Tutorial sobre MPI

Introducción

Autor: Abel Francisco Paz Gallardo
Evento: Tutorial MPI
Lugar / Fecha: Trujillo, 11 y 18 de Mayo 2010

GOBIERNO DE ESPAÑA
MINISTERIO DE CIENCIA E INNOVACIÓN
CENTRO EXTREMEÑO DE TECNOLOGÍAS AVANZADAS
CETA Ciemat
FEDER
Fondo Europeo de Desarrollo Regional
Una manera de hacer Europa

INDICE

GOBIERNO DE ESPAÑA
MINISTERIO DE CIENCIA E INNOVACIÓN
CENTRO EXTREMEÑO DE TECNOLOGÍAS AVANZADAS
CETA Ciemat
FEDER
Fondo Europeo de Desarrollo Regional
Una manera de hacer Europa

¿Qué es MPI?

- MPI es una **interfaz** de **paso de mensaje** que representa un esfuerzo prometedor de mejorar la disponibilidad de un software **altamente eficiente y portable** para satisfacer las necesidades actuales en la **computación de alto rendimiento** a través de la definición de un estándar de paso de mensajes **universal**.

William D. Gropp et al.

- **Protocolo** de comunicación entre máquinas
- **Estándar** para la comunicación entre los nodos que ejecutan un programa en un sistema de memoria distribuida
- **Conjunto de librerías** de funciones que pueden ser utilizadas en programas escritos en C, C++, Fortran y Ada

Ventajas de utilizar MPI

- **Portabilidad:** MPI ha sido implementado para casi toda arquitectura de memoria distribuida
- **Velocidad**: Cada implementación de la biblioteca ha sido optimizada para el hardware en la cual se ejecuta
  - *Ejemplo*: Clusters de Bull utilizan MPIBull2 optimizada para IB
- **Estándar**: 60 personas de 40 organizaciones diferentes principalmente de U.S.A. y Europa trabajaron en estandarizarlo (IBM, INTEL, NX*/, Express, nCUBE's Vernex, p4, PARMACS, Zipcode, Chimp, PVM, Chameleon, PICL…)*
- **Amplia funcionalidad**
- Gran variedad de **implementaciones libres** (*MPICH, OpenMPI, LAM-MPI…*)

Implementaciones de MPI ("MPI flavors/flavours")

- LAM/MPI (Local Area Multicomputer).
- **OpenMPI**
- **MPICH:** MPICH2 (estándar MPI 2.1)

- Diferentes ramas en cada implementación:
  - Para GRID: MPICH-G2, MPICH-GM, MPICH-VMI2…
  - Para Infiniband: MVAPICH, MPIBull2…

- Otros lenguajes: Python (PyMPI), Java (mpiJava), …
- Más info sobre el estándar MPI 2.0: http://www.mcs.anl.gov/research/projects/mpi/mpi-standard/mpi-report-2.0/mpi2-report.htm

Nociones básicas

- La **unidad básica** en MPI son los **procesos**.
- Tienen espacios de memoria independientes.
- El Intercambio de información es mediante **paso de mensajes**.

- Introduce en concepto de **comunicadores** (grupo de procesos más contexto).
- A **cada proceso** se le asigna **un identificador interno** propio de MPI (rank).

Primer programa en MPI

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

int main(int argc, char *argv[ ]) {
        int myrank, size;

        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
        MPI_Comm_size(MPI_COMM_WORLD, &size);

        printf(“Hola soy el proceso %d de %d\n”, myrank, size);

