UNIVERSIDAD DE BURGOS

Grado en Ingeniería Informática

Arquitecturas Paralelas

Curso 2024-2025

Practica 7

TIPOS DE DATOS DERIVADOS

AUTORES

Diego Urbaneja Portal

Hugo Gómez Martín

Nicolás Villanueva Ortega

ÍNDICE

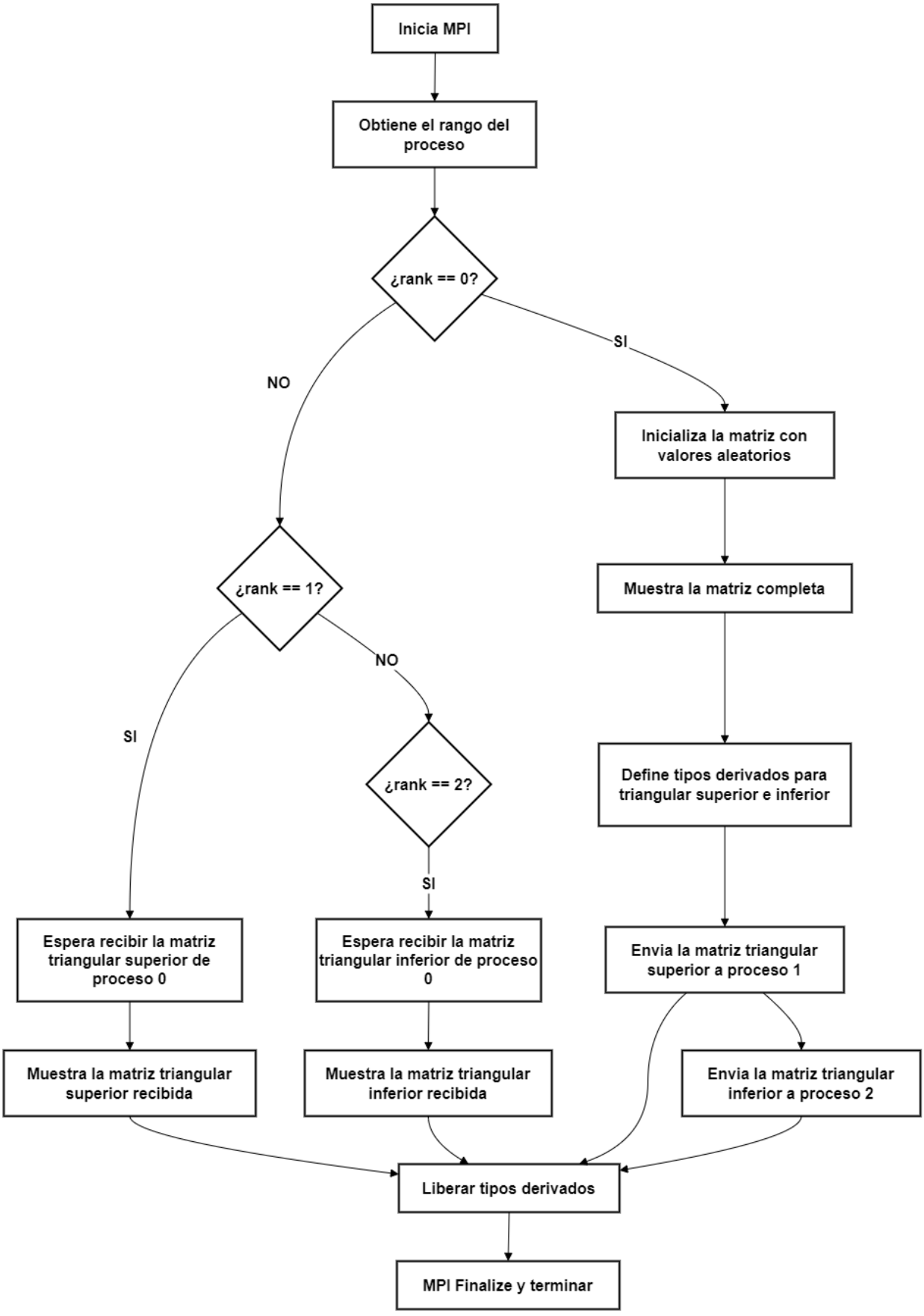
Introducción

Los **tipos de datos derivados** son estructuras personalizadas que **nos facilitan el manejo y envío de conjuntos de datos complejos**, los cuales van más allá de los tipos de datos primitivos como enteros o flotantes. Permiten una **mayor flexibilidad y eficiencia** en la comunicación entre procesos.

