

ARQUITECTURA DE COMPUTADORES

Ampliación de MPI y Práctica 3

Departamento de Electrónica y Sistemas - Facultad de Informática

Grado en Ingeniería Informática



Índice

1 Tipos de Datos Derivados

2 Comunicadores

3 Práctica 3



Tipos de Datos Derivados (I)

Clasificación

- Tipos homogéneos → Empaquetan elementos del mismo tipo
 - Tipo contiguo → Conjunto de elementos contiguos en memoria
 - Tipo vector → Conjunto de elementos con stride fijo
 - Tipo indexed → Conjunto de elementos con stride variable
- Tipos heterogéneos → Empaquetan elementos de distintos tipos
 - Tipo struct → Similar a las structs de C

Tipos de Datos Derivados (II)

Proceso de Creación

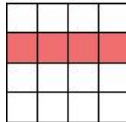
- 1 Declarar el tipo → *MPI_Datatype nuevoTipo;*
- 2 Especificar el tipo → *MPI_Type_XXX(..., ..., &nuevoTipo);*
- 3 Crear el tipo → *MPI_Type_commit(&nuevoTipo);*
- 4 Trabajar con el tipo
- 5 Liberar el la memoria asignada al tipo
→ *MPI_Type_free(&nuevoTipo);*

Tipos de Datos Derivados (III)

MPI_Type_contiguous: Empaquetado de *count* elementos del tipo *oldtype*.

```
MPI_Type_contiguous(count, oldtype, *newtype);
```

mat [16]




```
MPI_Type_contiguous(4, MPI_FLOAT, &nuevoTipo);  
MPI_Type_commit(&nuevoTipo);  
MPI_Send(&mat[4], 1, nuevoTipo, dest, tag, comm);
```

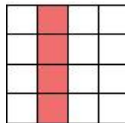
Complemento: ejemplo para structs

Tipos de Datos Derivados (IV)

MPI_Type_vector: Empaquetado de *count* bloques de *blocklength* elementos del tipo *oldtype* separados una distancia *stride* (variante: hvector).

```
MPI_Type_vector(count, blocklength, stride,  
                oldtype, *newtype);
```

mat [16]



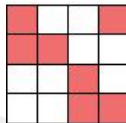
```
MPI_Type_vector(4, 1, 4, MPI_FLOAT, &nuevoTipo);  
MPI_Type_commit(&nuevoTipo);  
MPI_Send(&mat[1], 1, nuevoTipo, dest, tag, comm);
```

Tipos de Datos Derivados (V)

MPI_Type_indexed: Empaquetado de *count* bloques de una cantidad variable de elementos del tipo *oldtype* separados una distancia variable. La cantidad de elementos del bloque *i* se especifica en *blocklength[i]*. La distancia desde el primer elemento del array al primer elemento del bloque *i* se especifica en *offset[i]* (variante: *hindexed*).

```
MPI_Type_indexed(count, blocklength[], offset[],
                 oldtype, *newtype);
```

mat [16]



```
int lengths[4] = {1,3,1,2}; int offsets[4] = {0,3,10,14};
MPI_Type_indexed(4, lengths, offsets, MPI_FLOAT, &nuevoTipo);
MPI_Type_commit(&nuevoTipo);
MPI_Send(&mat[0], 1, nuevoTipo, dest, tag, comm);
```

Tipos de Datos Derivados (VI)

MPI_Type_struct: Empaquetado de *count* bloques de una cantidad y tipo variable de elementos. La cantidad de elementos del bloque *i* se especifica en *blocklength[i]*. La distancia desde el primer elemento del array al primer elemento del bloque *i* se especifica en *offset[i]*. El tipo de los elementos del bloque *i* se especifica en *old_types[i]*.

```
MPI_Type_struct (count, blocklength[], offset[],  
                old_types[], *newtype)
```



Tipos de Datos Derivados (VII)

