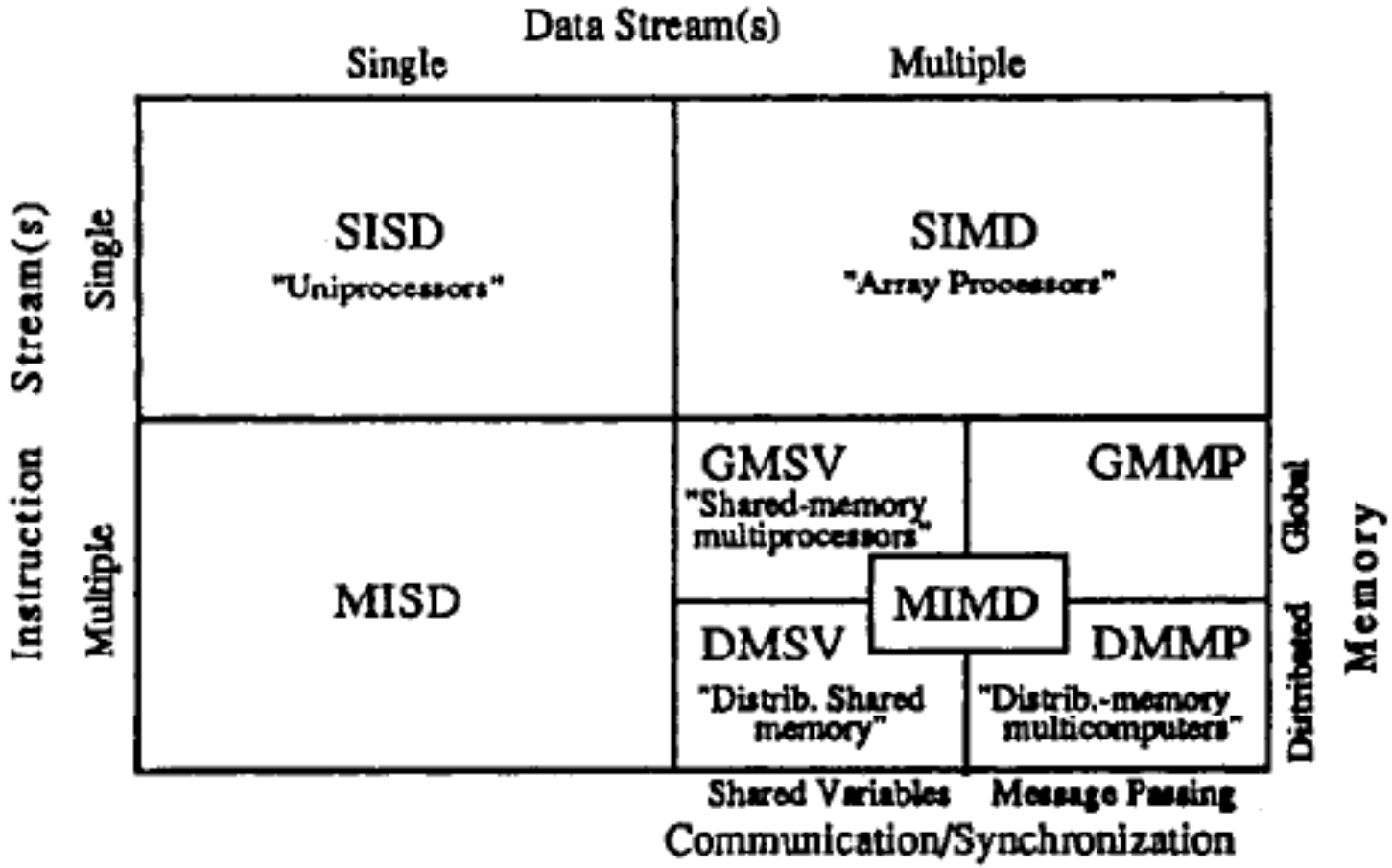


Figura A.7. Arquitectura MIMD

Clasificación de Flynn



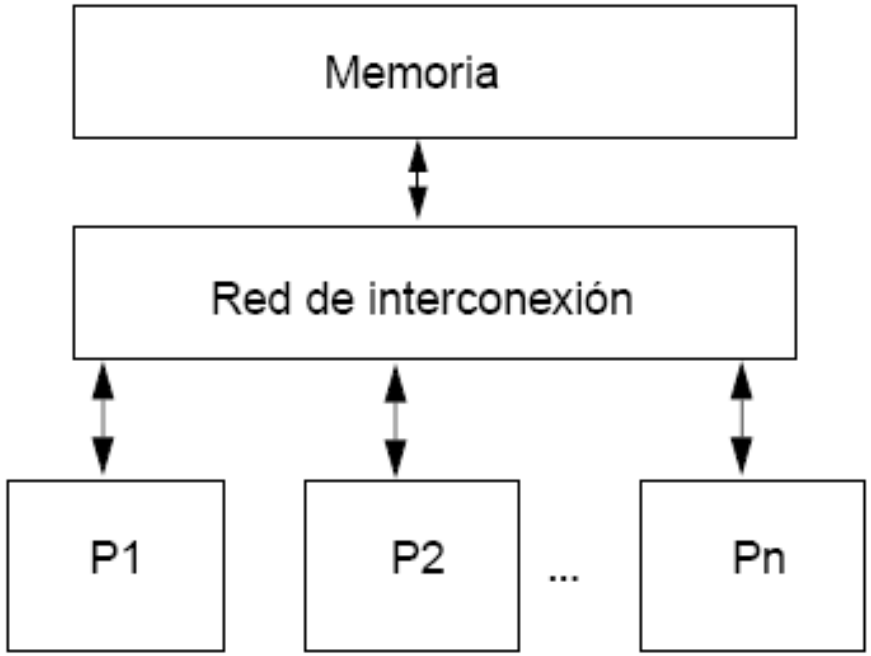
¿Cómo comparten datos los procesadores paralelos ?

- Existen procesadores con un *único espacio de direcciones*, algunas veces llamados *procesadores con memoria compartida*, ofrecen al programador un único espacio de direcciones de memoria que todos los procesadores comparten.
- Los procesadores se comunican a través de variables compartidas en memoria, todos los procesadores tienen acceso a cualquier localidad de memoria a través de cargas y almacenamientos.

Shared Memory Systems

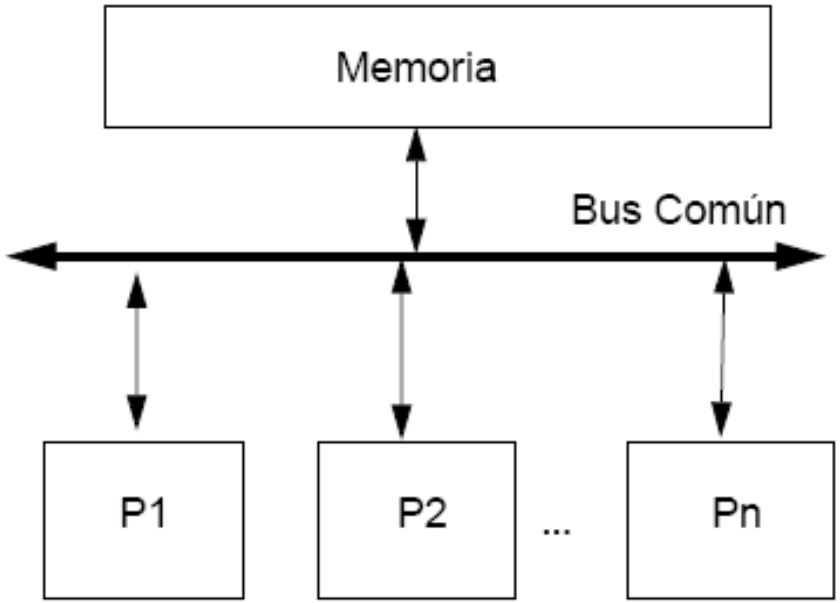
- cada nodo tiene acceso a una amplia memoria compartida que se añade a la memoria limitada privada, no compartida, propia de cada nodo.
- Los sistemas con memoria compartida, tienen multiples CPU's que comparten las mismas direcciones de memoria. Esto significa que que existe una única memoria que es accesada por todas las unidades de procesamiento.
- Los sistemas con memoria compartida pueden ser SIMD o MIMD, en dichos casos se pueden abreviar como SM-SIMD y SM-MIMD respectivamente.
- Para desarrollar programas usando este paradigma, se utiliza OpenMP disponible para C(++) y Fortran

Multiprocesadores con memoria compartida



La red de interconexión permite a cualquier procesador acceder a cualquier posición de memoria.

Multiprocesadores con memoria compartida



Problema

La red de interconexión representa un cuello de botella que impide que esta organización sea escalable.

Arquitectura de un sistema SMP

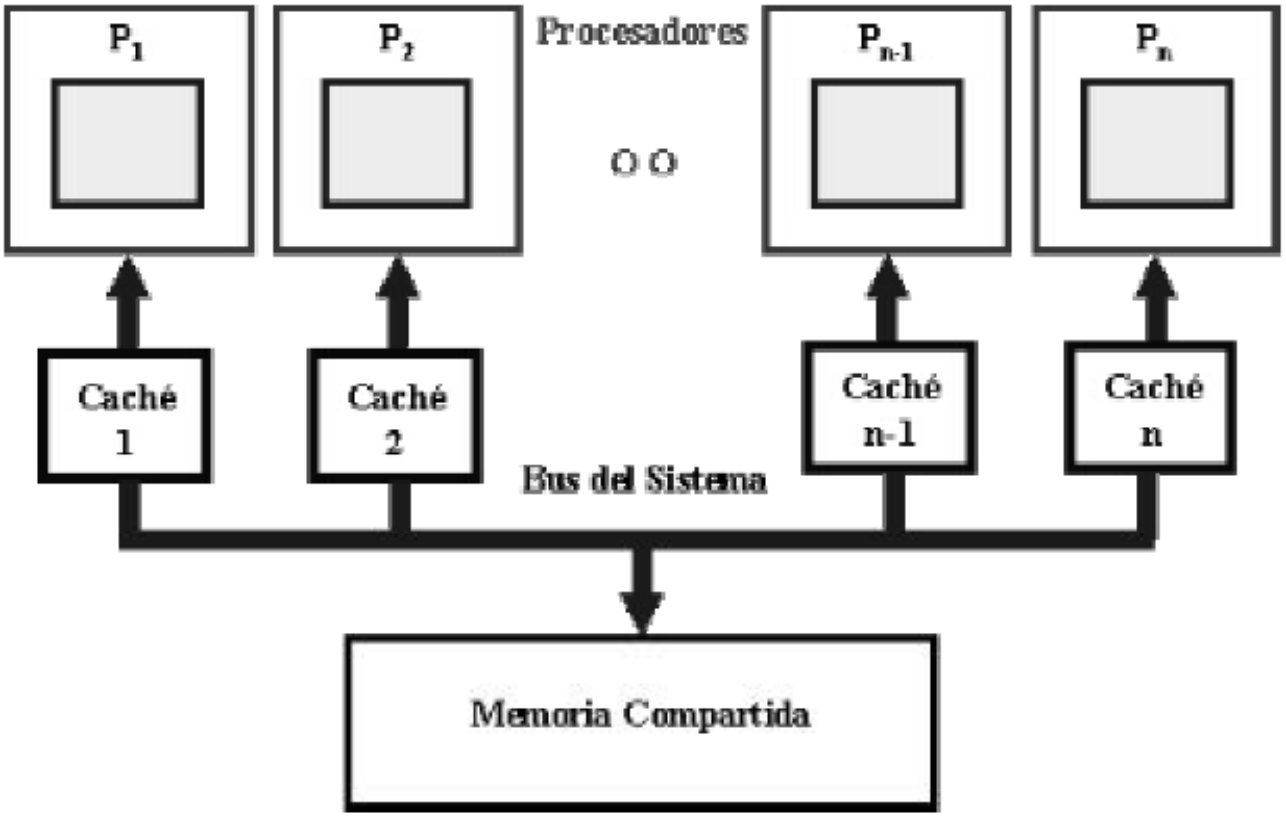


Figura 1.17. Arquitectura de un sistema SMP [15]

Tipos de procesadores con un único espacio de direcciones

- UMA (*uniform memory access)*
  - En estos sistemas todos los procesadores toman el mismo tiempo de acceso a la memoria, también son conocidos como multiptocesadores simétricos (*SMP, symmetric multiprocessors*).
  - Una computadora SMP se compone de microprocesadores independientes que se comunican con la memoria a través de un **bus compartido**. Dicho bus es un recurso de uso común. Por tanto, debe ser arbitrado para que solamente un microprocesador lo use en cada instante de tiempo.