Esta práctica se centra en la **creación y uso de estos tipos derivados**, lo cual es esencial en situaciones donde se requiere enviar estructuras más complejas o grandes bloques de datos de una sola vez, optimizando así el rendimiento en aplicaciones paralelas**. La capacidad de MPI para manejar estos tipos de datos no solo simplifica el código, sino que también permite que las operaciones de comunicación sean más eficientes al reducir el número de envíos y adaptarse mejor a la arquitectura subyacente**.

Recomendada la visualización del documento con un 120% - 130% de zoom.

Flujograma


Inicia MPI
Obtiene el rango del proceso
¿rank == 0?
SI
NO
Inicializa la matriz con valores aleatorios
Muestra la matriz completa
Define tipos derivados para triangular superior e inferior
Envia la matriz triangular superior a proceso 1
Envia la matriz triangular inferior a proceso 2
¿rank == 1?
SI
NO
Espera recibir la matriz triangular superior de proceso 0
Muestra la matriz triangular superior recibida
¿rank == 2?
SI
Espera recibir la matriz triangular inferior de proceso 0
Muestra la matriz triangular inferior recibida
Liberar tipos derivados
MPI Finalize y terminar

Este flujograma representa el flujo de trabajo de nuestro programa que usa MPI (Interfaz de Paso de Mensajes) para distribuir una matriz entre múltiples procesos:

1. **Inicialización (A y B)**: El programa comienza inicializando el entorno MPI y determinando el rango (número de identificación) del proceso actual.
2. **Proceso 0 (C - H)**:
    - Si el proceso es el de rango 0, inicializa una matriz de tamaño N x N con valores aleatorios.
    - Define dos tipos de datos derivados usando MPI para representar la matriz triangular superior e inferior.
    - Envía la parte triangular superior de la matriz al proceso de rango 1 y la triangular inferior al proceso de rango 2.
3. **Proceso 1 (I - K)**:
    - Si el proceso es el de rango 1, espera recibir la matriz triangular superior enviada por el proceso 0.
    - Una vez recibida, la muestra en pantalla como "Triangular Superior".
4. **Proceso 2 (L - N)**:
    - Si el proceso es el de rango 2, espera recibir la matriz triangular inferior enviada por el proceso 0.
    - Una vez recibida, la muestra en pantalla como "Triangular Inferior".
5. **Liberación de recursos y finalización (O - P)**:
    - Todos los procesos liberan los tipos de datos derivados que definieron y finalizan el entorno MPI, concluyendo la ejecución del programa.

El programa divide una matriz generada por el proceso 0 y envía sus partes a otros procesos para mostrar las secciones triangular superior e inferior de forma separada. Esto se hace utilizando tipos de datos personalizados en MPI para simplificar la gestión de bloques de datos de la matriz.

Código

```cpp
#include <mpi.h>
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>

#define N 5  // Tamaño de la matriz

void initialize_matrix(float matrix[N][N]) {
    srand(time(0));
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j)
            matrix[i][j] = static_cast<float>(rand() % 10);  // Genera números del 0 al 9
}

void print_matrix(const float matrix[N][N], const char* name) {
    std::cout << "Matriz " << name << ":\n";
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j)
            std::cout << matrix[i][j] << " ";
        std::cout << "\n";
    }
}
```

```cpp
int main(int argc, char* argv[]) {
    MPI_Init(&argc, &argv); // Iniciamos MPI
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Obtenemos rango

    float matrix[N][N] = { 0 };  // Matriz de trabajo

    if (rank == 0) {
        // Inicializar la matriz con valores aleatorios
        initialize_matrix(matrix);
        std::cout << "[Proceso 0] Matriz completa antes de enviar:\n";
        print_matrix(matrix, "Completa");
    }

    // Crear dos tipos derivados para la matriz triangular superior e inferior
    MPI_Datatype upper_triangle, lower_triangle;

    // Vector de longitud de bloques y desplazamientos
    std::vector<int> lengths_upper(N), lengths_lower(N); // vect lonj
    std::vector<int> displs_upper(N), displs_lower(N); // vect desp

    for (int i = 0; i < N; ++i) {
        lengths_upper[i] = N - i;          // Cantidad de elementos en la fila de la triangular superior
        lengths_lower[i] = i + 1;          // Cantidad de elementos en la fila de la triangular inferior
        displs_upper[i] = i * N + i;       // Desplazamiento para triangular superior
        displs_lower[i] = i * N;           // Desplazamiento para triangular inferior
    }

    // Definir los tipos de datos derivados
    MPI_Type_indexed(N, lengths_upper.data(), displs_upper.data(), MPI_FLOAT, &upper_triangle);
    MPI_Type_indexed(N, lengths_lower.data(), displs_lower.data(), MPI_FLOAT, &lower_triangle);

    //Confirmación de los Tipos Derivados
    MPI_Type_commit(&upper_triangle);
    MPI_Type_commit(&lower_triangle);
```

