3. Comunicación en Clusters

Comunicación paralela

- Computación paralela: forma eficaz de proceso de la información que favorece la explotación de sucesos concurrentes en el proceso de computo
- Objetivo: Aumentar el rendimiento
- Paralelismo de datos: operaciones ejecutadas en paralelo sobre colecciones de datos estructurados
  - Replicas del mismo programa trabajando sobre partes distintas de los datos

Comunicación paralela

- Mecanismos habituales de programación paralela:
  - Programas que pueden ser paralelizados usando directivas de compilación como OpenMP
  - Programas que pueden ser escritos en lenguajes masivamente paralelos como CUDA
  - Programas que pueden ser paralelizados usando librerías de paso de mensajes

- El algoritmo debe ser naturalmente paralelo!

Paradigma de paso de mensajes

- El mas extendido hoy día en programación de aplicaciones paralelas
- Consiste en una serie de procesos que interactúan por medio de mensajes
  - Cada proceso puede ejecutar código distinto sobre diferentes datos
- El intercambio de mensajes es cooperativo: los datos deben ser tanto enviados como recibidos explicitamente

MPI

- Estandar para paso de mensajes
- No es un lenguaje, sino una librería que se añade a lenguajes ya existentes
- Colección de funciones que ocultan detalles de bajo nivel, tanto hardware como software
- Pretende que las aplicaciones puedan ser portables

MPI

- Implementaciones del estándar MPI
  - MPICH
  - LAM/MPI
  - **OpenMPI**

Tipos de datos MPI

MPI Datatype	C Type
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_SHORT	unsigned short int

Tipos de datos MPI

MPI Datatype	C Type
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	(none)

Funciones MPI

- Envío bloqueante

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

    IN    buf       initial address of send buffer
    IN    count     number of entries to send
    IN    datatype  datatype of each entry
    IN    dest      rank of destination
    IN    tag       message tag
    IN    comm      communicator
```

Funciones MPI

- Recepción bloqueante

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

OUT	buf	initial address of receive buff
IN	count	max # of entries to receive
IN	datatype	datatype of each entry
IN	source	rank of source
IN	tag	message tag
IN	comm	communicator
OUT	status	return status

Funciones MPI

- Envío no bloqueante

```
int MPI_Isend(void* buf, int count, MPI_Datatype
 datatype, int dest, int tag, MPI_Comm comm,
 MPI_Request *request);
```

IN	buf	initial address of send buffer
IN	count	number of entries to send
IN	datatype	datatype of each entry
IN	dest	rank of destination
IN	tag	message tag
IN	comm	communicator
OUT	**request**	**request handle**

Funciones MPI

- Recepción no bloqueante

```
int MPI_Irecv(void* buf, int count, MPI_Datatype
 datatype, int source, int tag, MPI_Comm comm,
 MPI_Request*request)
```

OUT	buf	initial address of receive buff
IN	count	max # of entries to receive
IN	datatype	datatype of each entry
IN	source	rank of source
IN	tag	message tag
IN	comm	communicator
OUT	**request**	**request handle**

Funciones MPI

- Recepción no bloqueante

```
int MPI_Irecv(void* buf, int count, MPI_Datatype
 datatype, int source, int tag, MPI_Comm comm,
 MPI_Request*request)
```

OUT	buf	initial address of receive buff
IN	count	max # of entries to receive
IN	datatype	datatype of each entry
IN	source	rank of source
IN	tag	message tag
IN	comm	communicator
OUT	**request**	**request handle**

Funciones MPI

- Envío y Recepción no bloqueantes

```
int MPI_Wait(MPI_Request *request, MPI_Status * status);
    INOUT request      request handle
    OUT   status       status object

int MPI_Test(MPI_Request *request, int *flag,
      MPI_Status *status);
    INOUT request      request handle
    OUT   flag         true if operation completed
    OUT   status       status object

MPI_Barrier(MPI_Comm comm)
   IN       comm        communicator
```

Funciones MPI

- Envío y Recepción múltiples

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

```
MPI_Gather(void* sendbuf, int sendcount,
 MPI_Datatype sendtype, void* recvbuf, int
 recvcount, MPI_Datatype recvtype, int root,
 MPI_Comm comm)
```

