MEMORIA P4: Nuevos tipos de datos

Zdravko Dimitrov Arnaudov

Índice

1.

a) En un programa tienes una matriz y el problema a resolver te exige trabajar con columnas.Crea un programa MPI que tenga una matriz de 10x10 datos de tipo double y escribe el código para pasar la columna 2 del proceso 0, que inicializa la matriz (por ejemplo, con los índices como en el caso de la práctica 3), al proceso 1 que la tiene vacía. Comprueba que los datos del proceso 1 son los correctos

El flujo del programa es el siguiente. Primero, ha de indicarse y definirse el nuevo tipo de dato, luego se procederá a hacer commit del mismo. El último paso que nos queda, es mandar desde el proceso 0 la columna e imprimirla en P1.
El output que obtenemos es:

```
mpirun -np 2 Ejer1_A
Soy P2, imprimo columa 2 de M
0.00
1.00
2.00
3.00
4.00
5.00
6.00
7.00
8.00
9.00
```

b) De forma parecida a A), el proceso 0 rellena una matriz de NxN y manda al resto de procesos una columna de la misma. La diferencia con respecto a la anterior es que los procesos(incluido el mismo) reciben la columna en un array. Debes ejecutar el programa con N procesos.

Este apartado es parecido al anterior. La diferencia es que ahora las columnas se reciben en forma de array.

Entendemos que seguimos recibiendo 10 dobles. Por lo tanto, en recepción usamos la llamada: **MPI_Recv (M2, 10, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status);**

El resultado del programa es este:

```
mpirun -np 4 Ejer1_B
Soy P1, imprimo columa 2 de M
0.00
1.00
2.00
3.00
4.00
5.00
6.00
7.00
8.00
9.00
```

c) **Utiliza el mismo programa anterior y haz de él una copia. Ahora, tienes dos matrices de NxN float. El proceso 0 inicializa la matriz A y pasa la matriz triangular superior a un proceso 1 que pone esos datos en la matriz B (date cuenta que la matriz podría ser la misma en ambos procesos, pero inicializada sólo en el primero) . Comprueba que los datos del proceso 1 son los correctos y que por tanto la matriz del proceso 1 es una matriz triangular superior.**

Emplearemos el mismo programa que antes, utilizando una copia.

Primero, lo que haremos será inicializar la matriz triangular superior desde el proceso 0. Una vez esto, el siguiente paso es considerar de qué tipo de dato estamos hablando. Como vemos, ahora el número de bloques del dato es distinto, al igual que su separación. Por eso, consideramos que se trataría de un tipo de dato homogéneo-indexado.

Ahora, necesitaremos dos arrays, uno para las longitudes de los datos y otro para los desplazamientos.

```
mpirun -np 2 Ejer1_C
Soy el proceso 1, imprimo matriz triangular:
```

```
0.00  1.00  2.00  3.00  4.00  5.00  6.00  7.00  8.00  9.00
0.00  0.00  1.00  2.00  3.00  4.00  5.00  6.00  7.00  8.00
0.00  0.00  0.00  1.00  2.00  3.00  4.00  5.00  6.00  7.00
0.00  0.00  0.00  0.00  1.00  2.00  3.00  4.00  5.00  6.00
0.00  0.00  0.00  0.00  0.00  1.00  2.00  3.00  4.00  5.00
0.00  0.00  0.00  0.00  0.00  0.00  1.00  2.00  3.00  4.00
0.00  0.00  0.00  0.00  0.00  0.00  0.00  1.00  2.00  3.00
0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  1.00  2.00
0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  1.00
0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
```

d) Ten en cuenta los consejos del apartado B. Ahora tienes una matriz como la de la figura de abajo. El proceso 0 inicializa las casillas negras y las pasa como tipo derivado al proceso 1 que las coloca en su tablero. Comprueba que los datos del proceso 1 son los correctos. Sugerencia, se pueden crear tipos de datos derivados de un derivado. Es interesante que compruebes la extensión del tipo.