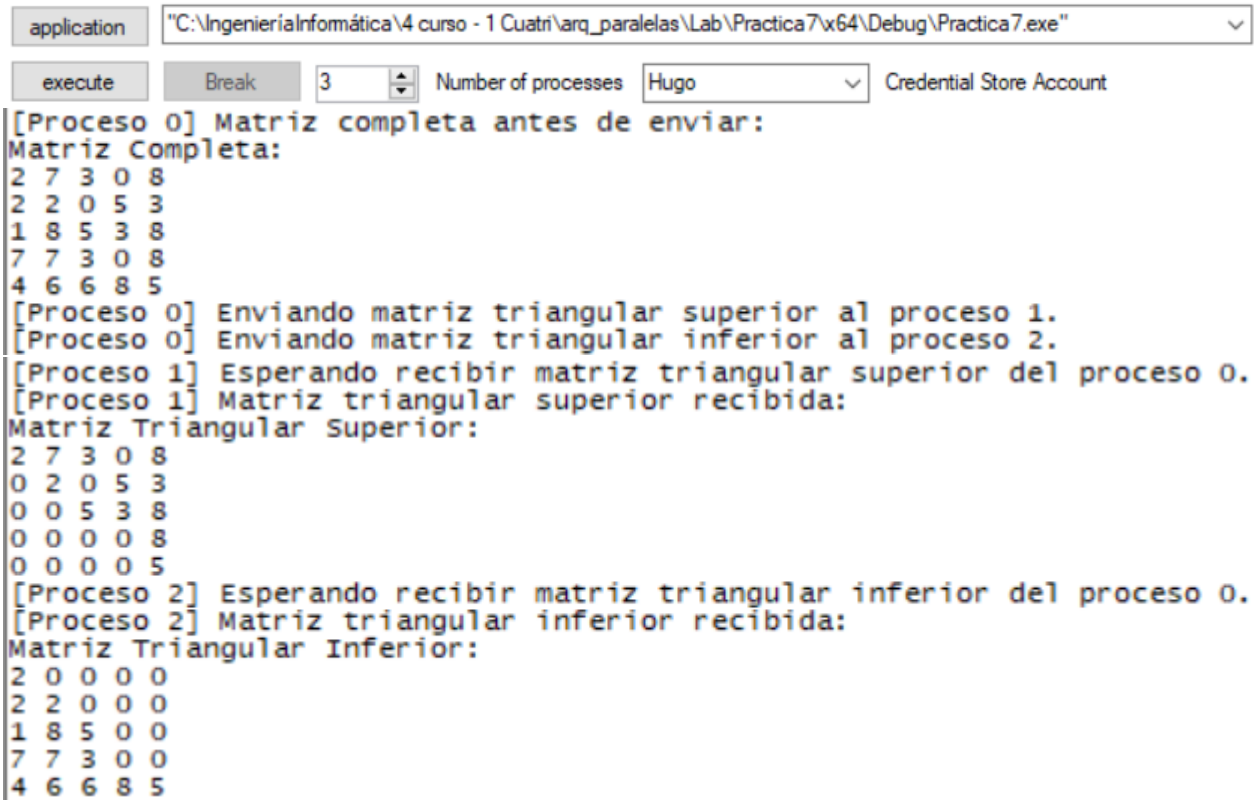
```
    if (rank == 0) {
        // Enviar la matriz triangular superior al proceso 1 y la inferior al proceso 2
        std::cout << "[Proceso 0] Enviando matriz triangular superior al proceso 1.\n";
        MPI_Send(matrix, 1, upper_triangle, 1, 0, MPI_COMM_WORLD);
        std::cout << "[Proceso 0] Enviando matriz triangular inferior al proceso 2.\n";
        MPI_Send(matrix, 1, lower_triangle, 2, 0, MPI_COMM_WORLD);
    }
    else if (rank == 1) {
        // Recibir la matriz triangular superior
        std::cout << "[Proceso 1] Esperando recibir matriz triangular superior del proceso 0.\n";
        MPI_Recv(matrix, 1, upper_triangle, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        std::cout << "[Proceso 1] Matriz triangular superior recibida:\n";
        print_matrix(matrix, "Triangular Superior");
    }
    else if (rank == 2) {
        // Recibir la matriz triangular inferior
        std::cout << "[Proceso 2] Esperando recibir matriz triangular inferior del proceso 0.\n";
        MPI_Recv(matrix, 1, lower_triangle, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        std::cout << "[Proceso 2] Matriz triangular inferior recibida:\n";
        print_matrix(matrix, "Triangular Inferior");
    }

    // Liberar los tipos de datos derivados
    MPI_Type_free(&upper_triangle);
    MPI_Type_free(&lower_triangle);

    MPI_Finalize();
    return 0;
}
```