```
MPI_Scatter(void* sendbuf, int sendcount,
 MPI_Datatype sendtype, void* recvbuf, int
 recvcount, MPI_Datatype recvtype,int root,
 MPI_Comm comm)
```

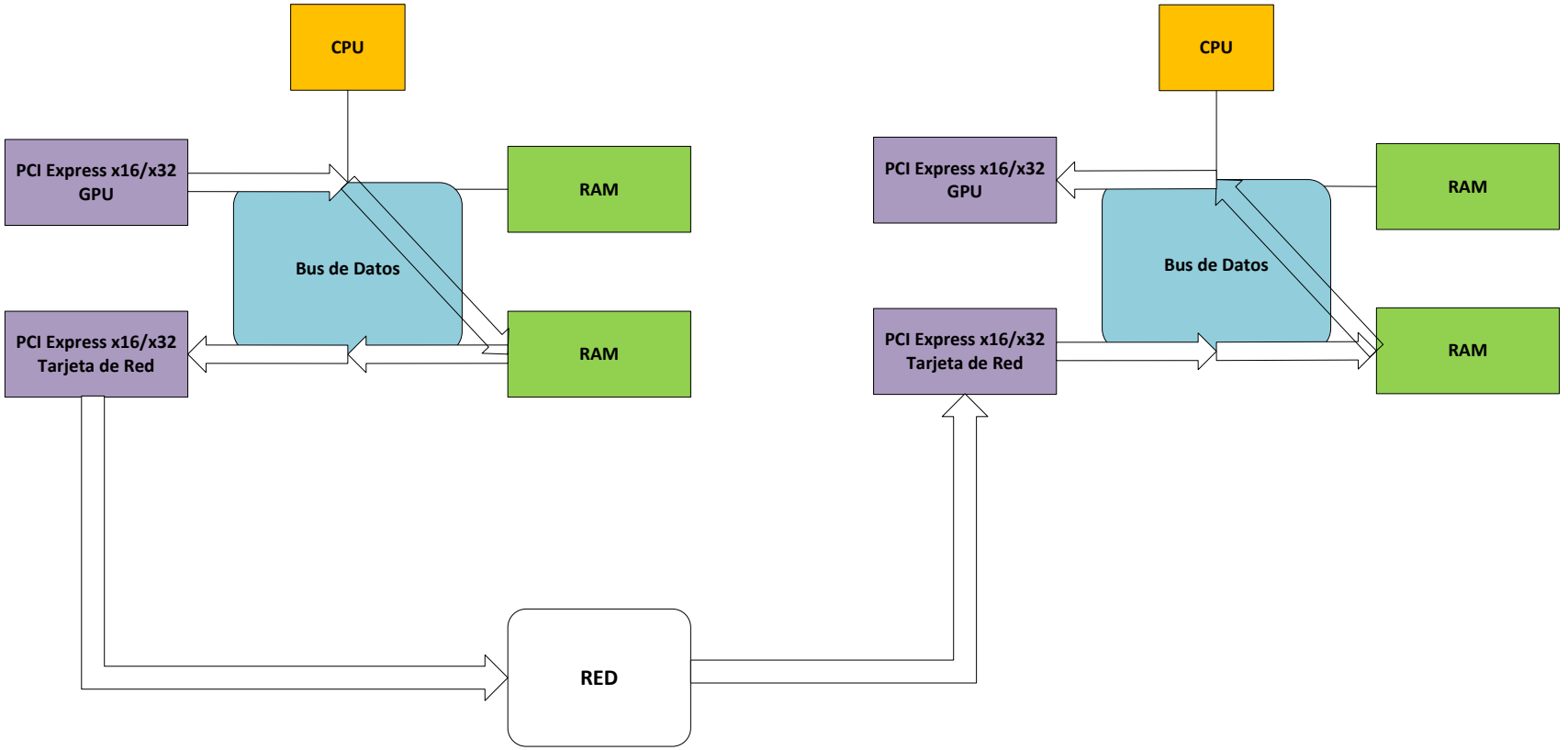
Esquema maestro/esclavo

```
#include “mpi.h”