Ejemplo: Envío de 3 enteros y 1 float conjuntamente

```
typedef struct{
    int i1, i2, i3;
    float f1;
} estructura;

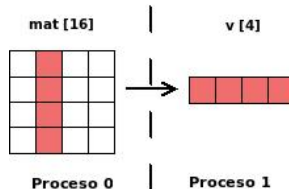
estructura est;
est.i1 = 1; est.i2 = 2; est.i3 = 3; est.f1 = 0.1;
int lengths[2] = {3,1};
int offsets[2]; offsets[0] = 0; offsets[1] = 3*sizeof(int);
int types[2]; types[0] = MPI_INT; types[1] = MPI_FLOAT;
MPI_Type_struct(2, lengths, offsets, types, &nuevoTipo);
MPI_Type_commit(&nuevoTipo);
MPI_Send(&est, 1, nuevoTipo, dest, tag, comm);

/* Complemento: Uso del tipo MPI_Aint */
/* y de la funcion MPI_Address */
```

Tipos de Datos Derivados (y VIII)

Desconocimiento del Tipo en Recepción

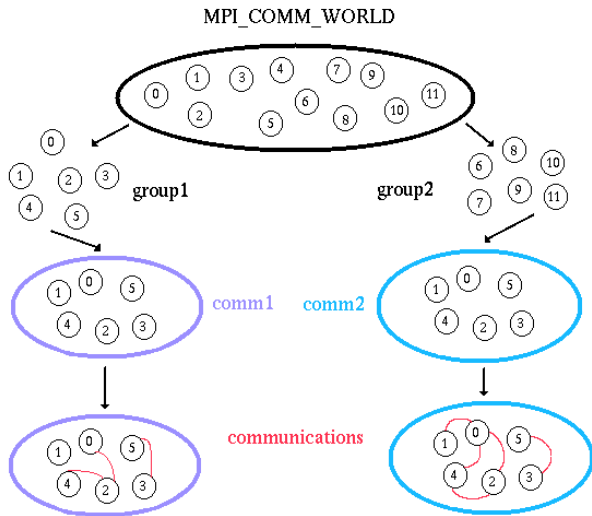
- El receptor no necesita conocer el nuevo tipo de dato. Puede recibir especificando el tipo de dato original y el número de elementos.



```
if(rank == 0){ MPI_Type_vector(4, 1, 4, MPI_FLOAT, &nuevoTipo);
  MPI_Type_commit(&nuevoTipo);
  MPI_Send(&mat[1], 1, nuevoTipo, 1, tag, comm);}
if(rank == 1){
  MPI_Recv(&v[0], 4, MPI_FLOAT, 0, tag, comm, &status);}
```

Complemento: MPI-Pack/Unpack (alternativa a TDD en ciertos casos)

Comunicadores (I)



Comunicadores (II)

Declaración

- `MPI_Comm nuevoComm;`

Creación

- `MPI_Comm_split(MPI_Comm comm, int color, int key, MPI_Comm *newcomm)`
 - Crea un *newcomm* que identifica subconjuntos de *comm*
 - *color* identifica el grupo al que va cada proceso
 - *key* identifica el rango del proceso en el nuevo grupo
- Creación genérica: `MPI_Comm_create` (en combinación con las funciones `MPI_Comm_group`, `MPI_Group_incl/excl` y el tipo `MPI_Group`)

Comunicadores (III)

Información

- `MPI_Comm_rank(MPI_Comm comm, int *rank)`
 - Devuelve el rango del proceso dentro de *comm*

Eliminación

- `MPI_Comm_free(MPI_Comm *comm)`

Ejemplo: División en grupos pares e impares

```
MPI_Comm parImparComm;  
int color = rank%2;  
int key = rank/2;  
MPI_Comm_split(MPI_COMM_WORLD, color, key, &parImparComm);  
...  
MPI_Comm_free(&parImparComm);
```

Comunicadores (y IV)