Explicación del codigo: el programa utiliza MPI para distribuir eficientemente partes específicas (en nuestro caso la parte superior e inferior) de una matriz cuadrada de tamaño 5×5 entre tres procesos.

El **Proceso 0** inicializa la matriz con valores aleatorios y define dos tipos de datos derivados utilizando MPI_Type_indexed: uno para la **matriz triangular superior** y otro para la **matriz triangular inferior**. Estos tipos derivados permiten seleccionar y empaquetar únicamente los elementos correspondientes a cada triángulo sin necesidad de enviar la matriz completa o fragmentarla manualmente.

Posteriormente, el Proceso 0 envía la triangular superior al **Proceso 1** y la triangular inferior al **Proceso 2** mediante MPI_Send, mientras que los procesos receptores utilizan MPI_Recv para recibir y mostrar sus respectivas submatrices. Al final, se liberan los tipos derivados y se finaliza el entorno MPI.

Salida por Pantalla



En este caso con los procesos suficientes para enviar la matriz (proceso 0) y recibirla (procesos 1 y 2) es decir, 3 procesos en total podemos observar cómo el proceso 0 genera la matriz completa antes de enviar la matriz superior e inferior a el proceso 1 y 2 para que puedan ser mostradas.

Si introducimos un solo proceso solo se imprime la matriz original, si introducimos dos solo se mostrará la matriz original por el proceso 0 y la matriz triangular superior por el proceso 1 y si introducimos más de 3 procesos estos quedaran libres ya que no están asignados para realizar alguna tarea en esta práctica.

Cuestiones

- **En la práctica realizada, ¿qué problemas aparecen si se realiza reserva dinámica de memoria para crear espacio para las matrices? ¿Cómo afecta esto a la definición de nuevos tipos de datos?**

Desventajas de la Memoria Dinámica:

- Almacenamiento Ininterrumpido: Las matrices generadas de manera dinámica pueden repartirse en diferentes zonas de la memoria, lo que complica su gestión.
- Administrar Punteros: El trabajo con punteros puede ser complicado y susceptible a equivocaciones, como señalar lugares equivocados en la memoria.
- Calculación de Desplazamientos: Es más difícil identificar de manera exacta los datos en memoria, lo que dificulta la determinación de los tipos de datos en MPI.

Efecto sobre las Clases de Datos de MPI:

- Definición Compleja: Elaborar tipos de datos personalizados es más difícil, dado que los datos no se encuentran ordenados de manera constante.
- Incremento en los Errores: Al establecer la forma en que deben ser transmitidos los datos, se aumenta la probabilidad de incurrir en errores.
- Rendimiento: Debido al incremento de la complejidad, la comunicación entre procesos puede perder eficiencia.

- **Plantear otras situaciones en las que sea de utilidad la definición de tipos de datos derivados.**
    - Envío de Estructuras Complejas.
    - Trabajo con Submatrices.
    - Manejo de Datos Dispersos.
    - Mejora del Rendimiento.
    - Compatibilidad entre Sistemas Diferentes.
    - Operaciones de Reducción Personalizadas.
    - Integración con Librerías o Programas Externos.

Bibliografía

Hemos obtenido la información para la realización de la práctica del **guion** proporcionado por el profesor.

Para la resolución de dudas con el código hemos empleado chat **GPT o1-mini**.