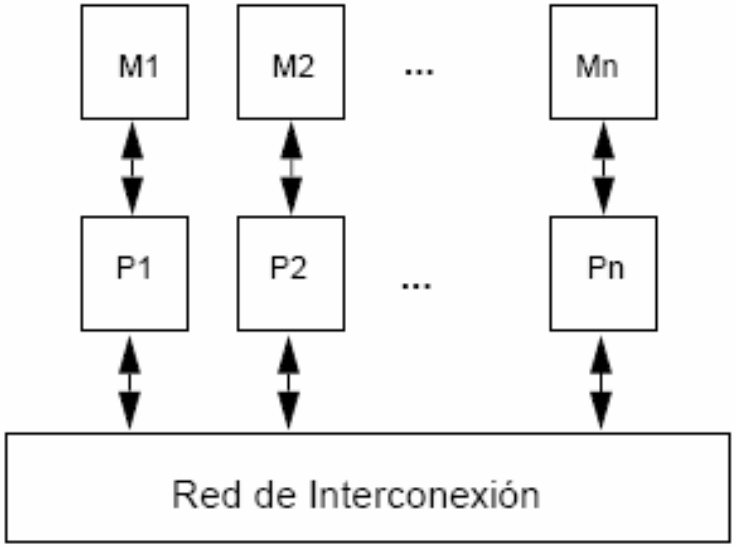
- NUMA (*nonuniform memory access)*
  - algunos accesos a memoria son más rápidos que otros dependiendo de cual procesador esté accesando y a cual palabra, son conocidos como ***multiprocesadores con acceso no uniforme a memoria***

- Como podría esperarse, hay mas desafíos de programación para obtener un rendimiento más alto con un multiprocesador NUMA que con un multiprocesador UMA, pero las máquinas NUMA pueden escalar a grandes tamaños y por lo tanto son potencialmente de un rendimiento más alto.

¿Cómo se coordinan los procesadores paralelos ?

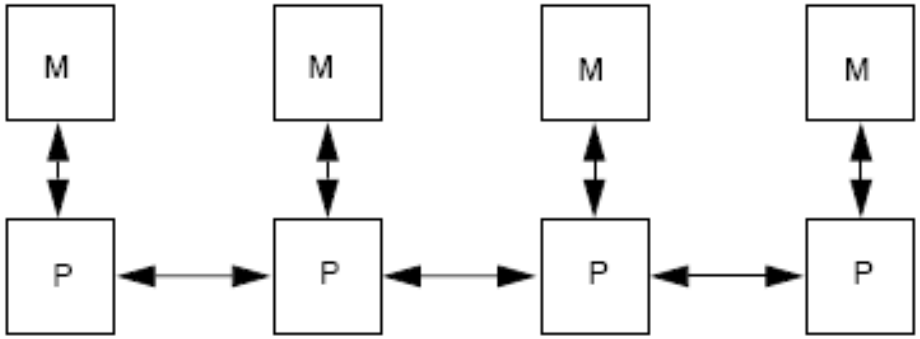
- *La sincronización* es un mecanismo que permite bloquear a otros procesadores mientras sólo uno tiene privilegios de escritura sobre datos compartidos. El resto de los procesadores, en caso de requerir acceso a los datos deben esperar
- Las operaciones de lectura no tienen problemas de sincronización.

Multiprocesadores con memoria distribuida



La red de interconexión permite a cualquier procesador comunicarse con cualquiera de los procesadores del sistema.

Multiprocesadores con memoria distribuida



La organización puede ser escalable en la medida en que los programas realicen comunicaciones locales.

Distributed memory systems

- Cada CPU tiene su propia memoria, los CPU's están interconectados a través de una "red de interconexión" que permite intercambiar datos entre los diferentes nodos cuando se requiere.
- En este tipo de implementaciones, el desarrollador debe controlar donde está cada parte de la información, es decir debe controlar y administrar cada parte de los datos que son distribuidos en todos los nodos.
- Los sistemas distribuidos pueden ser SIMD o MIMD (DM-SIMD o DM-MIMD)
- Los sistemas de interconexión para MIMD, tienen una gran cantidad de topologías de interconexión. En general estas topologías son transparentes para el usuario de tal forma que permita la portabilidad de aplicaciones.

Paso de mensajes

- El modelo alternativo a la memoria compartida utiliza el *paso de mensajes* para comunicación entre procesadores.
- Los procesadores en diferentes computadoras de escritorio se comunican pasando mensajes sobre una red de interconexión. Los sistemas tienen rutinas para *enviar* y *recibir* mensajes, la coordinación es construida con los mensajes que se envían, dado que un procesador sabe cuando un mensaje es enviado y el procesador receptor sabe cuando un mensaje ha llegado.
- El procesador receptor puede entonces enviar un mensaje al emisor indicando que el mensaje ha llegado, si el emisor necesita confirmación.

MPI

- MPI es la primera biblioteca de paso de mensajes estándar que ofrece portabilidad, estandarización y una implementación eficiente de la ejecución paralela, está formada por una colección de rutinas que facilitan la comunicación entre procesadores en programas paralelos

- La estandarización asegura que las llamadas hechas en algún equipo se comportan de igual manera en otras máquinas, independientemente de la implementación usada. La portabilidad permite la implementación de programas en diferentes plataformas como Linux, Solaris, UNIX, Windows NT, entre otras.

- MPI ha sido fuertemente influenciado por las siguientes entidades: Centro de Investigaciones Watson de IBM, Intel's NX/2, Express, nCUBES's Vértex p4 y PARMACS.

- La estructura general de un programa MPI es la siguiente:
  - Inicialización de la comunicación
  - Comunicar para compartir datos entre procesos cada vez que sea necesario.
  - Finalizar el ambiente paralelo.

```
1.  Ejemplo:
2.  #include <stdio.h>
3.  #include <mpi.h>
4.  main(int argc, char **argv){
5.    int soy, tam;
6.    MPI_Init(&argc, &argv);
7.    MPI_Comm_rank(MPI_COMM_WORLD, &soy);
8.    MPI_Comm_size(MPI_COMM_WORLD, &tam);
9.    printf(" Hola Mundo, soy %d de %d.\n“, soy, tam);
10.   MPI_Finalize();
11. }
```

Ejemplo

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

int main(argc,argv)
int argc;
char *argv[]; {
int  numtasks, rank, rc;

rc = MPI_Init(&argc,&argv);
if (rc != MPI_SUCCESS) {
  printf ("Error starting MPI program. Terminating.\n");
  MPI_Abort(MPI_COMM_WORLD, rc);
  }

MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
printf ("Number of tasks= %d My rank= %d\n", numtasks,rank);

/*******  do some work *******/

MPI_Finalize();
}
```

```
#include<iostream.h>
#include<mpi.h>
int main(int argc, char ** argv){
  int mynode, totalnodes;
  int sum,startval,endval,accum;
  MPI_Status status;

  MPI_Init(argc,argv);
  MPI_Comm_size(MPI_COMM_WORLD, &totalnodes); // get totalnodes
  MPI_Comm_rank(MPI_COMM_WORLD, &mynode);     // get mynode
  sum = 0; // zero sum for accumulation
  startval = 1000*mynode/totalnodes+1;
  endval =   1000*(mynode+1)/totalnodes;

  for(int i=startval;i<=endval;i=i+1)
     sum = sum + i;

  if(mynode!=0)
    MPI_Send(&sum,1,MPI_INT,0,1,MPI_COMM_WORLD);
  else
    for(int j=1;j<totalnodes;j=j+1){
      MPI_Recv(&accum,1,MPI_INT,j,1,MPI_COMM_WORLD, &status);
      sum = sum + accum;
    }
  if(mynode == 0)
    cout << "The sum from 1 to 1000 is: " << sum << endl;
  MPI_Finalize();