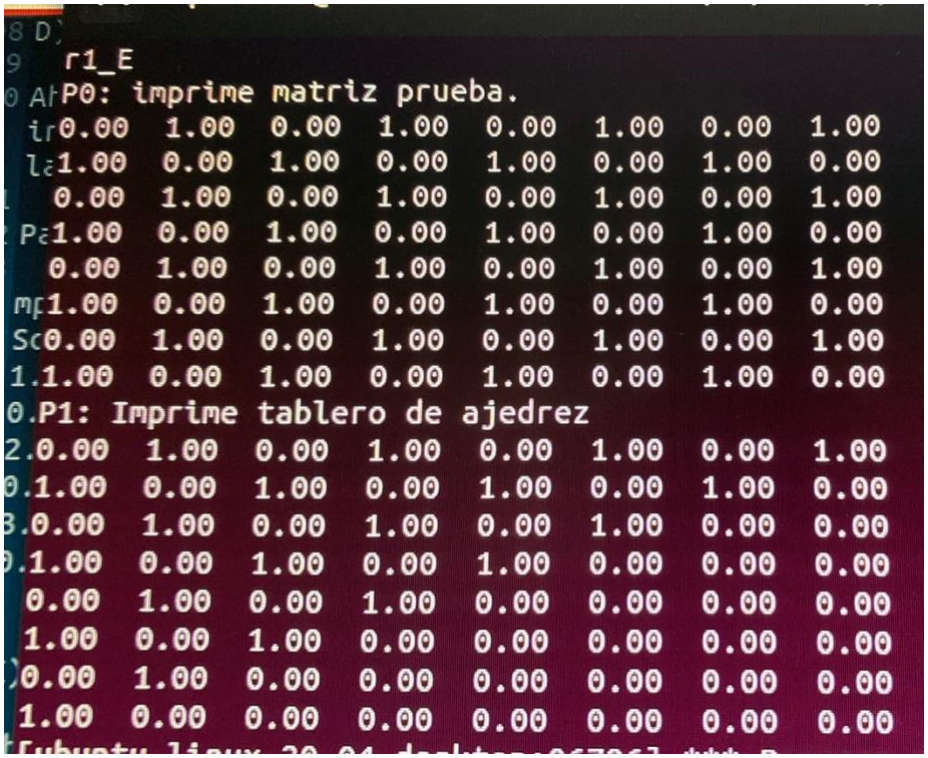
Nos hemos visto consumiendo demasiado tiempo resolviendo el ejercicio mediante tipos derivados de derivados. Con lo cual, hemos vuelto a usar el indexado. Para estas dimensiones, sigue pareciendo razonable.

```
mpirun -np 2 Ejer1_D
Soy el proceso 1, imprimo matriz triangular:
1.00  0.00  1.00  0.00  1.00
0.00  0.00  0.00  0.00  0.00
2.00  0.00  2.00  0.00  2.00
0.00  0.00  0.00  0.00  0.00
3.00  0.00  3.00  0.00  3.00
0.00  0.00  0.00  0.00  0.00
```

e) Ahora tienes una matriz que representa un tablero de ajedrez/damas. El proceso 0 inicializa las casillas blancas y las pasa como tipo derivado al proceso 1 que las coloca en su tablero. Realízalo con un tipo de datos derivado de un derivado. Si no puedes hazlo con indexado. Es interesante que te des cuenta del extent para que veas cómo se puede hacer

De nuevo, nos hemos encontrado con complicaciones para hacerlo con tipo de dato derivado de derivado. La idea principal, era adoptar las dos primeras líneas del tablero como un tipo de dato y luego el derivado sería unirlas 4 veces de forma seguida.

El problema al que hemos llegado es que el tablero aparece desplazado, de forma que solo aparece correcta una mitad triangular. Intuimos que es problema de los desplazamientos, pero no hemos conseguido arreglarlo con extent.



Por este motivo y para no consumir más tiempo, hemos decidido hacer un tipo de dato fila y mandarlo 8 veces al siguiente proceso. Para hacerlo, tenemos en cuenta que las filas pares se desplazan una columna a la derecha, mientras que las impares se mandan desde la columna 0.

El tipo de dato en sí, lo hemos hecho indexado pero podría haber sido perfectamente como vector.

```
mpirun -np 2 Ejer1_E
P0: imprime matriz prueba.
0.00  1.00  0.00  1.00  0.00  1.00  0.00  1.00
1.00  0.00  1.00  0.00  1.00  0.00  1.00  0.00
0.00  1.00  0.00  1.00  0.00  1.00  0.00  1.00
1.00  0.00  1.00  0.00  1.00  0.00  1.00  0.00
0.00  1.00  0.00  1.00  0.00  1.00  0.00  1.00
1.00  0.00  1.00  0.00  1.00  0.00  1.00  0.00
0.00  1.00  0.00  1.00  0.00  1.00  0.00  1.00
1.00  0.00  1.00  0.00  1.00  0.00  1.00  0.00
P1: Imprime tablero de ajedrez
0.00  1.00  0.00  1.00  0.00  1.00  0.00  1.00
1.00  0.00  1.00  0.00  1.00  0.00  1.00  0.00
0.00  1.00  0.00  1.00  0.00  1.00  0.00  1.00
1.00  0.00  1.00  0.00  1.00  0.00  1.00  0.00
0.00  1.00  0.00  1.00  0.00  1.00  0.00  1.00
1.00  0.00  1.00  0.00  1.00  0.00  1.00  0.00
0.00  1.00  0.00  1.00  0.00  1.00  0.00  1.00
1.00  0.00  1.00  0.00  1.00  0.00  1.00  0.00
```

2.

a) Crea un tipo de dato que sea combinación de un carácter y un flotante. Construye el tipo de dato MPI y mándalo entre dos procesos. Para construirlo utilizarás el extent.

