

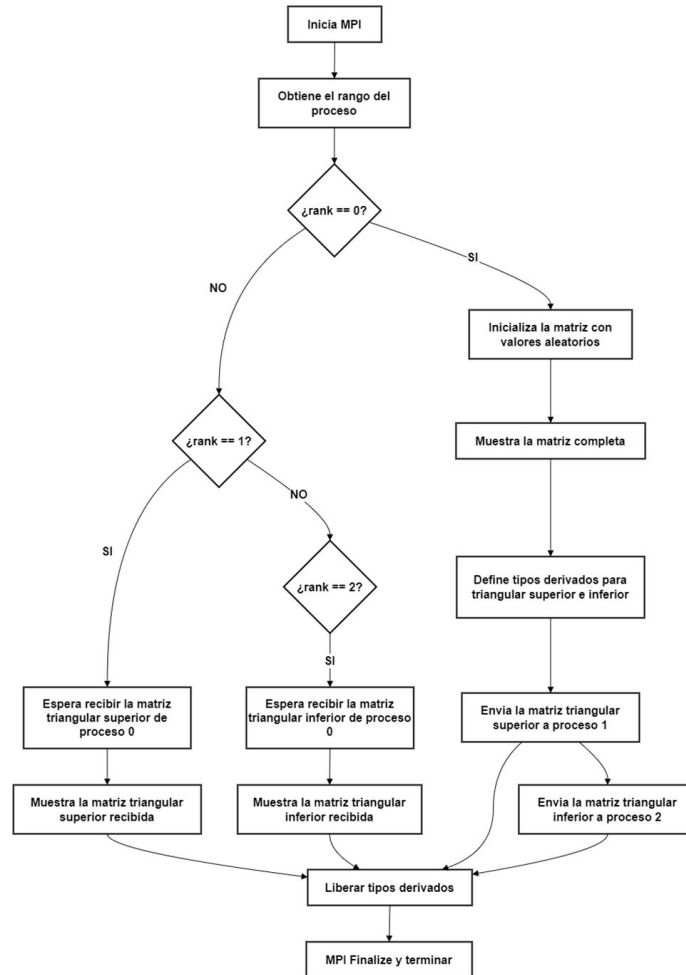
Práctica 7: Tipos de datos derivados

DIEGO URBANEJA
HUGO GÓMEZ
NICOLÁS VILLANUEVA

ÍNDICE

- **FLUJOGRAMA**
- **CÓDIGO DEL PROGRAMA**
- **EJECUCIÓN Y SALIDA POR PANTALLA**
- **CUESTIONES PLANTEADAS**

FLUJOGRAMA



CÓDIGO DEL PROGRAMA

```
#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";
    }
}
```

```
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
    }
}
```

CÓDIGO DEL PROGRAMA

```
// 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;
```

EJECUCIÓN Y SALIDA POR

PA

application "C:\Ingeniería Informática\4 curso - 1 Cuatr\arq_paralelas\Lab\Practica7\x64\Debug\Practica7.exe" 

execute Break 3   Number of processes Hugo  Credential Store Account

```
[Proceso 0] Matriz completa antes de enviar:
Matriz Completa:
2 7 3 0 8
2 2 0 5 3
1 8 5 3 8
7 7 3 0 8
4 6 6 8 5
[Proceso 0] Enviando matriz triangular superior al proceso 1.
[Proceso 0] Enviando matriz triangular inferior al proceso 2.
[Proceso 1] Esperando recibir matriz triangular superior del proceso 0.
[Proceso 1] Matriz triangular superior recibida:
Matriz Triangular superior:
2 7 3 0 8
0 2 0 5 3
0 0 5 3 8
0 0 0 0 8
0 0 0 0 5
[Proceso 2] Esperando recibir matriz triangular inferior del proceso 0.
[Proceso 2] Matriz triangular inferior recibida:
Matriz Triangular Inferior:
2 0 0 0 0
2 2 0 0 0
1 8 5 0 0
7 7 3 0 0
4 6 6 8 5
```

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?**
 - Disposición no contigua: Las matrices dinámicas pueden no estar almacenadas de forma continua en memoria.
 - Gestión de Punteros: Incrementa la complejidad y riesgo de errores (fugas, corrupción).
 - Cálculo de desplazamientos: Difícil especificar desplazamientos precisos para tipos derivados.
 - Tipos de datos más complejos: Requiere definiciones más elaboradas, reduciendo la eficiencia

Impacto en los Tipos de Datos de MPI:

- Definición Compleja: Es más complicado crear tipos de datos personalizados porque los datos no están en una secuencia fácil de seguir.
 - Mayor Posibilidad de Errores: Hay más riesgos de equivocarse al especificar cómo se deben enviar los datos.
 - Rendimiento: Puede hacer que la transferencia de datos sea menos eficiente.
- **Plantear otras situaciones en las que sea de utilidad la definición de tipos de datos derivados.**
 - Estructuras de datos complejas
 - Matrices multidimensionales
 - Datos no contiguos o dispersos
 - Comunicación eficiente