}
```

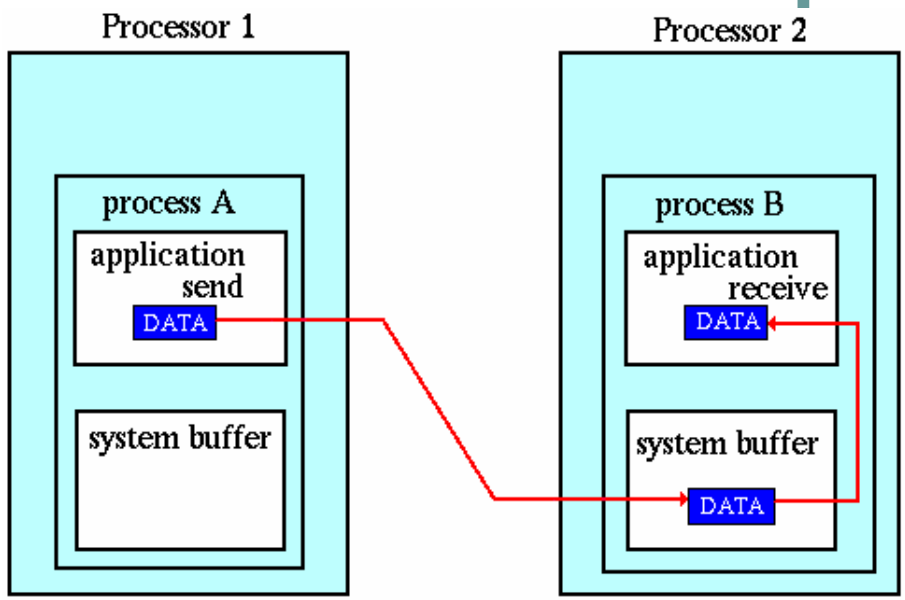
Rutinas

- **MPI_Send**
- Basic blocking send operation. Routine returns only after the application buffer in the sending task is free for reuse.
  - **MPI_Send (&buf,count,datatype,dest,tag,comm)**
  - **MPI_SEND (buf,count,datatype,dest,tag,comm,ierr)**

- **MPI_Recv**
- Receive a message and block until the requested data is available in the application buffer in the receiving task.
  - **MPI_Recv (&buf,count,datatype,source,tag,comm,&status)**
  - **MPI_RECV (buf,count,datatype,source,tag,comm,status,ierr)**

Algunos problemas programando en MPI

- **Buffering:**
- In a perfect world, every send operation would be perfectly synchronized with its matching receive. This is rarely the case. Somehow or other, the MPI implementation must be able to deal with storing data when the two tasks are out of sync.
- Consider the following two cases:
  - A send operation occurs 5 seconds before the receive is ready - where is the message while the receive is pending?
  - Multiple sends arrive at the same receiving task which can only accept one send at a time - what happens to the messages that are "backing up"?
- The MPI implementation (not the MPI standard) decides what happens to data in these types of cases. Typically, a **system buffer** area is reserved to hold data in transit.



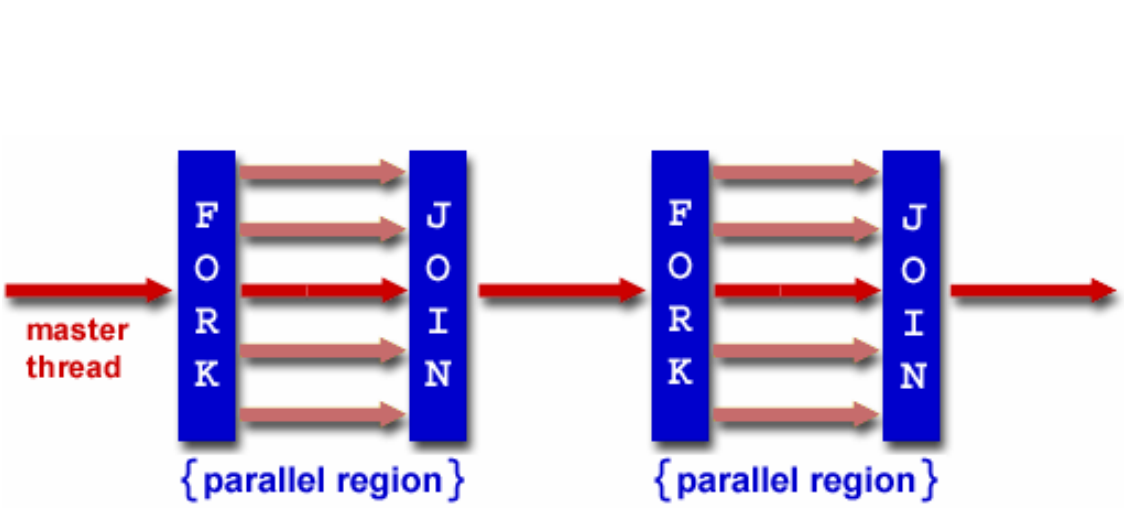
Path of a message buffered at the receiving process

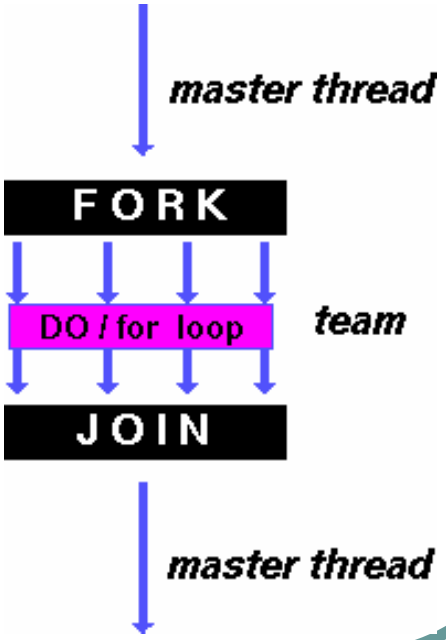
OpenMP

- OpenMP Is An Application Program Interface (API) that may be used to explicitly direct multi-threaded, shared memory parallelism
- The API is specified for C/C++ and Fortran. Supports Fortran (77, 90, and 95), C, and C++
- Multiple platforms have been implemented including most Unix platforms and Windows NT
- OpenMP is based upon the existence of multiple threads in the shared memory programming paradigm. A shared memory process consists of multiple threads.
- OpenMP is an explicit (not automatic) programming model, offering the programmer full control over parallelization.

Fork - Join Model:

- OpenMP uses the fork-join model of parallel execution:
- All OpenMP programs begin as a single process: the **master thread**. The master thread executes sequentially until the first **parallel region** construct is encountered.
- **FORK:** the master thread then creates a ***team*** of parallel threads
- The statements in the program that are enclosed by the parallel region construct are then executed in parallel among the various team threads
- **JOIN:** When the team threads complete the statements in the parallel region construct, they synchronize and terminate, leaving only the master thread





C / C++ - General Code Structure in OpenMP

```
#include <omp.h>

main ()  {
int var1, var2, var3;
Serial code
      ...
Beginning of parallel section. Fork a team of threads.
Specify variable scoping

#pragma omp parallel private(var1, var2) shared(var3)
  {
  Parallel section executed by all threads
        . . .
  All threads join master thread and disband
  }
Resume serial code
      ...
}
```

C / C++ - Parallel Region Example

```
#include <omp.h>

main ()  {

int nthreads, tid;

/* Fork a team of threads giving them their own copies of variables */
#pragma omp parallel private(tid)
  {

  /* Obtain and print thread id */
  tid = omp_get_thread_num();
  printf("Hello World from thread = %d\n", tid);

  /* Only master thread does this */
  if (tid == 0)
    {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
    }

  }  /* All threads join master thread and terminate */

}
```

- Simple vector-add program
- Arrays A, B, C, and variable N will be shared by all threads.
- Variable I will be private to each thread; each thread will have its own unique copy.
- The iterations of the loop will be distributed dynamically in CHUNK sized pieces.
- Threads will not synchronize upon completing their individual pieces of work (NOWAIT).

```
#include <omp.h>
#define CHUNKSIZE 100
#define N     1000

main ()
{
int i, chunk;
float a[N], b[N], c[N];
/* Some initializations */
for (i=0; i < N; i++)
  a[i] = b[i] = i * 1.0;
chunk = CHUNKSIZE;
#pragma omp parallel shared(a,b,c,chunk) private(i)
  {
  #pragma omp for schedule(dynamic,chunk) nowait
  for (i=0; i < N; i++)
    c[i] = a[i] + b[i];