main(int argc, char,**argv)
{
   int nproc, rank;
   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD, &nproc);
   MPI_Comm_Rank(MPI_COMM_WORLD, &rank);
   if (rank ==0)
          master( );
   else
        slave( );
   MPI_Finalize();
}
```

MPI y CUDA

1. Copiar datos de la tarjeta gráfica a la RAM
2. Enviar datos desde la RAM a través de la red
3. Recibir datos en la RAM del otro equipo
4. Copiar datos de la RAM a la tarjeta gráfica



Practica 2.1

- Modificar los archivos helloWorld1.cu y helloWorld2.cu para que funcionen con MPI en distintos nodos, haciendo que cada hilo imprima el nodo en el que está, además de lo que ya imprimiese anteriormente

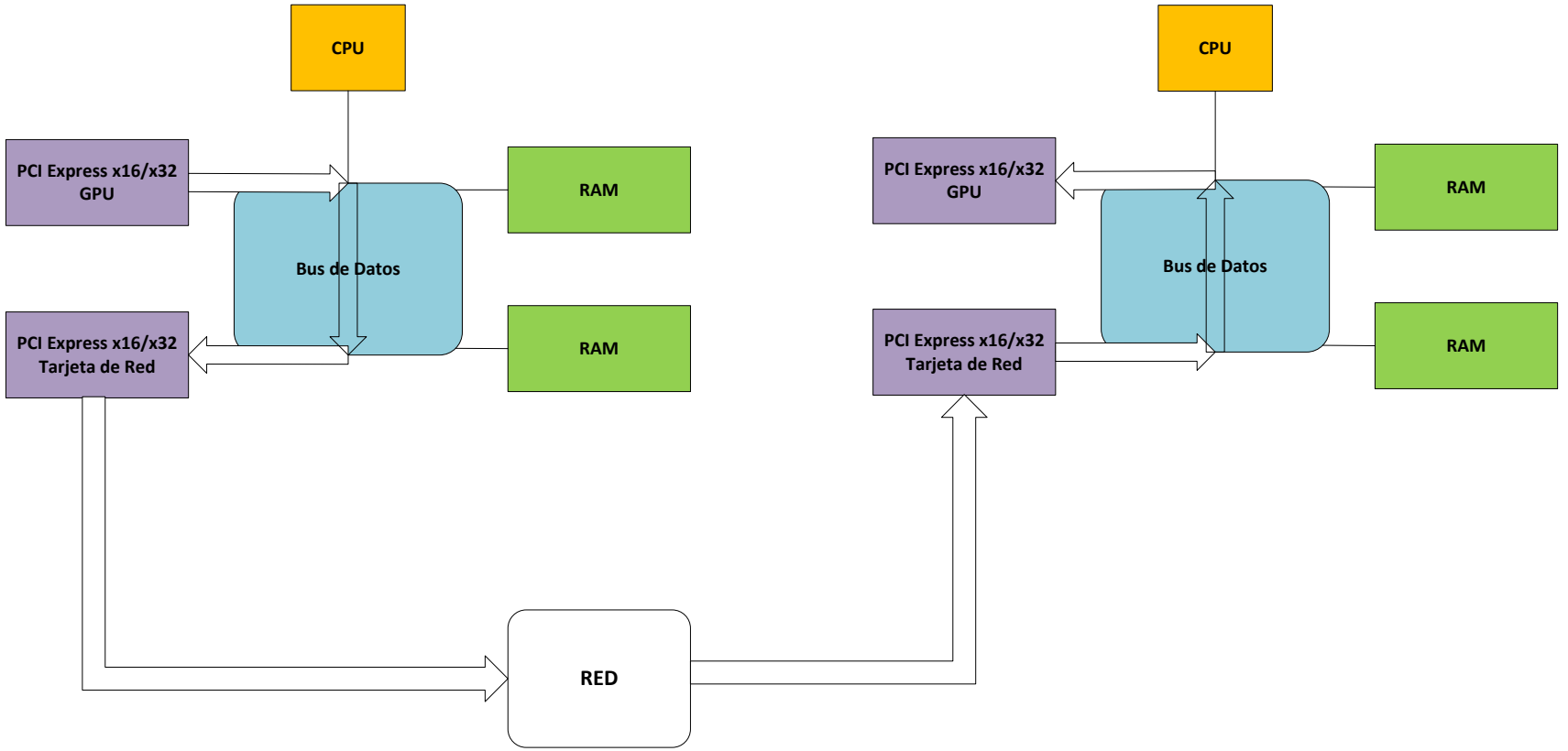
- Compilar:
  - **nvcc -I/usr/mpi/gcc/openmpi-1.4.6/include -L/usr/mpi/gcc/openmpi-1.4.6/lib64 -lmpi -arch=sm_30 -gencode arch=compute_30,code=sm_30 -g -G <nombre>.cu -o <nombre>**
- Ejecutar:
  - **mpirun –np <numeroNodos> <programa>**

Practica 2.2

- Modificar los ficheros suma.cu y sumaAfin.cu para que las operaciones se realicen en paralelo con MPI, haciendo que uno de los nodos (nodo maestro) realice una distribución de los datos, y el resto de los nodos (nodos esclavo) hagan las operaciones dentro de un kernel de CUDA. Después los nodos esclavo deberán enviar su resultado al nodo maestro. Finalmente el nodo maestro deberá imprimir el resultado.
  - **NOTA: Recordar que los datos no están en la RAM, sino en la tarjeta gráfica**

MPI CUDA-aware

- Copia los datos directamente al buffer de la tarjeta de red (si el bus de datos y la tarjeta gráfica lo soportan)
- Si el bus de datos y la tarjeta gráfica no soportan esta funcionalidad, la implementación de MPI realizará la copia a la RAM



Practica 2.3

- Modificar el ejercicio anterior para el soporte CUDA-aware de MPI