        MPI_Finalize();
        exit(0);
}
```

¿Qué son los comunicadores?

- MPI **agrupa los procesos** implicados en una ejecución paralela en **comunicadores**.
- Un comunicador **agrupa a procesos que pueden intercambiarse mensajes**.
- El comunicador **MPI_COMM_WORLD** está creado por defecto y engloba a todos los procesos.
- Pueden definirse varios comunicadores.
  - *MPI_Comm_create, MPI_Comm_free, MPI_Comm_group, MPI_Group_include, MPI_Group_exclude…*

Inicio y Finalización

- **int MPI_Init(int *argc, char **argv)**
  - Primera llamada **OBLIGATORIA** de cada uno de los procesos MPI
  - Establece entorno
  - Un proceso solo puede hacer una llamada MPI_INIT

- **int MPI_Finalize(void)**
  - Termina la ejecución en MPI
  - Libera los recursos utilizados por MPI

Identificación de procesos

- **MPI_Comm_rank (comm, &identificador);**
  - Devuelve el **identificador** del proceso dentro del comunicador **comm** especificado (ej.: MPI_COMM_WORLD).

- **MPI_Comm_size (comm, &num_proc);**
  - Devuelve en **num_proc** el número de procesos del comunicador especificado.

Primer programa en MPI (segundo vistazo)

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

int main(int argc, char *argv[ ]) {
        int myrank, size;

        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
        MPI_Comm_size(MPI_COMM_WORLD, &size);

        printf(“Hola soy el proceso %d de %d\n”, myrank, size);

        MPI_Finalize();
        exit(0);
}
```

Tipos de datos

- **Todo tipo de datos:**
  - **MPI_CHAR** signed char
  - **MPI_SHORT** signed short int
  - **MPI_INT** signed int
  - **MPI_LONG** signed long int
  - **MPI_UNSIGNED_CHAR** unsigned char
  - **MPI_UNSIGNED_SHOT** unsigned short int
  - **MPI_UNSIGNED** unsigned int
  - **MPI_UNSIGNED_LONG** unsigned long int
  - **MPI_FLOAT** float
  - **MPI_DOUBLE** double
  - **MPI_LONG_DOUBLE** long double
  - …

  - MPI también permite definir **tipos de datos propios** (MPI_TYPE_VECTOR, MPI_TYPE_STRUCT…)

Funciones de MPI

- **MPI 1.2** tiene **129 funciones**
- Las **funciones principales** de MPI son:
  - MPI_Init
  - MPI_Finalize
  - MPI_Comm_size
  - MPI_Comm_rank
  - MPI_Send
  - MPI_Recv

Tipos de mensajes (1)