  }  /* end of parallel section */

}
```

Clasificación de los Multiprocesadores

ORGANIZACIÓN	PROGRAMACIÓN: Variables Compartidas	PROGRAMACIÓN: Paso de Mensajes
Memoria Compartida	***SMP*** *(Symmetric Multiprocessors)* Combinación natural Poco escalable Fácil de programar	***?*** Poco escalable Programación difícil
Memoria Distribuida	***DSM*** *(Distributed Shared Memory)* Programación fácil Escalable Implementación difícil	***Multicomputer*** Combinación natural Programación difícil Escalable

Problemas y Retos

- **Hardware**
  - Buscar organizaciones del sistema que permitan al software obtener una fracción significativa de la velocidad máxima del sistema.
- **Aplicaciones y Algoritmos**
  - Identificar aplicaciones críticas que pueden beneficiarse del uso de los sistemas multiprocesadores.
  - Encontrar algoritmos paralelos eficientes para resolver los núcleos computacionales más habituales.
- **Herramientas**
  - Desarrollar herramientas (modelos de programación, compiladores, depuradores, monitorizadores de rendimiento, etc) que faciliten el uso del sistema.

Problemas en la programación paralela

a. Las comunicaciones disminuyen la aceleración de los algoritmos paralelos - La ley de Amdahl

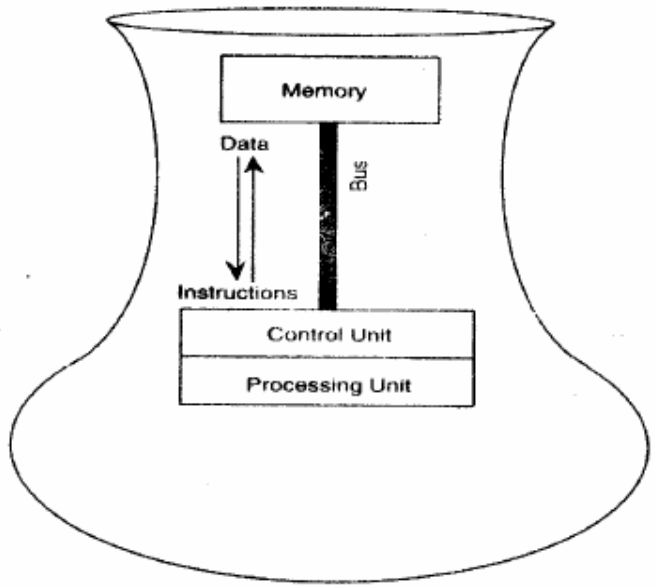
b. El programador debe saber mucho acerca del hardware, un conocimiento amplio para escribir programas que sean rápidos y capaces de ejecutarse en un número variable de procesadores.

c. Un programa adaptado a un multiprocesador no es transportable a otros multiprocesadores.

Cuello de botella de Von Neumann

- La velocidad de procesamiento del sistema está limitada entonces por la rapidez en que los datos e instrucciones pueden ser transportados desde la memoria hasta la unidad de procesamiento. Esta conexión entre la memoria y la unidad de procesamiento es conocida como el cuello de botella de Von Neumann.

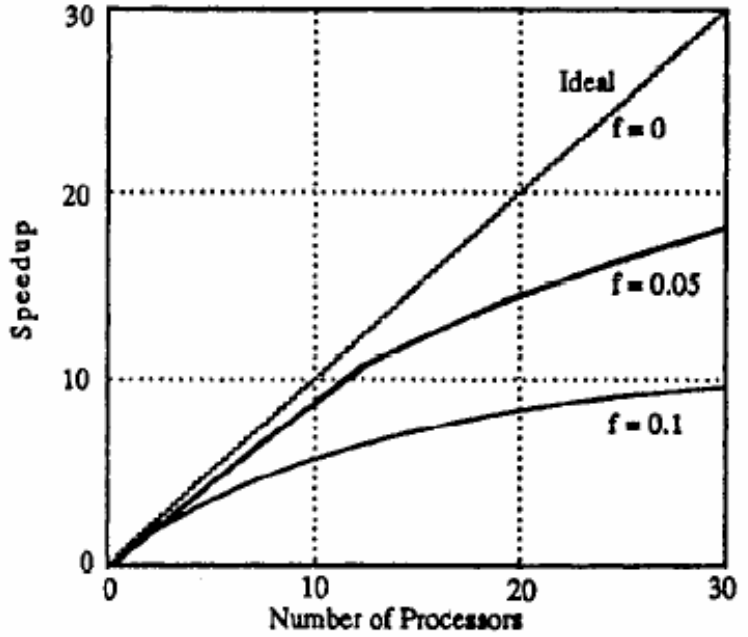
- Este cuello de botella puede ser evitado empleando más de una unidad de procesamiento y muchas memorias, así varios flujos de datos pueden estar activos al mismo tiempo



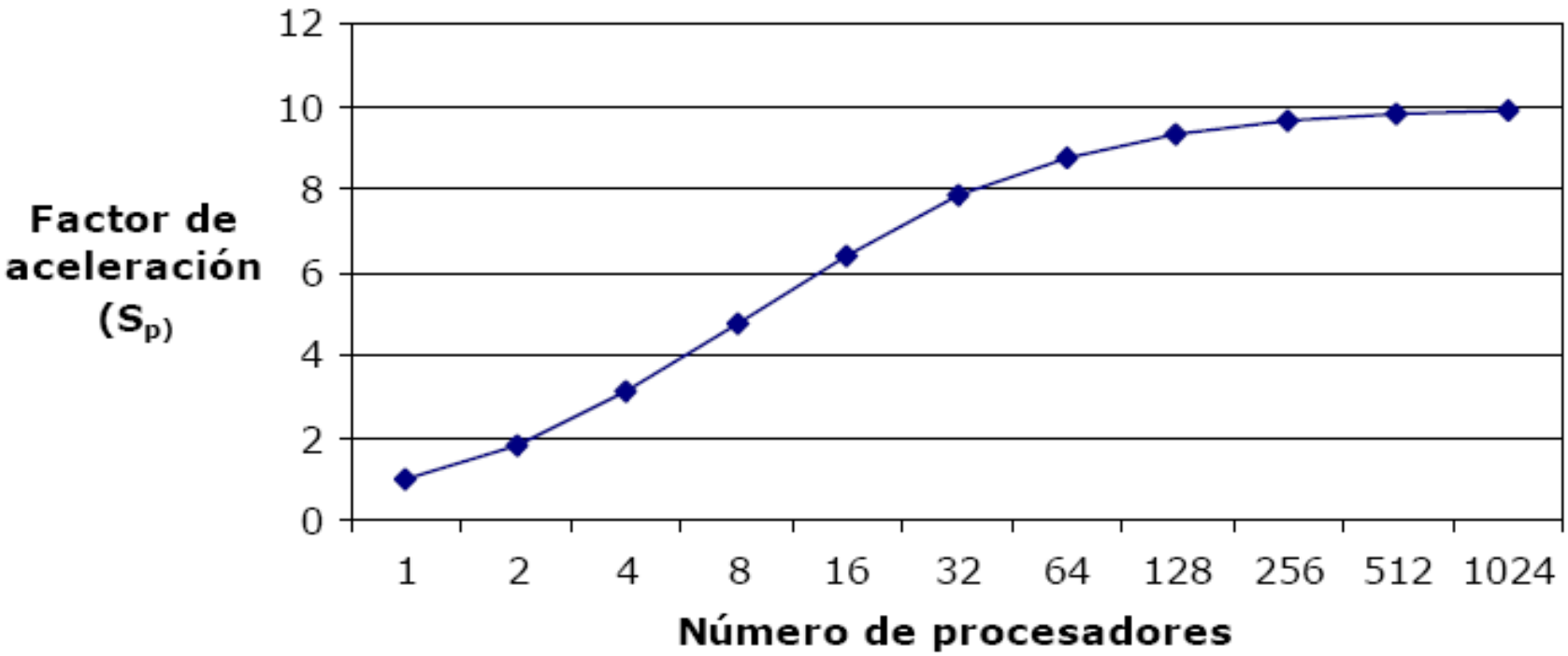
La ley de Amdahl

- Un número pequeño de operaciones secuenciales puede limitar el factor de aceleración de un algoritmo paralelo, estas operaciones son necesarias para la sincronización de todos los procesadores.
- f = fracción de operaciones secuenciales
- p = # procesadores
- Sp = aceleración

$$S_p \leq \frac{1}{f + \frac{(1-f)}{p}}$$



Aplicación de la ley de Amdahl

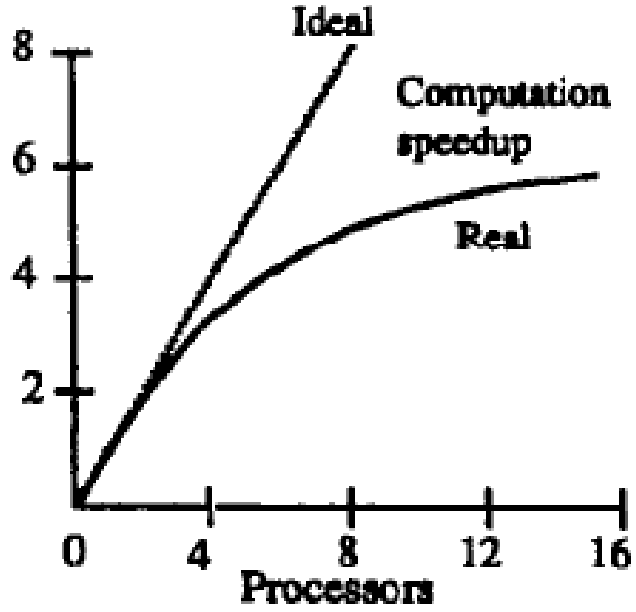


Curvas de velocidad

- El propósito de estas curvas es comparar el tiempo del programa serial más rápido T1 contra el tiempo del programa paralelo equivalente T(*N*), donde *N* es el número de procesadores usados.

$$S \leq \frac{T(1)}{T(N)}$$

Curvas de rapidez



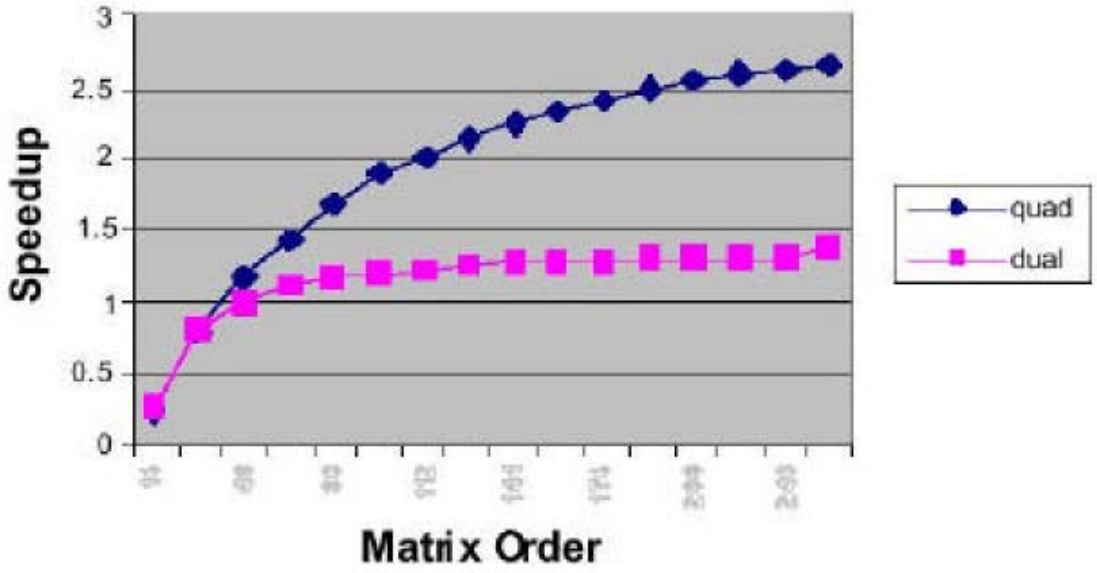
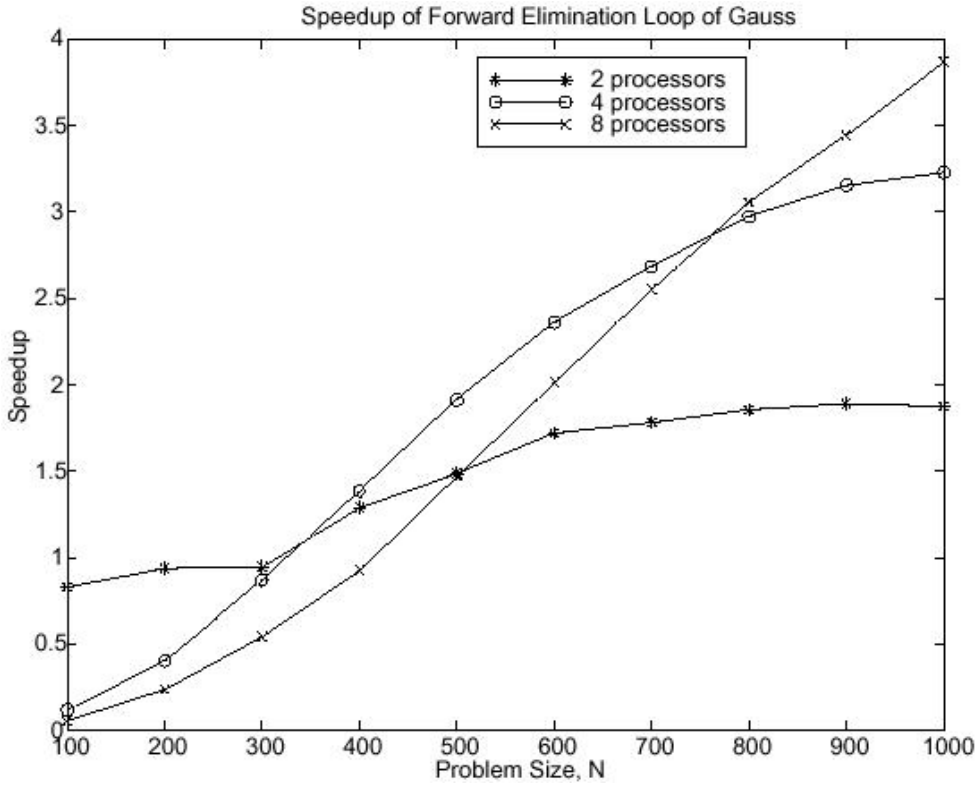


Figura 1.8. Curvas de rapidez *dual* vs. *Quad* para la Factorización **LU** [3]

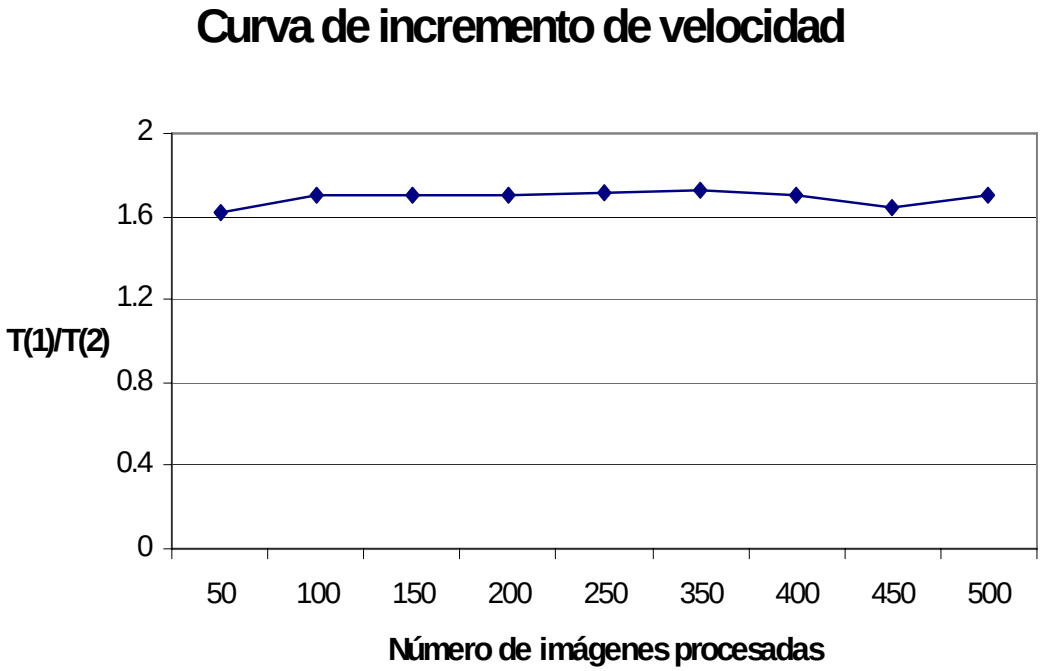
Curvas de rapidez

- **El incremento del número de procesadores no siempre garantiza la disminución del tiempo en el que es realizada una determinada tarea**.
- En el Instituto de Supercómputo de Minesota se hicieron pruebas al modelo de paralelización JavaSpMT (Java Speculative MultiThreading), que son librerías específicas para la programación de hilos. Una de las aplicaciones de prueba que se hicieron fue para resolver sistemas de ecuaciones lineales de *nxn* usando el método de eliminación gausiana. Las pruebas fueron hechas en un sistema multiprocesador IP25 SGI Challenge a 196 MHz con memoria compartida y 8 procesadores MIPS R10000.

