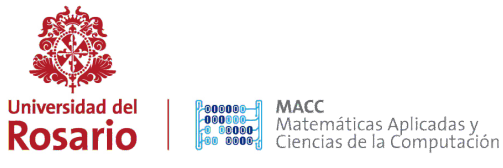
Paralelización con memoria distribuida

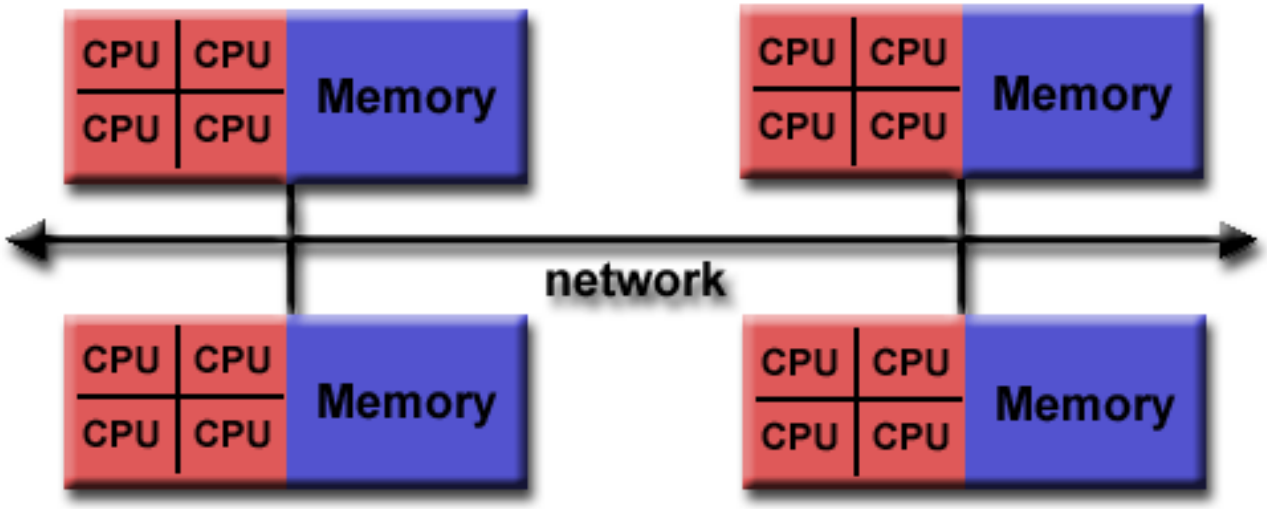
Carlos E. Alvarez[1].

[1]Dep. de Matemáticas aplicadas y Ciencias de la Computación, Universidad del Rosario

2020-I



Memoria compartida



Varios *nodos* conectados por medio de una *red*.

Memoria compartida

- Cada nodo tiene su memoria
- Envío de *mensajes* a través de la red
- La *red* tiene una capacidad finita de transmisión (mas lenta que el acceso a la memoria compartida)

Procesos e hilos

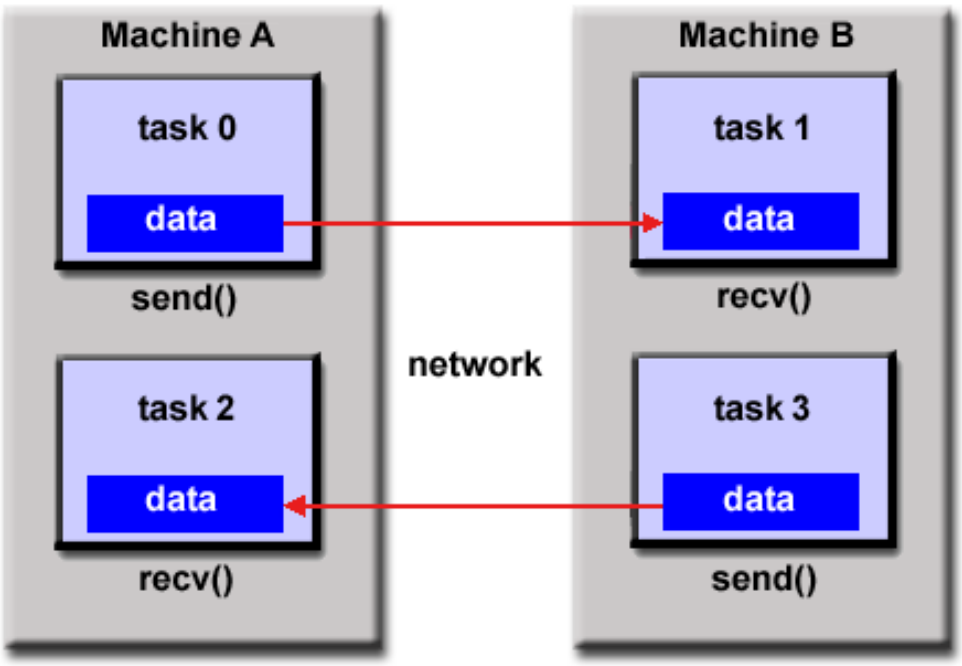
- **Proceso:** Provee los recursos para ejecutar un programa (Tabla de direcciones de memoria, código ejecutable, variables de ambiente, etc.). Posee un identificador único de proceso
- **Hilo:** Entidad dentro de un proceso que ejecuta una porción de código. Comparte recursos con otros hilos dentro del mismo proceso. Posee un identificador único de hilo dentro del proceso