Creamos una estructura código que contempla un carácter y un flotante. Creamos el dato heterogéneo y lo mandamos al otro proceso.

```
mpirun -np 2 Ejer2_A
P0 el código es:2.00A
P1 el código es:2.00A
```

b) Crea un tipo de dato que sea combinación de un carácter y un flotante. Construye el tipo de dato MPI y mándalo entre dos procesos. Para construirlo utilizarás las direcciones que proporciona MPI.

Simplemente reutilizamos el código anterior y cambiamos los desplazamientos con las direcciones.

```
mpirun -np 2 Ejer2_B
P0 el código es:2.00A
P1 el código es:2.00A
```

c) En cualquiera de los dos ejemplos anteriores prueba el tamaño del tipo de dato MPI nuevo con size y haz lo mismo con C con diferencias de direcciones (punteros de los miembros de la estructura). Indica cual sería el extent y el size con un tipo de dato doble.

Comenzaremos probando el tamaño del tipo de dato MPI nuevo con size, del apartado A. Simplemente, mediremos en el programa ya existente cuál es su tamaño.

```
mpirun -np 2 Ejer2_A
(Apartado C) El tamaño del tipo de dato code es :5
bytes
P0 el código es:2.00A
P1 el código es:2.00A
```

El tamaño con size es 5 bytes: 4 de float y 1 de char.

Ahora, veremos qué sucede si cambiamos el tipo de dato del primer campo de la estructura a doble.

```
mpirun -np 2 Ejer2_A
(Apartado C) El tamaño del tipo de dato code es :9 bytes
P0 el código es:2.00A
P1 el código es:2.00A
```

Efectivamente, 8 del doble y 1 del char. Veremos en este caso también cuánto vale el extent para este tipo de dato completo.

```
mpirun -np 2 Ejer2_C_A
(Apartado C) Size: 9 bytes
(Apartado C) Extent: 16 bytes
P0 el código es:2.00A
P1 el código es:2.00A
```

La diferencia es que extent incluye los alineamientos.

d) Copia el programa en uno nuevo y envía desde el programa 0 al 1 este tipo de dato derivado. Una vez recibido el mensaje, comprueba con su estatus lo que te dan las funciones MPI_Get_elements y MPI_Get_count poniendo el tipo derivado.

A raíz del tipo de dato anteriormente creado, ahora generaremos un nuevo tipo que combine dos de ellos consecutivos. En particular, se añadirá un entero a la secuencia de campos del primer tipo de dato. Para ello, creamos una nueva estructura donde su primer campo sea el tipo de dato antiguo y el segundo el entero(5).

```
mpirun -np 2 Ejer2_D
Size: 13 bytes
Extent: 24 bytes
P0 el código es:2.00A5
P1 el código es:2.00A5
```

Ahora, comprobaremos el valor que nos devuelven:

MPI_Get_elements: devuelve el número de elementos básicos en el tipo de dato

MPI_Get_count: devuelve el número de elementos recibidos desde el status de una operación de recepción.

```
mpirun -np 2 Ejer2_D
Size: 13 bytes
Extent: 24 bytes
P0 el código es:2.00A5
P1 el código es:2.00A5
GET ELEMENTS: 3
GET COUNT: 1
```

Entonces, el primero representa todos los campos que tiene en total la estructura enviada, mientras que el segundo nos enseña el número de este tipo de dato enviado.

e) Modifica el problema del tren de la práctica 3 para pasar un tipo de dato heterogéneo sin necesidad de conversión. No hagas un alltoall, haz un send/rec entre dos procesos para pasar el tipo de dato {entero+string[20]}.

Simulamos con los send/receive una operación alltoall. Ahora, el tipo de dato heterogéneo será el pasajero.

```
mpirun -np 4 Ejer2_E
Soy 0: 1 2 3 4
Soy 1: 1 2 3 4
Soy 2: 1 2 3 4
Soy 3: 1 2 3 4
Soy 2: 3 3 3 3
Soy 3: 4 4 4 4
Soy 0: 1 1 1 1
Soy 1: 2 2 2 2
```

3.

