



OPENMPI

MEMORIA DISTRIBUIDA

OPENMPI

- Modelo de programación

Biblioteca de pasaje de mensajes

Para utilizar en C, se debe incluir

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

```
MPI_Init(&argc, &argv); // Inicializa, se crean copias del mismo código para cada proceso
```

```
MPI_Finalize(); //Para finalizar
```

OPENMPI

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

int main(int argc, char* argv[]) {
    int rank, size; // numero de proceso y cantidad total de procesos

    MPI_Init(&argc, &argv); //Inicializa
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); //Consulta el numero de proceso
    MPI_Comm_size(MPI_COMM_WORLD, &size); // Cantidad de procesos

    if (rank == 0) { //Proceso 0, se le puede pedir tareas iniciales
        printf("Número total de procesos: %d\n", size);
    }

    printf("Hola desde el proceso %d\n", rank);
    MPI_Finalize();
    return 0;
}
```

OPENMPI

- Compilar

```
mpicc hola-mpi.c -o hola-mpi
```

- Ejecutar

```
srun --mpi=pmix -n X ./hola-mpi
```

OPENMPI

```
#!/bin/bash
```

```
#SBATCH --nodes=4
```

```
#SBATCH --ntasks-per-node=4
```

```
#SBATCH --output=mpi-out.%j
```

```
#SBATCH --error=mpi-err.%j
```

```
#SBATCH --partition=cronos
```

```
cd $SLURM_SUBMIT_DIR
```

```
mpirun nombre
```

```
NOMBRAR mpi-job.sh
```

```
EJECUTAR sbatch ./mpi-job.sh
```



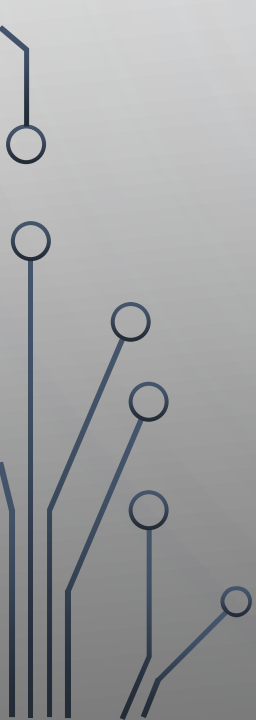

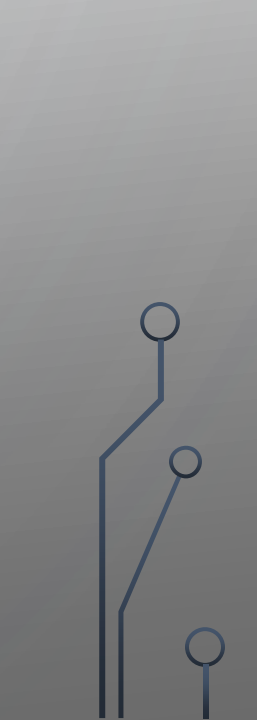
OPENMPI

MPI ofrece una funciones para comunicar todos los procesos que intervienen.

Las comunicaciones colectivas, facilitan la programación evitando posibles errores.

Son mas eficientes que las comunicaciones punto a punto.

Las funciones básicas disponibles son:

- Barrier
 - Broadcast
 - Reduce
- 
- 
- 

OPENMPI

Barrier

```
int MPI_Barrier(MPI_Comm comm)
```

Todos los procesos esperan que todos lleguen a la barrera, para continuar la ejecución. Sincroniza un grupo de procesos.

Broadcast

```
int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
```

Broadcast (comunicación uno a todos), se envían count datos del tipo datatype desde el proceso raíz (root) al resto de los procesos en el comunicador.

Reduce

```
int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
```

Lleva a cabo una reducción de todos a uno, es decir, se recogen los valores distribuidos en los diferentes procesos y se les aplica una operación a todos. Esto significa que los datos sobre los que hay que hacer la reducción se toman de la zona apuntada por sendbuf, y el resultado se deja en la apuntada por recvbuf.

OPENMPI

Programa para sumar números entre dos valores, left y right

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

