

MPI

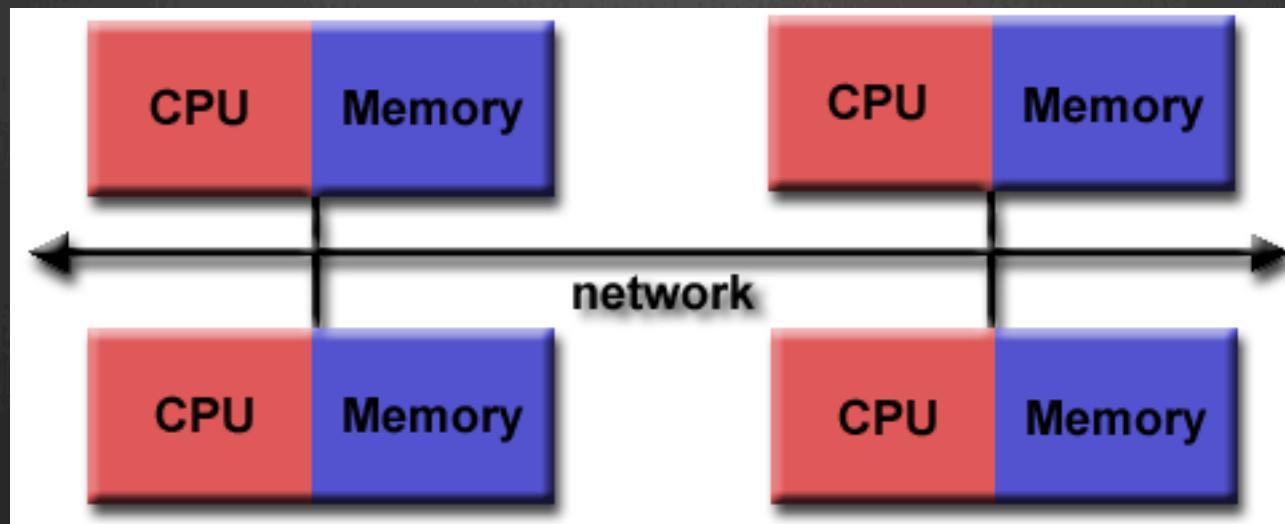
Dra. María Guadalupe Sánchez Cervantes

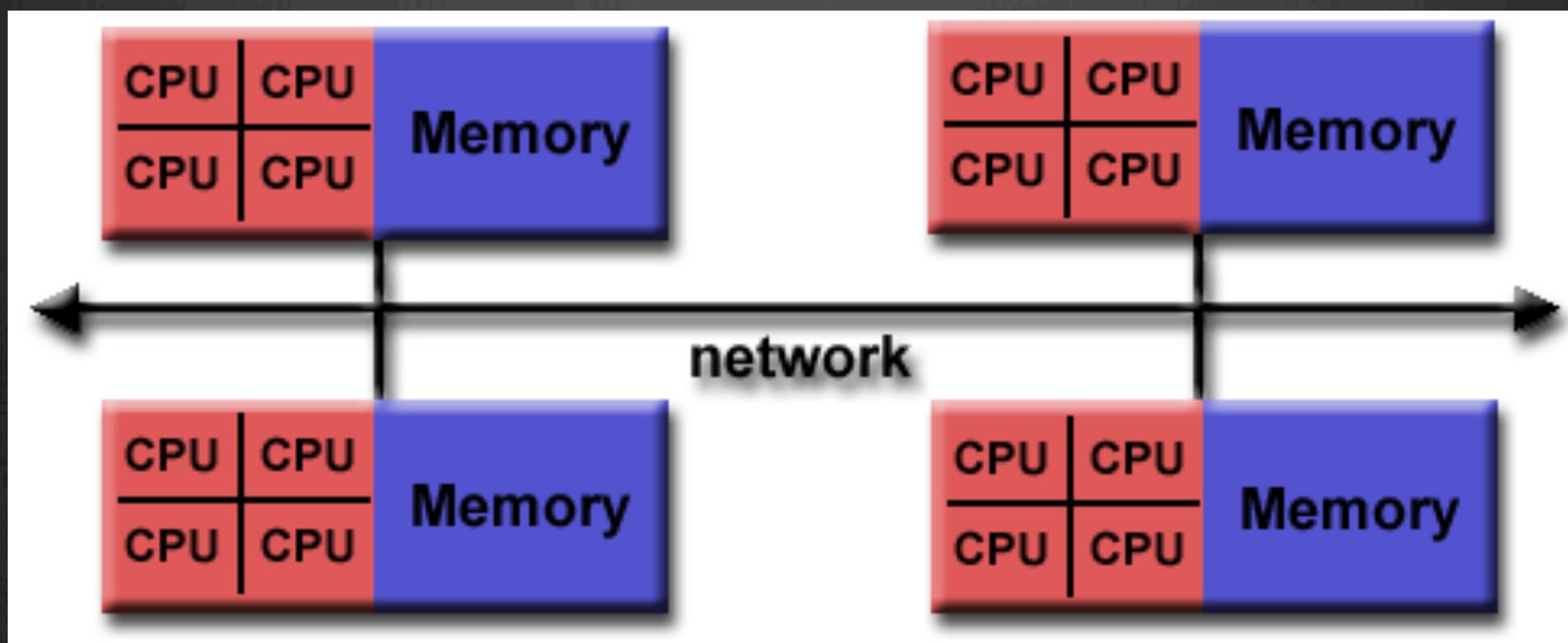
¿Qué es MPI?