Realiza dos comunicadores en que uno sean los procesos pares y otros los impares y hagan cadauno un all_reduce con suma de sus identificadores. Se admite que el número de procesos sea par siempre. Lo harás de dos formas diferentes:

a). A partir de un comunicador creado desde el global (split).

```
mpirun -np 4 Ejer3_B
Soy 3 de color 1
Soy 0 de color 0
Soy 1 de color 1
Soy 2 de color 0
Soy 3 en global y 1 en local
Los tamaños son: 4 y 2
Soy 0 en global y 0 en local
Los tamaños son: 4 y 2
Soy 1 en global y 0 en local
Los tamaños son: 4 y 2
Soy 2 en global y 1 en local
Los tamaños son: 4 y 2
```

b). A partir de un grupo nuevo que puede crear desde el comunicador global y después modificándole.

Primero, imprimimos los rangos.

```
mpirun -np 4 Ejer3_A
Soy 0 en global y 0 en par local.
Soy 1 en global y 0 en impar local.
Soy 2 en global y 1 en par local.
Soy 3 en global y 1 en impar local.
```

Ahora, probaremos usando la llamada **MPI_Allreduce** en cada uno de los comunicadores. Usaremos los rangos globales, por curiosidad, pues si usásemos los de cada grupo, obtendríamos siempre 1.

```
Soy 1 en global y 0 en impar local.
Soy 2 en global y 1 en par local.
Soy 3 en global y 1 en impar local.
Soy 0 en global y 0 en par local.
La suma en PAR es: 2
La suma en IMPAR es: 4
```

Como vemos, todo funciona como debería. Los procesos están en el comunicador adecuado.

c). Voluntaria: Se tiene una malla 2D de procesos de 2 filas y 3 columnas (6 procesos). Para que la comunicación sea más eficiente cada elemento de la malla se puede comunicar solo con su vecino de abajo- arriba y derecha izquierda. Crea los comunicadores necesarios para esta situación

Cada proceso dispondrá de un comunicador horizontal y vertical. Generamos un switch, de forma que según el rango global, los procesos tengan asignación de un comunicador u otro.

```
mpirun --oversubscribe -np 6 Ejer3_C
Soy 2 de color horizontal 0 y vertical 4
Soy 5 de color horizontal 1 y vertical 4
Soy 0 de color horizontal 0 y vertical 2
Soy 4 de color horizontal 1 y vertical 3
Soy 3 de color horizontal 1 y vertical 2
Soy 1 de color horizontal 0 y vertical 3
Soy 4 en global y 1 en local horizontal
Soy 0 en global y 0 en local horizontal
Soy 0 en global y 0 en local vertical
Soy 2 en global y 2 en local horizontal
```

```
Soy 2 en global y 0 en local vertical
Soy 3 en global y 0 en local horizontal
Soy 3 en global y 1 en local vertical
Soy 1 en global y 1 en local horizontal
Soy 1 en global y 0 en local vertical
Soy 5 en global y 2 en local horizontal
Soy 5 en global y 1 en local vertical
Soy 4 en global y 1 en local vertical
```

Ahora, usamos la llamada AllReduce para los comunicadores horizontales y verticales. En este caso, emplearemos los rangos globales.

```
mpirun --oversubscribe -np 6 Ejer3_C
Soy 0 de color horizontal 0 y vertical 2
Soy 2 de color horizontal 0 y vertical 4
Soy 1 de color horizontal 0 y vertical 3
Soy 3 de color horizontal 1 y vertical 2
Soy 4 de color horizontal 1 y vertical 3
Soy 5 de color horizontal 1 y vertical 4
Soy 0 en global y 0 en local horizontal
Soy 0 en global y 0 en local vertical
Soy 4 en global y 1 en local horizontal
Soy 4 en global y 1 en local vertical
Soy 1 en global y 1 en local horizontal
Soy 5 en global y 2 en local horizontal
Soy 5 en global y 1 en local vertical
Soy 1 en global y 0 en local vertical
Soy 2 en global y 2 en local horizontal
Soy 2 en global y 0 en local vertical
Soy 3 en global y 0 en local horizontal
Soy 3 en global y 1 en local vertical
Soy P5 global: Color Horizontal: 1, Valor: 12
Soy P5 global: Color Vertical: 4, Valor: 7
Soy P4 global: Color Horizontal: 1, Valor: 12
Soy P4 global: Color Vertical: 3, Valor: 5
Soy P3 global: Color Horizontal: 1, Valor: 12
Soy P3 global: Color Vertical: 2, Valor: 3
Soy P2 global: Color Horizontal: 0, Valor: 3
Soy P2 global: Color Vertical: 4, Valor: 7
Soy P0 global: Color Horizontal: 0, Valor: 3
Soy P0 global: Color Vertical: 2, Valor: 3
Soy P1 global: Color Horizontal: 0, Valor: 3
Soy P1 global: Color Vertical: 3, Valor: 5
```