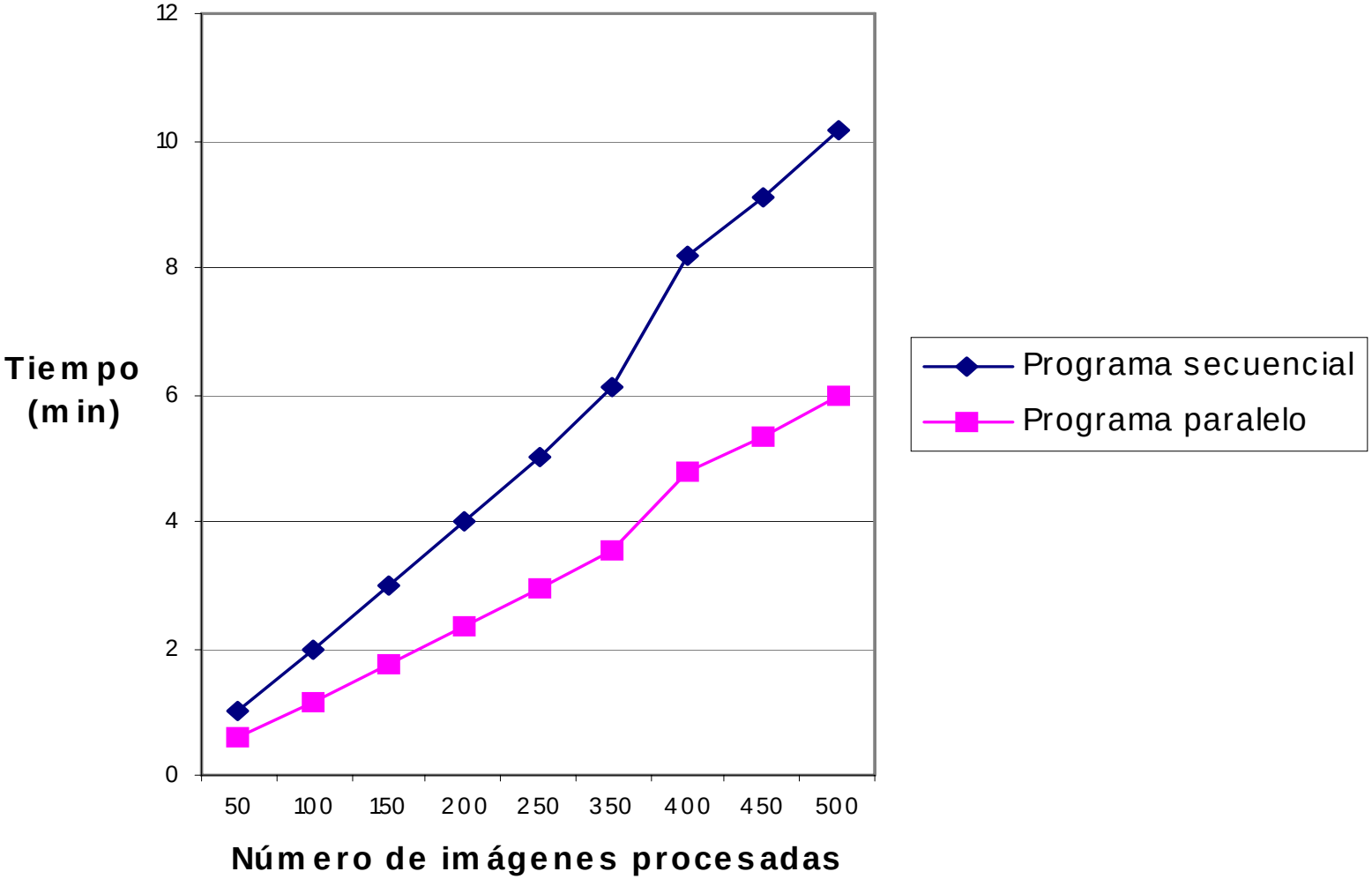


Curvas de rapidez al utilizar 2, 4, y 8 procesadores para resolver un sistema de ecuaciones lineales de *nxn,* por el método de Gauss

Número de Imágenes procesadas	Tiempo del programa secuencial, T(1) (min)	Tiempo del programa en paralelo, T(2) (min)	T(1)/T(2)
50	1	0.6166	1.6166
100	1.9833	1.1666	1.7
150	3	1.76	1.698
200	4.02	2.366	1.697
250	5.033	2.9333	1.7159
350	6.1	3.533	1.7264
400	8.166	4.7833	1.7073
450	9.1166	5.333	1.6475
500	10.1666	5.9666	1.7039



Comparación de los tiempos de procesamiento de los programas paralelo y el secuencial.



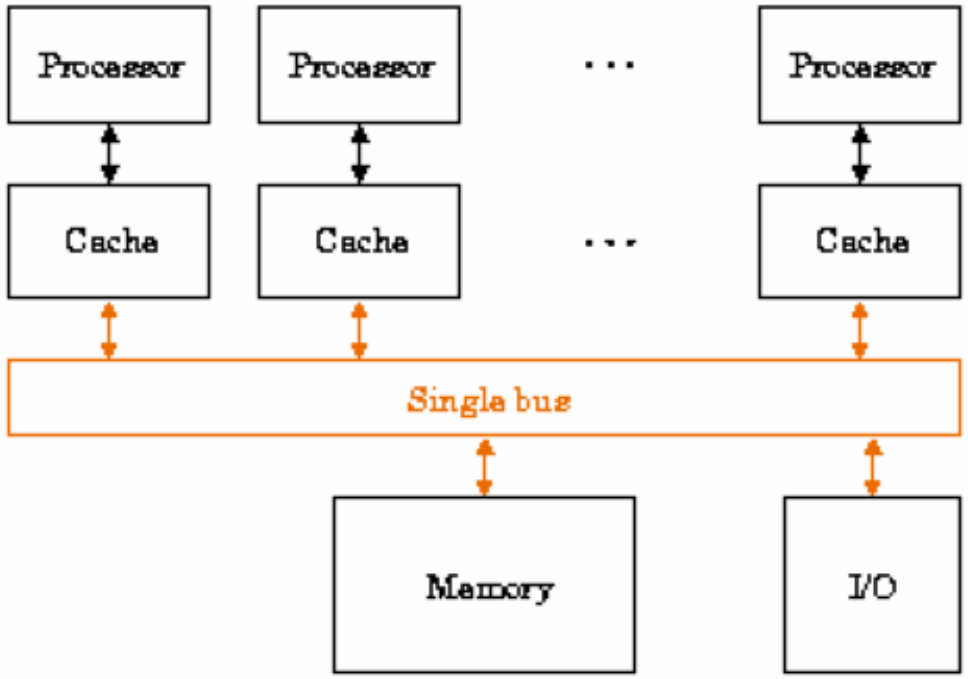
Granularidad

- ***Granularidad:*** se refiere al número de operaciones de cómputo realizadas en cada módulo del algoritmo paralelo.
  - **Granularidad gruesa.-** ocurre cuando en el algoritmo hay un número elevado de operaciones de cómputo y como consecuencia tiene pocas operaciones de comunicación.
  - **Granularidad fina.-** ocurre cuando el algoritmo tiene pocas operaciones de cómputo y un elevado número de comunicaciones.

Multiprocesadores conectados por un solo bus

- El alto rendimiento y el bajo costo de los microprocesadores inspiró renovados intereses en los multiprocesadores en los 60’s.
- Varios microprocesadores pueden colocarse adecuadamente en un bus común por diversas razones:
  - Cada microprocesador es más pequeño que un procesador multi-chip, de manera que más procesadores pueden ser situados sobre un bus.
  - La caché puede reducir el tráfico del bus.
  - Se tienen los mecanismos para mantener la consistencia entre la caché y la memoria para multiprocesadores, así como caché y memoria se mantienen consistentes con I/O, por lo tanto se simplifica la programación.

- El tráfico por procesador y el ancho de banda del bus determinan el número útil de procesadores en tales microprocesadores. Las caché s replican datos en sus memorias más rápidas tanto para reducir la latencia a los datos *y* reducir el tráfico de memoria sobre el bus.



Multiprocesadores de un solo bus,
su tamaño típico está entre 2 y 32 procesadores.

Computadoras con multiprocesadores conectadas por un solo bus, disponibles en el mercado en 1997

Nombre	Máximo número de procesadores	Nombre del procesador	Frecuencia de reloj	Máximo tamaño de memoria/Sistema	Razón de comunicación
Compaq ProLiant 5000	4	Pentium Pro	200 MHz	2, 048 MB	540 MB/seg.
Digital AlphaServer 8400	12	Alpha 21164	440 MHz	28, 672 MB	2150 MB/seg.
HP 9000 K460	4	PA-8000	180 MHz	4, 096 MB	960 MB/seg.
IBM RS/6000 R4	8	PowerPC 604	112 MHz	2, 048 MB	1800 MB/seg.
SGI Power Challenge	36	MIPS R10000	195 MHz	16, 384 MB	1200 MB/seg.
Sun Enterprise 6000	30	UltraSPARC 1	167 MHz	30, 720 MB	2600 MB/seg.

Ejemplo: Programa paralelo (Un solo bus)

- Se quieren sumar 100, 000 números con una computadora cuyo multiprocesador es de un solo bus. Se asume que cuenta con 10 procesadores. Realizar un programa para esta tarea

- **Respuesta:**
  - El primer paso consistiría en dividir el conjunto de números en subconjuntos del mismo tamaño. Dado que hay una memoria única para esta máquina, los subconjuntos tendrán diferentes direcciones de inicio que usará cada procesador.
  - *Pn* es el número del procesador, entre 0 y 9. Todos los procesadores inician el programa ejecutando un lazo que suma sus subconjuntos de números:

```
1. Sum[ Pn ] = 0;
2. for ( i = 10000*Pn; i < 10000*(Pn + 1); i = i + 1 )
3.     Sum[ Pn ] = Sum[Pn] + A[i]; /* Suman sus áreas asignadas */
```

- El siguiente paso consiste en sumar estas sumas parciales, se aplicará una estrategia del tipo divide y vencerás.

- La mitad de los procesadores sumará un par de las sumas parciales, después un cuarto sumará un par de cada una de las nuevas sumas parciales, y así hasta obtener una única suma final.

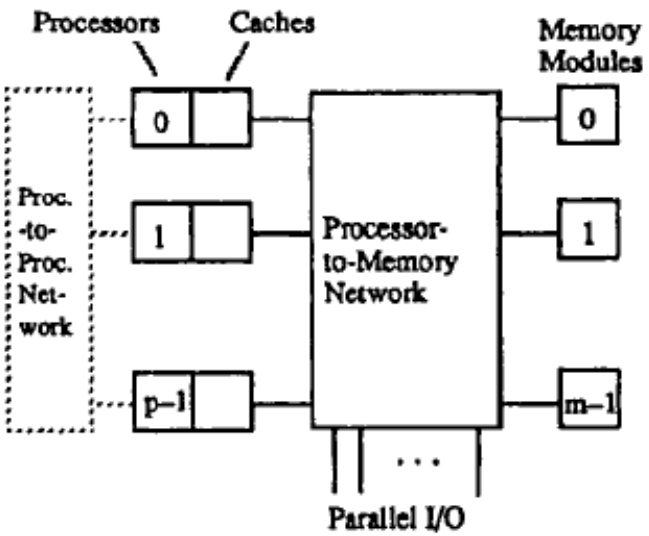
- Es conveniente que cada procesador tenga su propia versión de la variable de control de lazo *i*, de manera que se debe indicar que ésta es una variable privada (*private*).

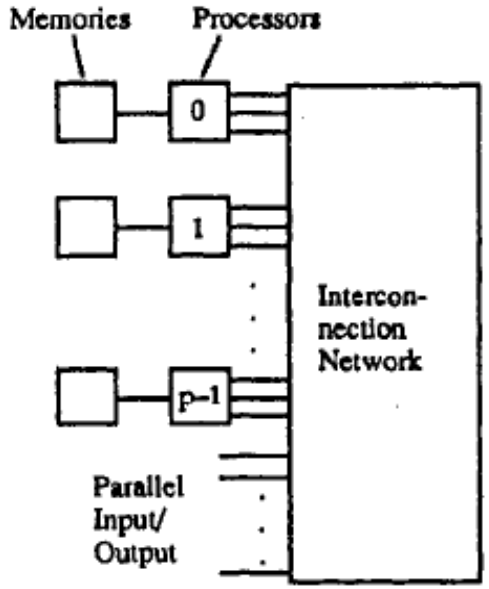
- En este ejemplo, los procesadores deben sincronizarse antes de que los procesadores “consumidores” intenten leer el resultado desde la ubicación de la memoria escrito por los 10 procesadores “productores”; en otro caso, el consumidor puede leer los valores anteriores de los datos.

- El código es el siguiente (*half* es una variable privada):