```
int main(int argc, char **argv)
```

```
{ int left, right;
```

```
    int number, sum; //usar number como indice del for
```

```
    //código,
```

```
    printf("Suma : %d", sum);
```

```
    return 0;
```

```
}
```


OPENMPI

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv)
{ int left, right;
  int number, sum;
  int rank, size, root;
  int interval, mystart, myend, GrandTotal;

  MPI_Init(&argc, &argv);

  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
```

OPENMPI

```
root=0;  
left=1;  
right=1000;  
if (rank==root)  
{  
  
}
```

OPENMPI

```
MPI_Bcast(&left,1,MPI_INT,root,MPI_COMM_WORLD);  
MPI_Bcast(&right,1,MPI_INT,root,MPI_COMM_WORLD);
```

```
//Calculo limites
```

```
if (((right-left+1) % size) !=0)  
    interval=(right-left+1) / size +1;  
else  
    interval=(right-left+1)/size
```

OPENMPI

```
//límites de cada proceso  
mystar=left + rank*interval;  
myend=mystart+interval;
```

```
if (mystart <= right)  
    //código
```

```
printf("Proceso: %d Suma acumulada: %d Suma del proceso:  
%d\n",rank,sum,number);
```

OPENMPI

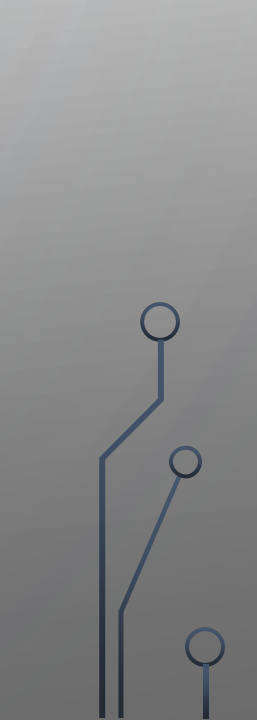

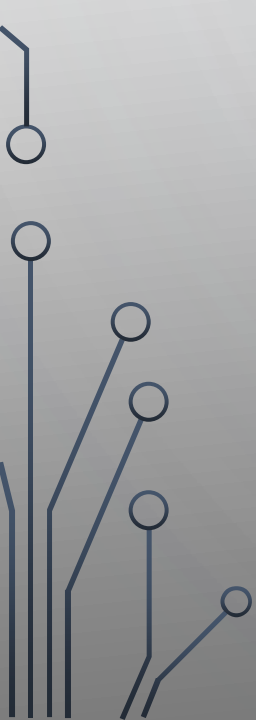
```
MPI_Reduce(&sum,&GrandTotal,1,MPI_INT,MPI_SUM,root,MPI_COMM_
WORLD);
MPI_Barrier(MPI_COMM_WORLD);
if (rank==root)
    printf("Total: %d\n",GrandTotal);
MPI_Finalize();
return 0;
}
```



OPENMPI

```
int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
```

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



OPENMPI

```
#include <mpi.h>
#include <stdio.h>
int main(int argc,char **argv){
int rank,size;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);

printf("Hola mundo, Este es el numero de proceso %d de %d\n",rank,size);

if(rank==1){
    double data=1.5;
    double data1=2.5;
    MPI_Send(&data,1,MPI_DOUBLE,0,5,MPI_COMM_WORLD);
    MPI_Send(&data1,1,MPI_DOUBLE,2,6,MPI_COMM_WORLD);
}
if(rank==0){
    double data;
    MPI_Recv(&data,1,MPI_DOUBLE,1,MPI_ANY_TAG,
    MPI_COMM_WORLD,MPI_STATUS_IGNORE);
    printf("El proceso 1 dice %g\n",data);
}
if (rank==2)
{
    double data1;
    MPI_Recv(&data1,1,MPI_DOUBLE,1,6,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
    printf("El proceso 1 dice %g\n",data1);
}
MPI_Finalize();
return 0;
}
```