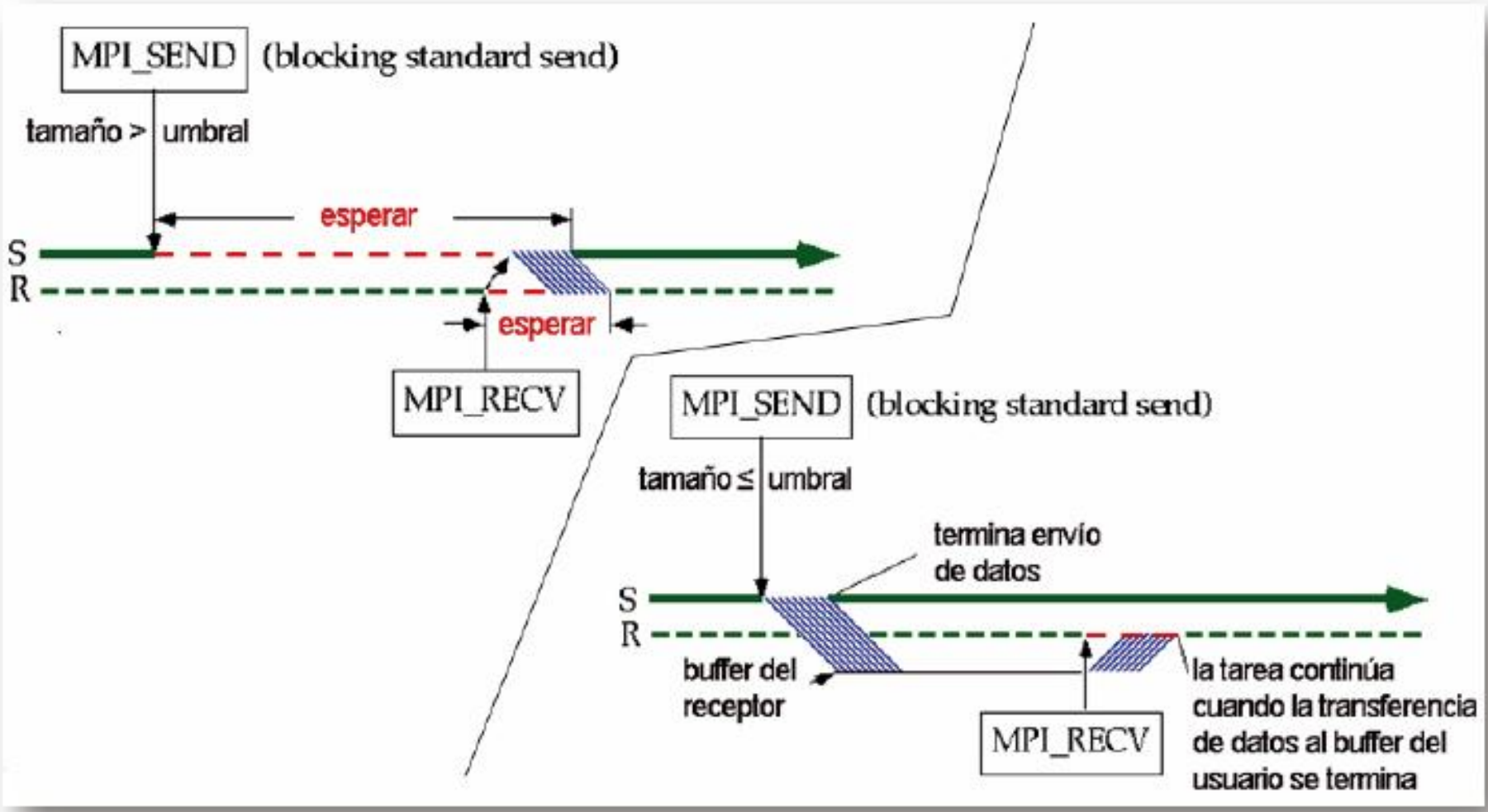
- **Punto a punto**: 1 emisor - 1 receptor
  - **Síncronos**: El emisor **se espera** a que la comunicación se produzca (o que el buffer del receptor este disponible). ***BLOQUEANTE***
    - *Send, Recv, SendRecv, Bsend, Ssend*, …
  - **Asíncronos**: El emisor **NO se espera** a que la comunicación se produzca y se comprueba más tarde si se ha producido. ***NO BLOQUEANTES***
    - *Isend, Irecv*, …

Tipos de mensajes (2)

- **MPI_Send**(buf, count, datatype, dest, tag, comm)
- **MPI_Recv**(buf, count, datatype, source, tag, comm, status)

- **buf**: Dirección donde comienza el mensaje.
- **count**: número de elementos del mensaje.
- **datatype**: tipo del mensaje.
- **dest/source**: posición relativa del proceso fuente/destino dentro del comunicador.
  - MPI_ANY_SOURCE: permite recibir mensaje de cualquier fuente
- **status:** estado de la recepción

Tipos de mensajes (3)



Tipos de mensajes (4)