- ⦿ **M P I = Message Passing Interface.**
- ⦿ MPI is a *specification* for the developers and users of message passing libraries.
- ⦿ MPI primarily addresses the message-passing parallel programming model: data is moved from the address space of one process to that of another process through cooperative operations on each process.

Modelo de Programación

- Arquitectura de memoria distribuida. 1980-1990





Razones para utilizar MPI

- Standardization
- Portability
- Performance Opportunities
- Functionality
- Availability

Como compilar archivos mpi

- mpicc –o <ejecutable> <fuente>

Iniciar el ambiente MPI:

```
MPI_Init (int *argc, char ***argv);
```

Finalizar el ambiente MPI:

```
MPI_Finalize();
```

Obtener el número de procesos. Regresa el tamaño de un comunicador.

```
MPI_Comm_size(MPI_Comm communicator, int* size)
```

MPI_COMM_WORLD → Encierra todos los procesos en el trabajo, esta llamada regresa la cantidad de procesos que fueron solicitados por el trabajo.

Obtener la posición en los procesos. Regresa el rango/la posición de un proceso en un comunicador. Cada procesos dentro de un comunicador es asignado una posición incremental comenzando desde cero. La posición se usa principalmente con el fin de identificar quien envía y quien recibe.

```
MPI_Comm_rank(MPI_Comm communicator, int* rank);
```

Obtener el nombre actual del procesador en quien el proceso se está ejecutando.

```
MPI_Get_processor_name(char * name, int* name_length);
```

```
Char processor_name[MPI_MAX_PROCESSOR_NAME]
```

MPI send and receive

- Enviar y recibir son dos conceptos fundamentales de MPI.

```
MPI_Send(void* data, int count, MPI_Datatype datatype, int destination,  
        int tag, MPI_Comm communicator)
```

```
MPI_Recv(void* data, int count, MPI_Datatype datatype, int source,  
        int tag, MPI_Comm communicator, MPI_Status* status)
```

Tipo de Datos elementales de MPI

MPI datatype	C equivalent
MPI_CHAR	char
MPI_SHORT	short int
MPI_INT	int
MPI_LONG	long int
MPI_LONG_LONG	long long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	char

Ejemplo 1

```
#include "mpi.h"

#include <stdio.h>

int main( argc, argv )

int argc;
char **argv;

{
MPI_Init( &argc, &argv );
printf( "Hello world\n" );
MPI_Finalize();
return 0;
}
```

Ejemplo 2

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

int main(int argc, char** argv) {
    MPI_Init(NULL, NULL);

    // Get the number of processes //procesos
    int world_size1;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size1);
```

```
// Get the rank of the process //

int world_rank1;

MPI_Comm_rank(MPI_COMM_WORLD, &world_rank1);

// Get the name of the processor

char processor_name1[MPI_MAX_PROCESSOR_NAME];

int name_len;

MPI_Get_processor_name(processor_name1, &name_len);

// Print off a hello world message

printf("Hola Mundo!!! desde el procesador %s, rank %d out of %d processors\n",

processor_name1, world_rank1, world_size1);

// Finalize the MPI environment. No more MPI calls can be made after this

MPI_Finalize();

}
```

Ejemplo 3

```
int main(int argc, char** argv) {  
    // Initialize the MPI environment  
    MPI_Init(NULL, NULL);  
    // Find out rank, size  
    int world_rank;  
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);  
    int world_size;  
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
```

```
// We are assuming at least 2 processes for this task  
if (world_size < 2) {  
    fprintf(stderr, "World size must be greater than 1 for %s\n",  
            argv[0]);  
    MPI_Abort(MPI_COMM_WORLD, 1);  
}
```

```
int number;

if (world_rank == 0) {

    // If we are rank 0, set the number to -1 and send it to process 1

    number = -1;

    MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);

} else if (world_rank == 1) {

    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

    printf("Process 1 received number %d from process 0\n", number);

}

MPI_Finalize();

}
```

Tarea

- ⦿ Funciones de envío y recepción síncrono y asíncrono.
- ⦿ Cuál es la diferencia