Tipos Derivados

OpenMPI

Computación Paralela y Distribuida

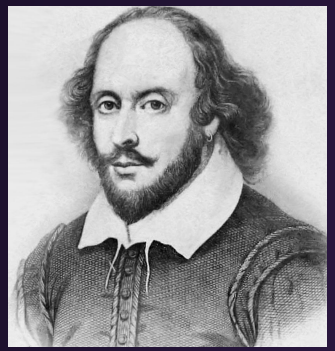
Agenda

- Reflexiones Importantes
- Tipos Derivados
- MPI_Type_create_struct
- Broadcast de un Tipo Derivado

¿Un mensaje o varios?

¿Reflexione si es mejor enviar un mensaje o varios?

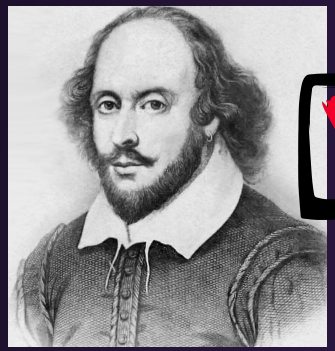
1 mensaje de KN bytes



Es mejor enviar un mensaje grande

Debido al “costo” del uso de la red.

1 mensaje de KN bytes



Estrategias para reducir el número de mensajes

- Parámetro `count` en cualquier función de mensaje puntual o colectiva.
  - Arreglos del mismo tipo
- `MPI_Pack` & `MPI_Unpack`.
- Tipos Derivados

Tipos Derivados

Tipos Derivados de MPI

- Representan cualquier colección de datos en memoria
- Describimos el tipo MPI de cada elemento
- Indicamos la ubicación relativa a la dirección base.

Elemento 1 MPI_type	Dirección base
Elemento 2 MPI_type	Dirección relativa a base
Elemento 2 MPI_type	Dirección relativa a base

Acceso a Tipos Derivados

Función de envío

Si conoce el tipo y la ubicación relativa de un conjunto de elementos, puede obtener cualquier elemento antes de enviarlos.

Función que recibe

Puede distribuir elementos a memoria antes que sean recibidos, asignando los tipos y ubicaciones relativas correctas.

Mensajes en Regla Trapezoidal

En el ejemplo de la Regla Trapezoidal queremos distribuir los valores globales:

1. **Límite inferior a** — **MPI_Receive / MPI_Bcast**
2. **Límite superior b** — **MPI_Receive / MPI_Bcast**
3. **N trapezoides** — **MPI_Receive / MPI_Bcast**

Estos son al menos 3 mensajes x N procesos ...

Organización de memoria para Mensajes en Regla Trapezoidal

Organización de memoria para valores globales:

1.	Límite inferior a	tipo double (8B)	dirección base
2.	Límite superior b	tipo double (8B)	base + 8B
3.	N trapezoides	tipo int (4B)	base + 16B

Distancia relativa – Regla Trapezoidal

El desplazamiento relativo depende del tipo:

Dirección	Tipo	Tamaño	B3	B2	B1	B0
0x0020	a (double)	8 bytes	23	22	21	20
0x0024			27	26	25	24
0x0028	b (double)	8 bytes	31	30	29	28
0x0032			35	34	33	32
0x0036	n (int)	4 bytes	39	38	37	36

{(MPI_DOUBLE, 0), (MPI_DOUBLE, 8), (MPI_INT, 16)}

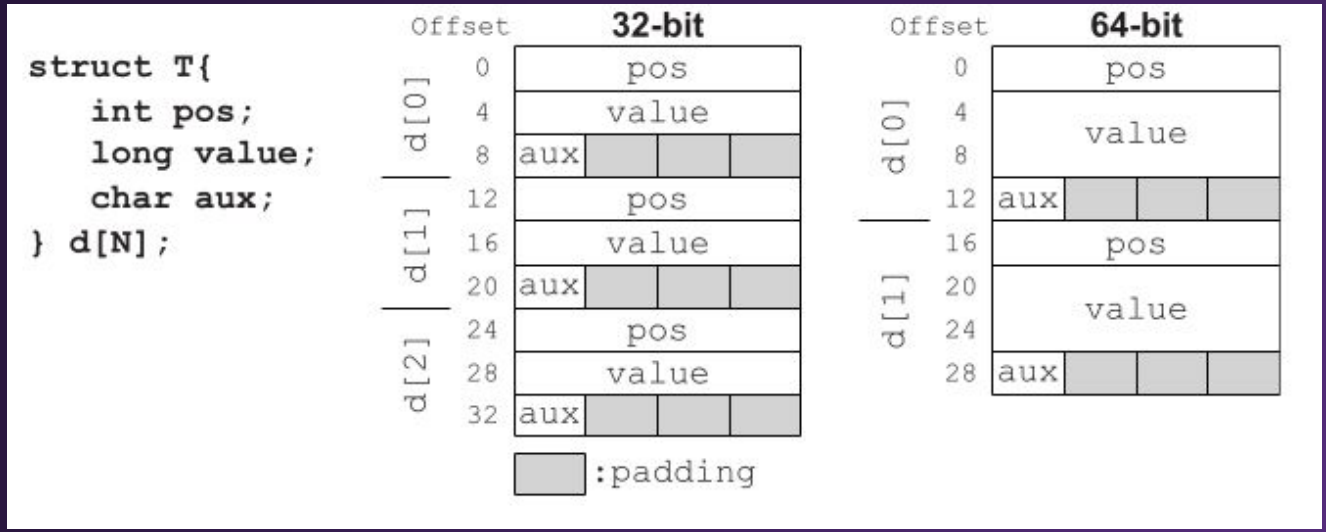
MPI_Type_create_struct

Recordemos que...