Complemento: Topologías virtuales

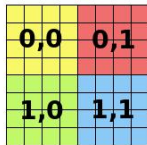
- Creación de topologías cartesianas: `MPI_Cart_create(MPI_Comm oldcomm, int ndims, int dims[], int periods[], int reorder, MPI_Comm *cartcomm)`
- Funciones de información:
 - `MPI_Cart_get (MPI_Comm comm, int maxdims, int dims[], int periods[], int coords[])`
 - `MPI_Cart_rank(MPI_Comm comm, int coords[], int *rango)`
 - `MPI_Cart_coords(MPI_Comm comm, int rango, int maxdims, int coords[])`
 - `MPI_Cart_shift(MPI_Comm, int direccion, int displ, int *rango_origen, int *rango_destino) (+ MPI_PROC_NULL)`
- Otras funciones:
 - Crear dims: `MPI_Dims_create(int nprocs, int ndims, int dims[])`
 - Subtopologías virtuales: `MPI_Cart_sub(MPI_Comm comm, int freedims[], MPI_Comm *newcomm)`

Práctica 3 (I)

Características

- **SUMMA:** Scalable Universal Matrix Multiplication Algorithm
- Misma 2D distribución en A , B y C
- Minimiza la sobrecarga de memoria por proceso
- $\sqrt{\text{numprocs}}$ iteraciones
- Dos broadcast y un producto secuencial por iteración y proceso

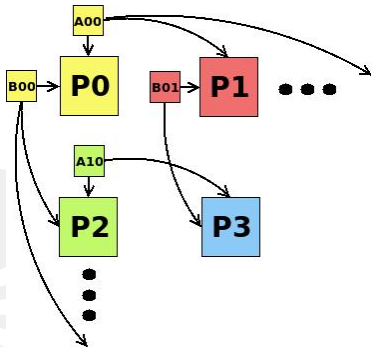
Rank 0 Rank 1 Rank 2 Rank 3



Práctica 3 (II)

Iteración 1

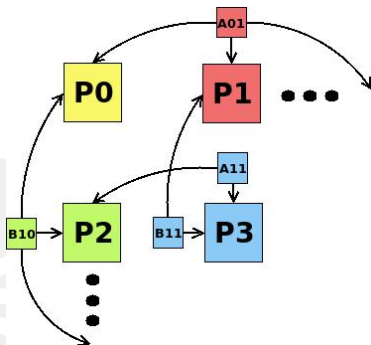
- Broadcast de $A_{i,0}$ en horizontal
- Broadcast de $B_{0,i}$ en vertical
- Cada proceso multiplica los bloques que tiene



Práctica 3 (III)

Iteración 2

- Broadcast de $A_{i,1}$ en horizontal
- Broadcast de $B_{1,i}$ en vertical
- Cada proceso multiplica los bloques que tiene y suma a lo de la iteración anterior



Práctica 3 (IV)

Pseudocódigo

- $\forall j \in [0, \sqrt{\text{numprocs}} - 1]$
 - Broadcast $A_{\text{fila},j}$ a la fila de la malla
 - Broadcast $B_{j,\text{col}}$ a la columna de la malla
 - Cada proceso multiplica los bloques que acaba de recibir
 - Cada proceso suma el resultado de su multiplicación a los resultados previos

Práctica 3 (V)

PROBABLEMENTE REQUIERA TRABAJO EN CASA!!!

Pautas Obligatorias

- m , n , k y α pasados por teclado
- Inicialización de A y B en proceso 0
- Distribución de datos en 2D
- Recolección de matriz C al final en proceso 0
- Resultados correctos para cualquier malla de procesos cuadrada ($\sqrt{\text{numprocs}}$ X $\sqrt{\text{numprocs}}$)
- Resultados correctos para cualquier m , n y k múltiplos de $\sqrt{\text{numprocs}}$
- Crear comunicadores para representar a filas y columnas de la malla
- Absolutamente **TODOS** los broadcast internos al bucle deben realizarse utilizando los **NUEVOS COMUNICADORES**
- Práctica **INDIVIDUAL** que se debe defender ante el profesor
- Fecha límite para defensa: Por determinar

Práctica 3 (y VI)

Bonificaciones

- Todas las comunicaciones con tipos de datos derivados
- Defensa en clases de prácticas anteriores a la fecha límite