Modelo de paso de mensajes

Single Instruction Multiple Data (SIMD)

- Proceso: Instancia de un programa junto con sus datos
- Muchos procesos, una taréa
- Cada proceso tiene acceso a sus datos propios
- Los procesos se comunican por medio de *mensajes*
- Usualmente se corre un solo proceso por nodox

Modelo de paso de mensajes



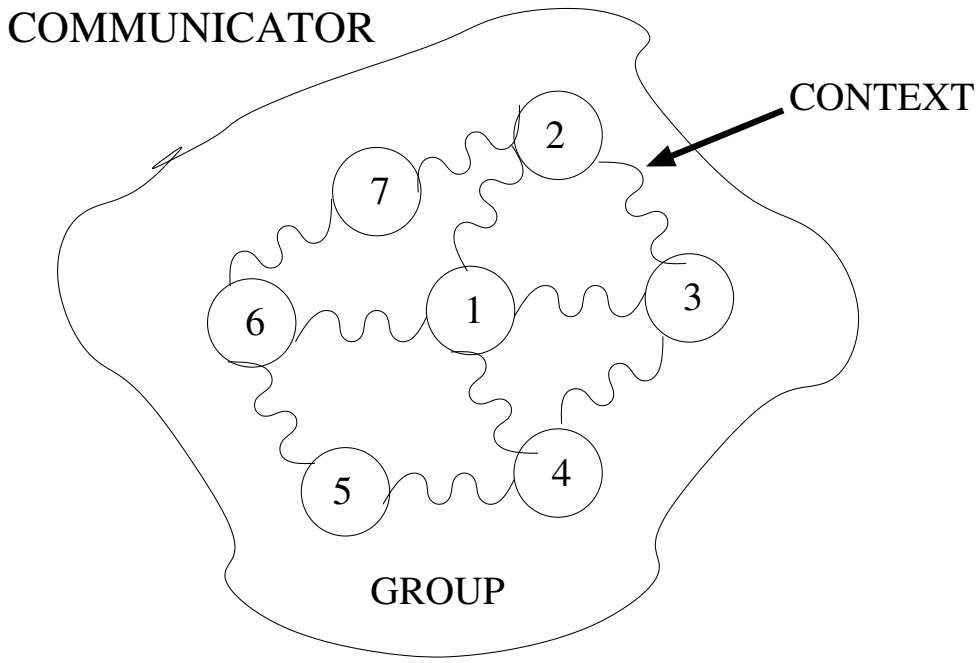
Paso de mensajes.

Componentes de un mensaje

- ID del proceso que envía el mensaje (*source*)
- ID del proceso al que está destinado el mensaje (*destination*)
- Tipo de datos que se envían (*datatype*)
- Cantidad de datos que se envía (*count*)
- Apuntador a los datos que se envían (*address*)
- ID del mensaje (*tag*)

Comunicadores

Comunicador: Objeto que agrupa un conjunto de procesos. Todas las comunicaciones, punto a punto o collectivas suceden entre los miembros del grupo.



Comunicador.

Primer programa

```
#include <cstdio>
#include <unistd.h>
#include <sched.h>
#include <mpi.h>

int main(int argc, char* argv[]){
  int rank ,size;
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  printf("I'm MPI process %d of %d. I'm at core %d.\n",
  rank,size,sched_getcpu());
  sleep(1);

  MPI_Finalize();
  return 0;
}
```

Compilación

- Para poder compilar programas con MPI debe instalarse alguna distribución de la API (por ejemplo openMPI) que viene con su compilador (`mpic++` para C++)
- Debe incluirse la librería de MPI: `#include <mpi.h>`
- Ejemplo:

```
1 mpic++ mpi_test.cpp -o mpi_test
```

Ejecución

- Se usa el comando `mpiexec`
- La opción `-np` permite definir el número de procesos a lanzar
- Ejemplo:

```
1 mpiexec -np 4 mpi_test
```

La región paralela de un programa MPI se inicia con el llamado

```
MPI_Init(&argc,&argv);
```

y se finaliza con el llamado

```
MPI_Finalize();
```

- `MPI_COMM_WORLD` es el comunicador por defecto
- `MPI_Comm_rank(comm, &rank)` guarda en `rank` el id del proceso dentro del grupo `comm`
- `MPI_Comm_size(comm, &size)` guarda en `size` el número de procesos dentro del grupo `comm`