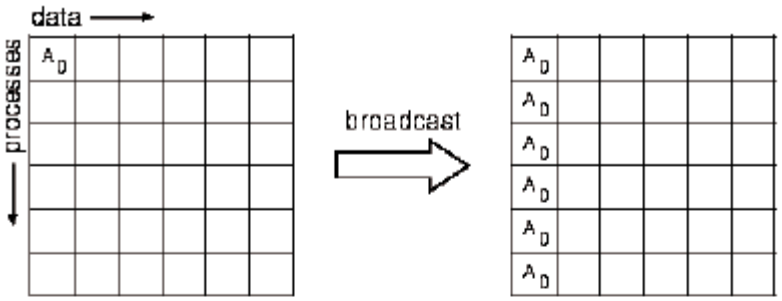
- **Variantes** de **MPI_Send** para **casos especiales**
- **Síncrono:** MPI_Ssend**(.....)**
  - La operación se da por terminada sólo **cuando el mensaje ha sido recibido en destino**. (MPI_Send termina cuando encuentra y receptor y envía su mensaje)
- **Buffered:** MPI_Bsend(.....)
  - Cuando se hace un envío con buffer **se guarda inmediatamente, en un buffer al efecto en el emisor**, una copia del mensaje. La operación se da por completa en cuanto se ha efectuado esta copia. Si no hay espacio en el buffer, el envío fracasa.
- **Ready**: MPI_Rsend(.....)
  - Sólo se puede hacer si antes **el otro extremo está preparado para una recepción** inmediata. No hay copias adicionales del mensaje (como en el caso del modo con buffer), y tampoco podemos confiar en bloquearnos hasta que el receptor esté preparado.

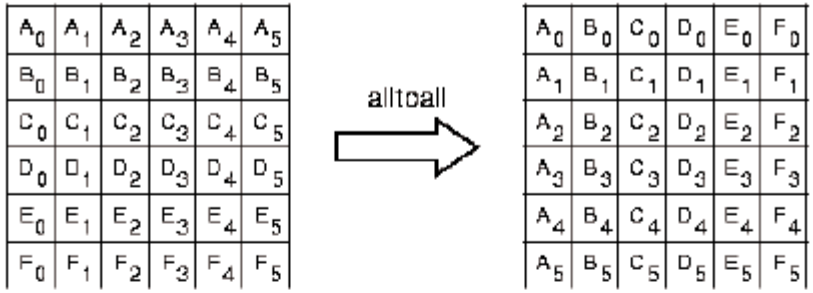
(para más info mirar referencias al final)

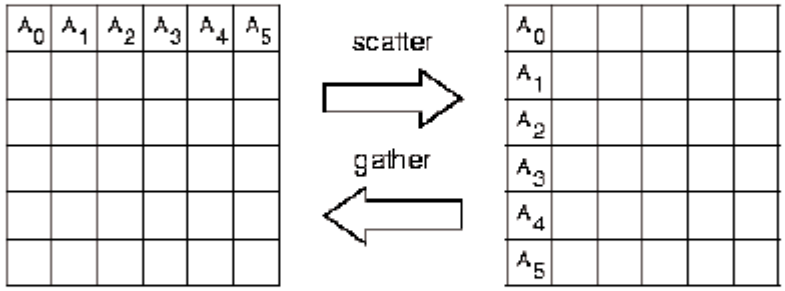
Tipos de mensajes (5)

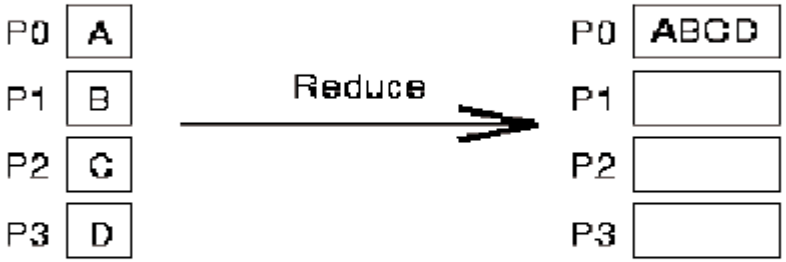
- **Multipunto**: 1 o N emisores - 1 o N receptores
  - MPI_Barrier( ) - Sincronización global entre los procesadores del comunicador.
  - MPI_Bcast( )
  - MPI_Gather( )
  - MPI_Scatter( )
  - MPI_Alltoall( )
  - MPI_Reduce( )
  - MPI_Reduce_scatter( )
  - MPI_Scan( ) …