Dependiendo de la Arquitectura y factores relacionados, la representación de tipos de datos puede no ser igual en todos los casos (esp. En computación heterogénea).

Por ejemplo:

long



MPI_Type_create_struct

Rutina genérica para creación de tipos:

```
int MPI_Type_create_struct(
    int             count                   //in,
    int             blocklengths_array[]    //in,
    MPI_Aint        displacements_array[]   //in,
    MPI_Datatype    types_array[]           //in,
    MPI_Datatype*   new_type_handler_p      //out);
```

Inicilización de MPI_Type_create_struct

- Count: número ***n*** de elementos en el nuevo tipo
- blocklength_array[n] = {i_0, i_1, i_2, ... i_n}
- displacement_array[n] = {0, d_1, d_2, ... d_n}
- datatype_array[n] = {MPI_datatype, MPI_datatype ...}
- new_type_handler_p = Nombre del nuevo tipo

Argumentos para inicializar MPI_Type_create_struct

Todos los argumentos deben estar inicializados antes.

```
//- Un arreglo INT para el número de elementos del tipo
int blocklengths_array[3] = {1,1,1};

//- Otro arreglo MPI_Datatype para los tipos MPI
MPI_Datatype datatypes_array[3] = {MPI_DOUBLE,
        MPI_DOUBLE, MPI_INT};

//- El arreglo MPI_Aint donde guardaremos los offsets
MPI_Aint displacements_array[3] = {0,0,0};
    //El primer elemento tiene desplazamiento 0
...
```

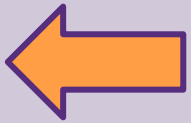
MPI_Get_address: dirección referencias

MPI_Get_address devuelve en un tipo MPI_Aint la dirección de una referencia de memoria.

```
MPI_Aint a_addr, b_addr, n_addr;

MPI_Get_address(&a, &a_addr);
MPI_Get_address(&b, &b_addr);
MPI_Get_address(&n, &n_addr);

displacements_array[1]= b_addr - a_addr;
displacements_array[2]= n_addr - a_addr;
...
```



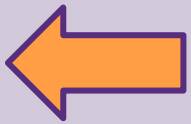
Cálculo del offset para MPI_Type_create_struct

Usamos las direcciones obtenidas con MPI_Get_address para llenar el arreglo de offsets (nótese que el primer elemento tiene offset 0).

```
MPI_Aint a_addr, b_addr, n_addr;

MPI_Get_address(&a, &a_addr);
MPI_Get_address(&b, &b_addr);
MPI_Get_address(&n, &n_addr);

displacements_array[1]= b_addr - a_addr;
displacements_array[2]= n_addr - a_addr;
...
```



Creación del tipo derivado

La llamada incluye todos los argumentos. Debemos hacer un commit para la creación final.

```
//- Una vez inicializados los argumentos, llamamos
//- a MPI_Type_create_struct
MPI_Datatype my_new_type;
MPI_Type_create_struct(3, blocklengths_array,
    displacements_array, types_array, my_new_type_p);

//- Debemos hacer commit de los cambios para usar
//- la estructura con tipos derivados
MPI_Type_commit(my_new_type_p)
...
```

Broadcast de un tipo derivado

Mensaje usando tipos derivados

Podemos usar el tipo derivado en cualquier tipo de mensaje MPI, recordando siempre liberar recursos cuando no se estén utilizando.

```
MPI_Datatype my_new_type;
...
MPI_Bcast(a_p, 1, my_new_type, 0, MPI_COMM_WORLD);
/*  Como buffer pasamos un puntero al primer
 *  elemento del tipo. Reemplazamos MPI_XType por
 *  el identificador de mi tipo derivado my_new_type.
 */
...
MPI_Type_free(&my_new_type);  //Recuerden liberar mem.
```

Ejercicio (Lab 8, referirse a Canvas para mas info)

1. Descargue y compile el programa *mpi_trap4_do.c*
   a. Observe que la función Get_input llama a una función Build_mpi_type
2. La función Build_mpi_type tiene los enunciados en desorden. Organice los enunciados para que la función Build_mpi_type esté correcta y pueda crear una estructura de tipo derivado MPI.
3. Cambie la función Get_input para reemplazar los 3 broadcasts para cada input por uno solo que envíe la nueva estructura
4. Modifique la función Build_mpi_type y las llamadas correspondientes para que la estructura tenga el orden {b, n, a}. Asegúrese que el programa funciona.
5. Compare el performance de la actual versión de Trap (usando tipos derivados y broadcast) con la última versión anterior de Trap. Evidencie la mejora con capturas y calculando speedup.
6. Entregar lo solicitado (sus códigos y capturas de pantalla evidenciando el correcto funcionamiento).

REFERENCIAS:

1. **Pacheco, P**. "3. Distributed-Memory Programming with MPI" *An Introduction to Parallel Programming*. Morgan-Kaufmann. 2011.
2. **Trobec, R. Slivnik, B. Bulic, P. Robic, B**. "4. MPI Processes and Messaging" *Introduction to Parallel Computing – From Algorithms to Programming on State-of-the-Art Platforms*. Springer. 2018.
3. **Rauber, T. Rünger, G.** "5 Message-Passing Programming" Parallel Programming for Multicore and Cluster Systems. Springer. 2010.
4. **Nielsen, F.** Introduction to HPC with MPI for Data Science. Springer. 2016