```
Half = 10; /* 10 procesadores en un multiprocesador de un bus
Do {
        Synch(); /* Espera a que la suma parcial se complete */

        If ( Half % 2 != 0 && Pn == 0 )
                Sum[0] = Sum[0] + Sum[Half – 1];

        Half = Half / 2; /* Línea de división sobre los que suman

        If ( Pn < Half )
                Sum[ Pn ] = Sum[ Pn ] + Sum[ Pn + Half ]
} while ( Half != 1); /* Termina con la suma final en Sum[0] */
```

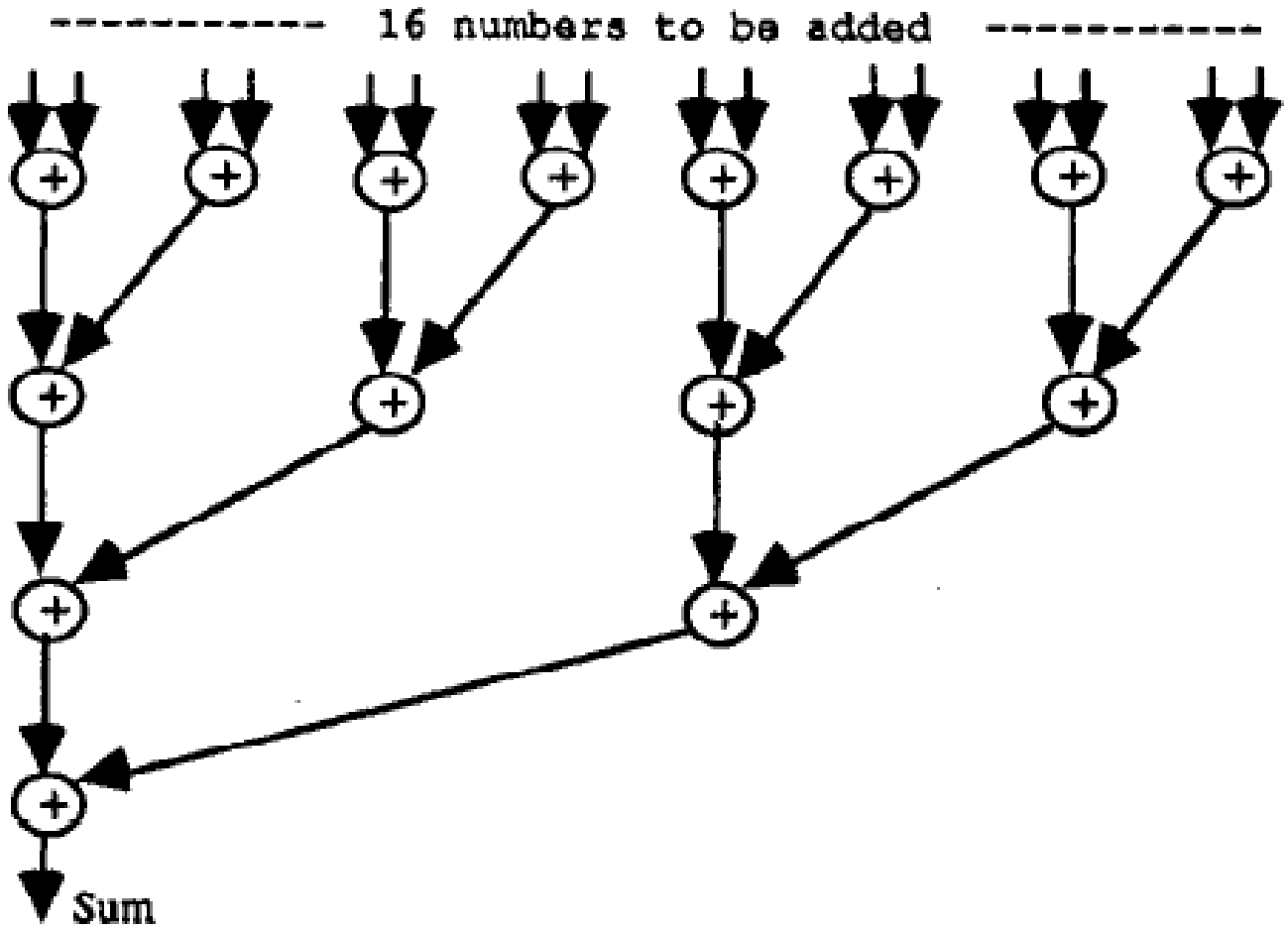
Configuraciones de memoria compartida y distribuida



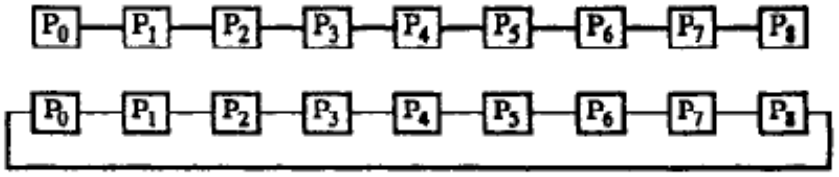


A parallel processor with global memory and processor caches.

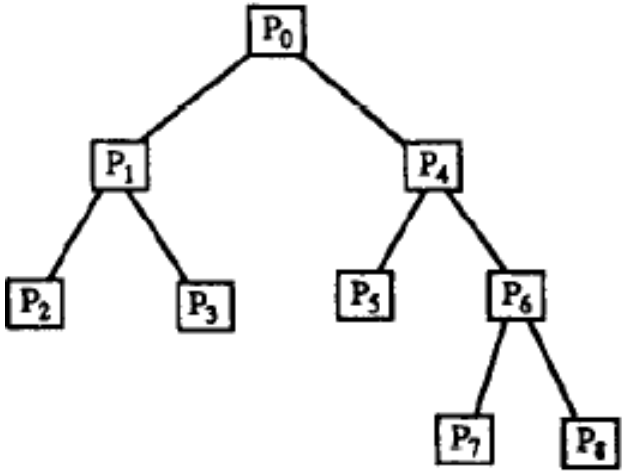
Suma de 16 números en paralelo



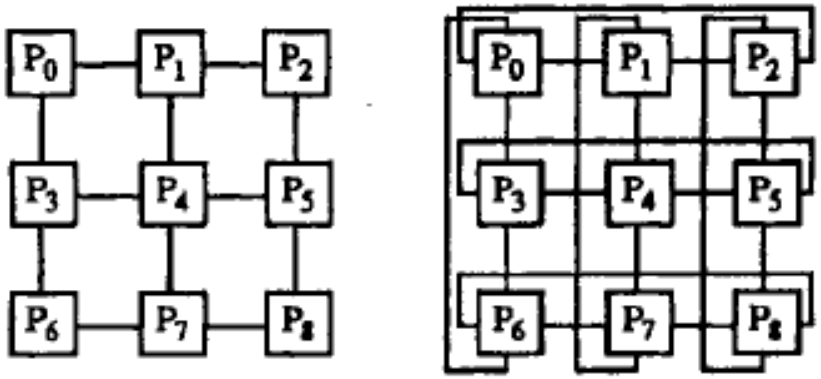
Configuraciones de redes de interconexión



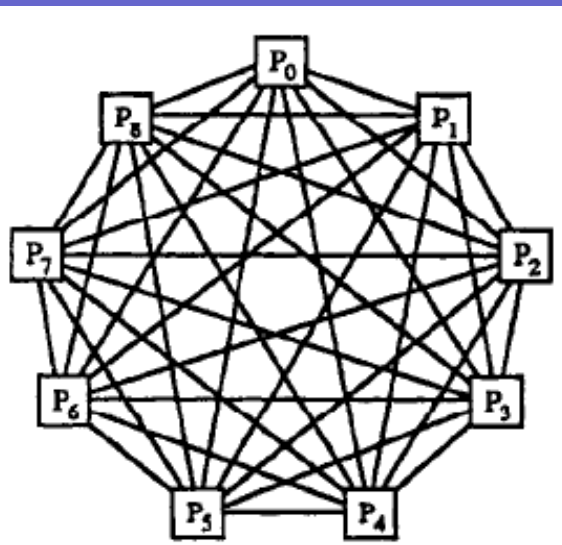
A linear array of nine processors and its ring variant

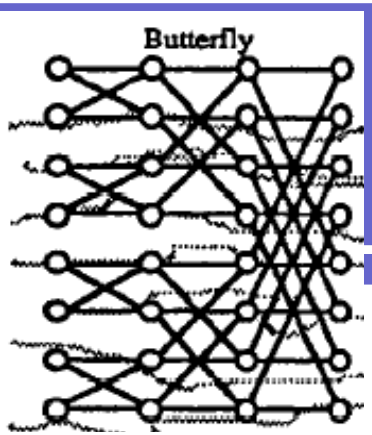


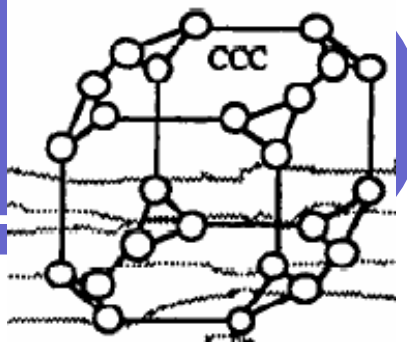
. A balanced (but incomplete) binary tree of nine processors.



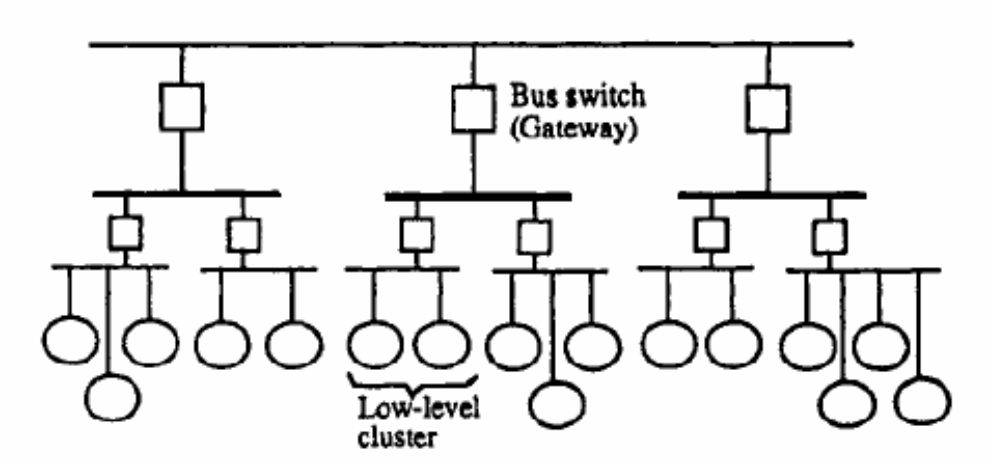
A 2D mesh of nine processors and its torus variant.



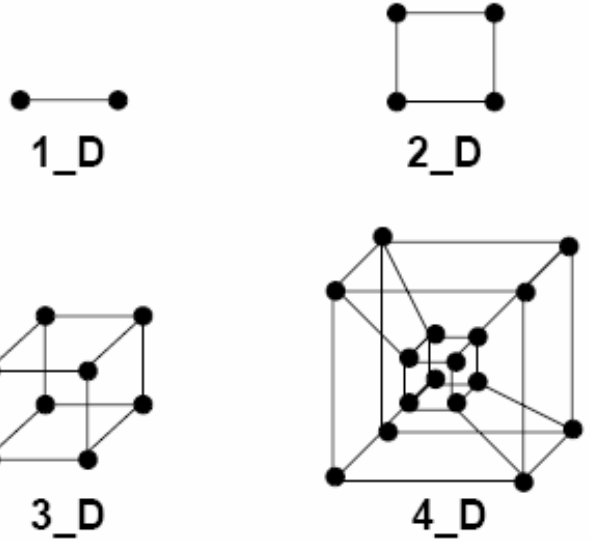


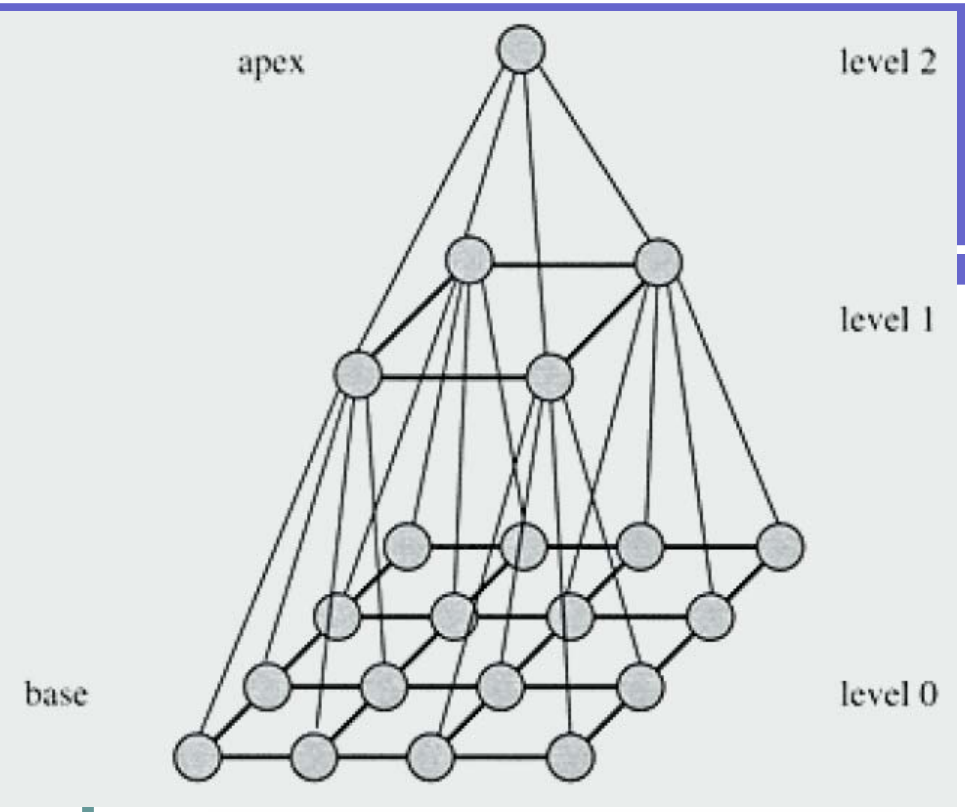


A shared-variable architecture modeled as a complete graph.

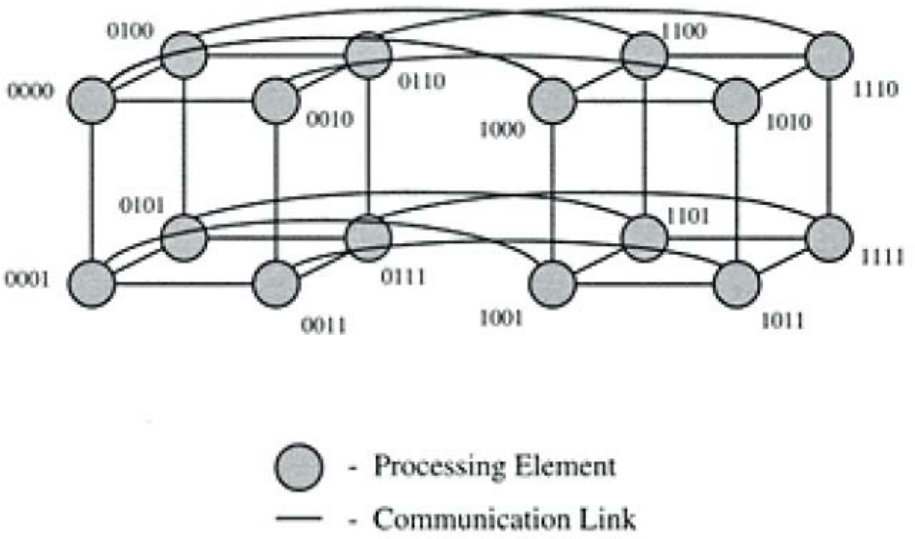


Hipercubo

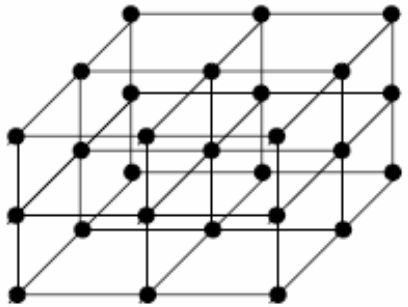




A pyramid computer of size 16.



A hypercube of size $n = 16$ with the nodes labeled using a binary representation.



Malla 3_D

Clusters

- Simplemente, cluster es un grupo de múltiples computadoras unidas mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único ordenador, más potente que los comunes de escritorio.

- De un cluster se espera que presente combinaciones de los siguientes servicios:
  - Alto rendimiento (*High Performance*)
  - Alta disponibilidad (*High Availability)*
  - Equilibrio de carga (*Load Balancing*)
  - Escalabilidad (*Scalability*)

- La construcción de los ordenadores del cluster es más fácil y económica debido a su flexibilidad: pueden tener todos la misma configuración de hardware y sistema operativo (cluster homogéneo), diferente rendimiento pero con arquitecturas y sistemas operativos similares (cluster semi-homogéneo), o tener diferente hardware y sistema operativo (cluster heterogéneo).

- Para que un cluster funcione como tal, no basta sólo con conectar entre sí los ordenadores, sino que es necesario proveer un sistema de manejo del cluster, el cual se encargue de interactuar con el usuario y los procesos que corren en él para optimizar el funcionamiento

Componentes de un cluster

- En general, un cluster necesita de varios componentes de software y hardware para poder funcionar. A saber:
  - **Nodos** (los ordenadores o servidores)
  - **Conexiones de Red** (Ethernet, Fast Ethernet, Gigabit Ethernet, Myrinet, Infiniband, SCI,…)
  - **Middleware** (capa de abstracción entre el usuario y los sistemas operativos) genera la sensación al usuario de que utiliza un único ordenador muy potente (MOSIX, OpenMOSIX)
  - **Protocolos de Comunicación y servicios**.
  - **Aplicaciones** (pueden ser paralelas o no)

El middleware

- El middleware también debe poder migrar procesos entre servidores con distintas finalidades:
  - **balancear la carga**: si un servidor está muy cargado de procesos y otro está ocioso, pueden transferirse procesos a este último para liberar de carga al primero y optimizar el funcionamiento;
  - **Mantenimiento de servidores**: si hay procesos corriendo en un servidor que necesita mantenimiento o una actualización, es posible migrar los procesos a otro servidor y proceder a desconectar del cluster al primero
  - **priorización de trabajos:** en caso de tener varios procesos corriendo en el cluster, pero uno de ellos de mayor importancia que los demás, puede migrarse este proceso a los servidores que posean más o mejores recursos para acelerar su procesamiento.

Grid

- Grid computing or grid clusters are a technology closely related to cluster computing. The key differences (by definitions which distinguish the two at all) between grids and traditional clusters are that grids connect collections of computers which do not fully trust each other, or which are geographically dispersed. Grids are thus more like a computing utility than like a single computer. In addition, grids typically support more heterogeneous collections than are commonly supported in clusters.

- Grid computing is optimized for workloads which consist of many independent jobs or packets of work, which do not have to share data between the jobs during the computation process. Grids serve to manage the allocation of jobs to computers which will perform the work independently of the rest of the grid cluster. Resources such as storage may be shared by all the nodes, but intermediate results of one job do not affect other jobs in progress on other nodes of the grid.

- An example of a very large grid is the Folding@home project. It is analyzing data that is used by researchers to find cures for diseases such as Alzheimer's and cancer. Another large project is the SETI@home project, which may be the largest distributed cluster in existence. It uses approximately three million home computers all over the world to analyze data from the Arecibo Observatory radiotelescope, searching for evidence of extraterrestrial intelligence.

Supercomputadoras

- Una supercomputadora es una computadora que es considerada en su momento de introducción, como lo máximo en capacidad de procesamiento y cálculo, con capacidades muy superiores a la de una computadora de uso común.

- Una supercomputadora es un tipo de computadora muy potente y rápida, diseñada para procesar enormes cantidades de información en poco tiempo y dedicada a una tarea específica.

- La primera vez que apareció el término fue en 1929 en el periodico “New York World” refiriéndose a una computadora construida en la Universidad de Columbia.

Historia

- En 1960, Seymour Cray trabajaba en la CDC (Control Data Corporation) fue quien introdujo la primera súpercomputadora, fue líder durante toda la década hasta que en 1970 Cray formó su propia compañía, **Cray Research**

- Durante 5 años (1985-90), Cray lideréo el mercado de supercomputadoras con sus nuevos diseños.

- Actualmente el mercado de las supercomputadoras está dominado por las empresas IBM y HP que han absorvido a muchas de las empresas que figuraron en la década de los 80’s, con la intensión de ganar experiencia.

Seymour Cray

- En 1957, junto con otros ingenieros fundó una nueva compañía denominada Control Data Corporation, en abreviatura CDC, para la cual construyó el CDC 1604, que fue uno de los primeros ordenadores comerciales que utilizaron transistores en lugar de tubos de vacío.
- En 1963, el CDC 6600, batió ampliamente en capacidad de cálculo y en coste al ordenador más potente de que disponía IBM en aquella época.
- En el año 1972 fundó Cray Research, con el compromiso de dedicarse a construir exclusivamente supercomputadores y además de uno en uno, por encargo.
- CRAY-1 (1976) en el Laboratorio Nacional Los Álamos, incorporaba el primer ejemplo práctico en funcionamiento de procesador vectorial, junto con el procesador escalar más rápido del momento, con una capacidad de 1 millón de palabras de 64 bits y un ciclo de 12,5 nanosegundos. Su coste se situaba en torno a los 10 millones de dólares.

- El CRAY-2, entre 6 y 12 veces más rápido que su predecesor. Disponible en 1985, disponía de 256 millones de palabras y 240.000 chips. Su interior se encontraba inundado con líquido refrigerante.

- En el año 1986, existían en todo el mundo unos 130 sistemas de este tipo, de los cuales más de 90 llevaban la marca Cray.

- A mediados de los 80 controlaba el 70% del mercado de la supercomputación. Sin embargo Seymour Cray se encontraba incómodo, pues la problemática empresarial le resultaba escasamente interesante y difícil de soportar. Cedió la presidencia, y dejó la responsabilidad del desarrollo tecnológico de la línea CRAY-2 a Steve Chen, que concibió y construyó los primeros multiprocesadores de la firma, conocidos como serie **X-MP**.

Primeros prototipos de investigación