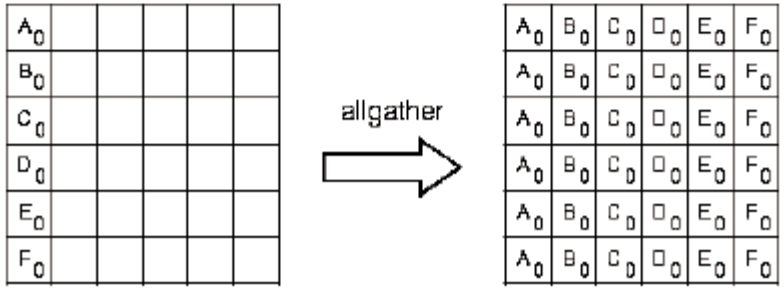
Funciones de MPI

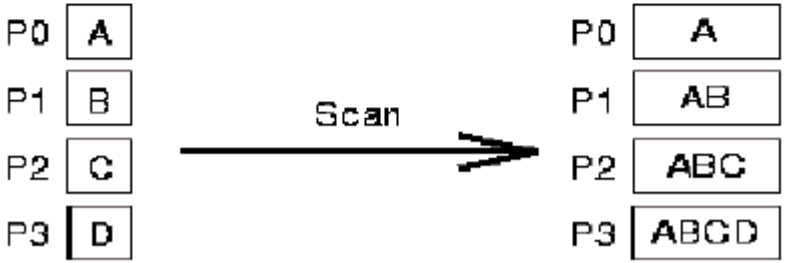












Tipos de reducción

- **MPI_MAX** maximum
- **MPI_MIN** minimum
- **MPI_SUM** sum
- **MPI_PROD** product
- **MPI_LAND** logical and
- **MPI_BAND** bit-wise and
- **MPI_LOR** logical or
- **MPI_BOR** bit-wise or
- **MPI_LXOR** logical xor
- **MPI_BXOR** bit-wise xor
- **MPI_MAXLOC** max value and location
- **MPI_MINLOC** min value and location

Caso práctico 1

- Crear un programa sencillo en C utilizando MPI, en el que un nodo (maestro) salude a otros nodos (esclavos).
- Condiciones:
  - El saludo tiene que realizarlo antes el maestro que los esclavos.
  - El maestro tiene que enviar un mensaje que muestren por pantalla los nodos esclavos.

```
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 )
```

Caso práctico 1

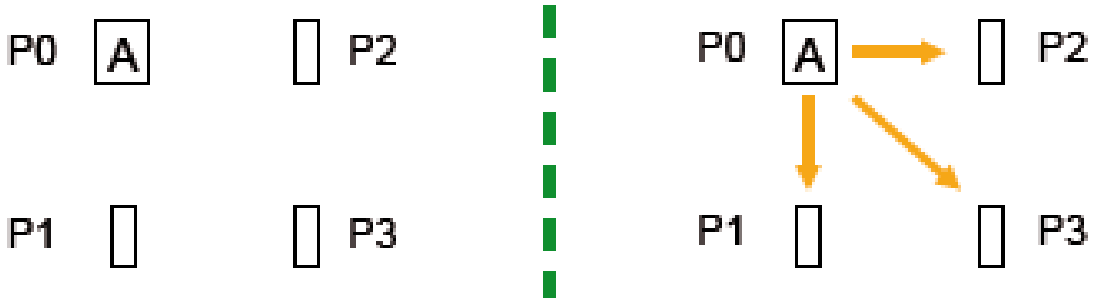
```
#include <stdio.h>
#include <mpi.h>
#include <unistd.h>

int main(int argc, char** argv) {
  char host[20], Recv[20];
  MPI_Status status;
  int rank, size, tag=0, i;
  gethostname(host,20);
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  if (rank == 0) {      /* proc 0 => send message */
        for (i = 1; i < size; ++i) {
        MPI_Send("Hola desde P0 !", 20, MPI_CHAR, i , tag, MPI_COMM_WORLD);
        printf ("Hola, soy el nodo % de %i nodos. Estoy en la máquina %s y he enviado un mensaje a %i\n",rank,size, host,i);
        }
  }
  if (rank != 0) {      /* proc 1 => receive message */
        MPI_Recv(Recv, 20, MPI_CHAR, 0, tag, MPI_COMM_WORLD,&status);
        printf("Hola! Soy el nodo %i de un total de %i nodos. Estoy en la máquina %s y he recibido el mensaje: '%s'\n",
              rank,size,host,Recv);
  }
  MPI_Finalize();
}
```