- Eran equipos que variaban desde arquitecturas SIMD, SIMD/MIMD o completamente MIMD. Los principios usados para la construcción de estos equipos fueron útiles para las siguientes generaciones.

Máquina	Control	Lugar y Fecha
Illiac IV	SIMD	Universidad de Illinois, 1968
MPP	SIMD	Goodyear AeroSpace, 1980
HEP	MIMD	Finales de los 70’s
PASM	SIMD/MIMD	Universidad Pardue, 1980
TRAC	SIMD/MIMD	U.T. Austin, 1977
NYU Ultra	MIMD	NYU, 1983
RP3	MIMD	IBM, 1983
Cosmic Cube	MIMD	Caltech, 1980

Tabla 1. Primeras computadoras paralelas

Primera generación

- Estas computadoras fueron en principio proyectos de empresas comerciales.

Máquina	Control	Lugar y Fecha
NCUBE-1	MIMD	NCUBE, 1988
GP1000	MIMD	BBN, 1985
Balance	MIMD	Sequent, 1985
FX/8	Propio	Alliant, 1985
iPSC/1	MIMD	Intel, 1985
SUPRENUM	MIMD	GMD FIRST, 1985
MP-1	SIMD	MasPar, 1985
CM-2	SIMD	TMC, 1985
GF11	SIMD	IBM, 1986

Tabla 2. Primera generación de computadoras paralelas

Segunda generación

- Gracias a la VLSI se pudieron construir computadoras cada vez más veloces y pequeñas.

Máquina	Control	Lugar y Fecha
AP1000	MIMD	Fujitsu, 1991
Cedar	MIMD	Universidad de Illinois
IPSC/2 iPSC/860	MIMD	Intel, 1988/89
NCUBE-2	MIMD	NCUBE, 1992
CM-5	MIMD	TMC 1992
TC2000	MIMD	BBN, 1989
Symmetry	MIMD	Sequent, 1990
FX/2800	MIMD	Alliant, 1990
MP-2	SIMD	MasPar, 1992
KSR1	MIMD	Kendall Square 1989

Tabla 3. Segunda generación de computadoras paralelas

Tercera generación

- Existe una variedad mayor de equipos cada vez más complejos y son cada vez más comerciales. Los sistemas con memoria compartida prácticamente han desaparecido para optar por la memoria distribuida, dispuesta para cada unidad de procesamiento.

Máquina	Procesador	Fabricante	MHz	# elementos de procesamiento	Memoria (MB)
T3D	Dec Alpha	Cray	150	128-2048	16-64
VPP500	Propio GAs, Vector	Fujitsu	100	4-222	128-256
SP1 (SP2)	RS/6000	IBM	62.5	8-64	64-256 (64M – 2GB)
Paragon XP/S	i860XP	Intel	50	4-2048	64-128
Cenju-3	NEC/MIPS CR4400SC	NEC	75	8-256	32-64
GC	Motorola Pwr PC 601	Parsytec	50	16-128	2-32
CS-2	Sparc	Meiko Limited	40	4-1024	32-128

Tabla 4. Tercera generación de computadoras paralelas

Las computadoras más rápidas del mundo

(www.top500.org)

Rank	Site	Computer	# CPU's	Year	R_{max}	R_{peak}
1	DOE/NNSA/LLNL United States	BlueGene/L - eServer Blue Gene Solution IBM	131,072	2005	280,600	367,000
2	Oak Ridge National Laboratory United States	Jaguar - Cray XT4/XT3 Cray Inc.	23,016	2006	101,700	119,350
3	NNSA/Sandia National Laboratories United States	Red Storm - Sandia/ Cray Red Storm, Opteron 2.4 GHz dual core Cray Inc.	26,544	2006	101,400	127,411
4	IBM Thomas J. Watson Research Center United States	BGW - eServer Blue Gene Solution IBM	40,960	2005	91,290	114,688
5	Stony Brook/BNL, New York Center for Computional Sciences United States	New York Blue - eServer Blue Gene Solution IBM	36,864	2007	82,161	103,219
6	DOE/NNSA/LLNL United States	ASC Purple - eServer pSeries p5 575 1.9 GHz IBM	12,208	2006	75,760	92,781
7	Rensselaer Polytechnic Institute, Computional Center for Nanotechnology Innovations United States	eServer Blue Gene Solution IBM	32,768	2007	73,032	91,750
8	NCSA United States	Abe - PowerEdge 1955, 2.33 GHz, Infiniband Dell	9,600	2007	62,680	89,587
9	Barcelona Supercomputing Center Spain	MareNostrum - BladeCenter JS21 Cluster, PPC 970, 2.3 GHz, Myrinet IBM	10,240	2006	62,630	94,208
10	Leibniz Rechenzentrum Germany	HLRB-II - Altix 4700 1.6 GHz SGI	9,728	2007	56,520	62,259.2

Tecnologías relevantes

- Hoy en día el diseño de Supercomputadoras se sustenta en 4 importantes tecnologías:
  - La tecnología de **registros vectoriales**, creada por Seymour Cray, considerado el padre de la Supercomputación, quien inventó y patentó diversas tecnologías que condujeron a la creación de máquinas de computación ultra-rápidas. Esta tecnología permite la ejecución de innumerables operaciones aritméticas en paralelo.
  - El sistema conocido como M.P.P. por las siglas de **Massively Parallel Processors** o Procesadores Masivamente Paralelos, que consiste en la utilización de cientos y a veces miles de microprocesadores estrechamente coordinados.

Tecnologías relevantes (cont)

- La tecnología de **computación distribuida**: los clusters de computadoras de uso general y relativo bajo costo, interconectados por redes locales de baja latencia y el gran ancho de banda.

- **Cuasi-Supercómputo**: Recientemente, con la popularización de la Internet, han surgido proyectos de computación distribuida en los que software especiales aprovechan el tiempo ocioso de miles de ordenadores personales para realizar grandes tareas por un bajo costo.

  A diferencia de las tres últimas categorías, el software que corre en estas plataformas debe ser capaz de dividir las tareas en bloques de cálculo independientes que no se ensamblaran ni comunicarán por varias horas. En esta categoría destacan BOINC y Folding@home, **SETI@home**.

Earth Simulator

- Desarrollado por las agencias japonesas NASDA, JAERI y JAMSTEC y en operación desde finales del año 2001, para aplicaciones de carácter científico siendo utilizado principalmente en simulaciones climáticas y de convección en el interior terrestre.

- Hasta finales del año 2003, ostentó el título de superordenador más rápido del mundo, con una capacitad computacional de más de 35 Teraflops.

  - 5120 CPUs especiales de 500 MHz fabricados por NEC Corporation
  - 640 nodos, con 8 procesadores cada uno
  - 8 GFLOPS por CPU (41 TFLOPS total)
  - 2 GB (4 módulos de 512 MB FPLRAM) por CPU (10 TB total)
  - memoria compartida en cada nodo
  - Switch crossbar 640 × 640 entre los nodos
  - Ancho de banda de 16 GB/s entre los nodos
  - Consumo de energía de 20 kVA por nodo
  - Sistema operativo Super-UX, basado en Unix

MareNostrum

- El superordenador más potente de Europa y el noveno en todo el mundo. Cuando fue puesto en marcha en 2005, el superordenador constaba de 2406 nodos de computación, cada uno de los cuales cuenta con procesadores duales IBM PowerPC 970FX de 64 bits a una velocidad de reloj de 2.2 GHz, 4812 CPUs en total. Actualmente cuenta con con **10,240 procesadores.**

- Los nodos del ordenador se comunican entre sí a través de una red Myrinet de gran ancho de banda y baja latencia. El sistema cuenta con **20 terabytes de memoria central, 280 terabytes de disco**. Utiliza el sistema operativo **Suse Linux**. Capacidad de cálculo de 62,63 teraflops (94,208 teraflops pico). Ocupa una instalación de 160 m² y pesa 40.000 kg.

- El MareNostrum será utilizado en la investigación del genoma humano, la estructura de las proteínas y en el diseño de nuevos medicamentos.

- **Myrinet** es una red de interconexión de altas prestaciones. Desarrollado por Myricom. Una de sus principales características, además de su rendimiento, es que el procesamiento de las comunicaciones de red se hace a través de chips integrados en las tarjetas de red, descargando a la CPU de parte del procesamiento de las comunicaciones.

Kan Balam (super.unam.mx)

- El sistema **HP Cluster Platform 4000**, **"KanBalam"** es la supercomputadora paralela más poderosa de México y América Latina.
  - Capacidad de procesamiento de 7.113 Teraflops (7.113 billones de operaciones aritméticas por segundo).
  - Cuenta con 1,368 procesadores (core AMD Opteron de 2.6 GHz)
  - Memoria RAM total de 3,000 Gbytes y un sistema de almacenamiento masivo de 160 Terabytes.

Algunos términos importantes …

- **Superescalar:**
  - es el término utilizado para designar un tipo de arquitectura de procesador capaz de ejecutar más de una instrucción por ciclo de reloj. En la clasificación de Flynn, un procesador superescalar es un procesador de tipo MIMD (multiple instruction multiple data).
- **Supersegmentado**:
  - Una solución para alcanzar mayores prestaciones, consiste en que muchas etapas de la segmentación requieren tareas que requieren menos de la mitad del ciclo de reloj, entonces se dobla la velocidad de reloj interna lo que permite que se realice una mayor cantidad de tareas en un ciclo de reloj externo.
- ***Procesamiento paralelo:*** un programa único que simultáneamente se ejecuta en múltiples procesadores.
- **Paralelismo** : es el proceso de realizar tareas concurrentemente
- **Paradigma**: es un modelo del mundo que se usa para formular una solución computacional a un problema