Caso práctico 2

- Crear un programa sencillo que utilice la función Broadcast para que el nodo maestro mande un entero al resto de nodos.



Int MPI_Bcast (void *buffer, int count, MPI_Datatype datatype, source, MPI_Comm comm)

Caso práctico 2

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

int main(int argc, char** argv) {
  int rank;
  int n;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  if (rank == 0) {      /* proc 0 => send message */
        printf("Hola, soy el nodo maestro, vamos a mandarle el numero 2010 al todos los nodos. ");
        n = 2010;
  }

  MPI_Bcast(&n,1,MPI_INT,0,MPI_COMM_WORLD);

  printf("Hola, soy el proceso %d y el numero es el %d\n",rank,n);

  MPI_Finalize();
}
```

Enviar trabajos al cluster de GPUs

- Acceder como usuario **tutorial**

  ssh tutorial@ce-gpu-test.ceta-ciemat.es

- Crear carpeta de usuario en /home/tutorial/**minombredeusuario**
- Enviar un trabajo:

  qsub -q gpu -pe gpupe <numero_nodos> script.sh

- Script.sh

```
export MPICH2_ROOT=/opt/mpi/mpibull2-1.3.9-14.s
export PATH=$MPICH2_ROOT/bin:$PATH
export MPD_CON_EXT="sge_$JOB_ID.$SGE_TASK_ID"

echo "Got $NSLOTS slots."

mpiexec -machinefile $TMPDIR/machines -n $NSLOTS /opt/exp_soft/tutorial/mpisend

exit 0
```

Tiene que ser accesible por todos los nodos

Referencias

- **MPI: Message Passing Interface**. Universidad Carlos III Madrid. http://svn.ceta-ciemat.es/gpus/trunk/Bibliograf%c3%ada/MPI/Curso-MPI_CarlosIIIMadrid.pdf
- **Interfaz de Paso de Mensajes** – Wikipedia**.** http://es.wikipedia.org/wiki/Interfaz_de_Paso_de_Mensajes
- **Message Passing Interface (MPI)** - Blaise Barney, Lawrence Livermore National Laboratory**.** https://computing.llnl.gov/tutorials/mpi/
- **MPI: The Complete Reference -** http://www.netlib.org/utk/papers/mpi-book/mpi-book.html
- **MPI Routines (official)**: **http://www.mcs.anl.gov/research/projects/mpi/www/**

Conventual de San Francisco, Sola 1, 10200 Trujillo
Teléfono: 927 65 93 17 Fax: 927 32 32 37
www.ceta-ciemat.es

GOBIERNO DE ESPAÑA

MINISTERIO DE CIENCIA E INNOVACIÓN

CENTRO EXTREMEÑO DE TECNOLOGÍAS AVANZADAS

CETA Ciemat

FEDER

Fondo Europeo de Desarrollo Regional

Una manera de hacer Europa

Cómo funciona MPI

- **SEGUNDA PARTE DEL TUTORIAL MPI**

- Mpich, demonios… MPD..