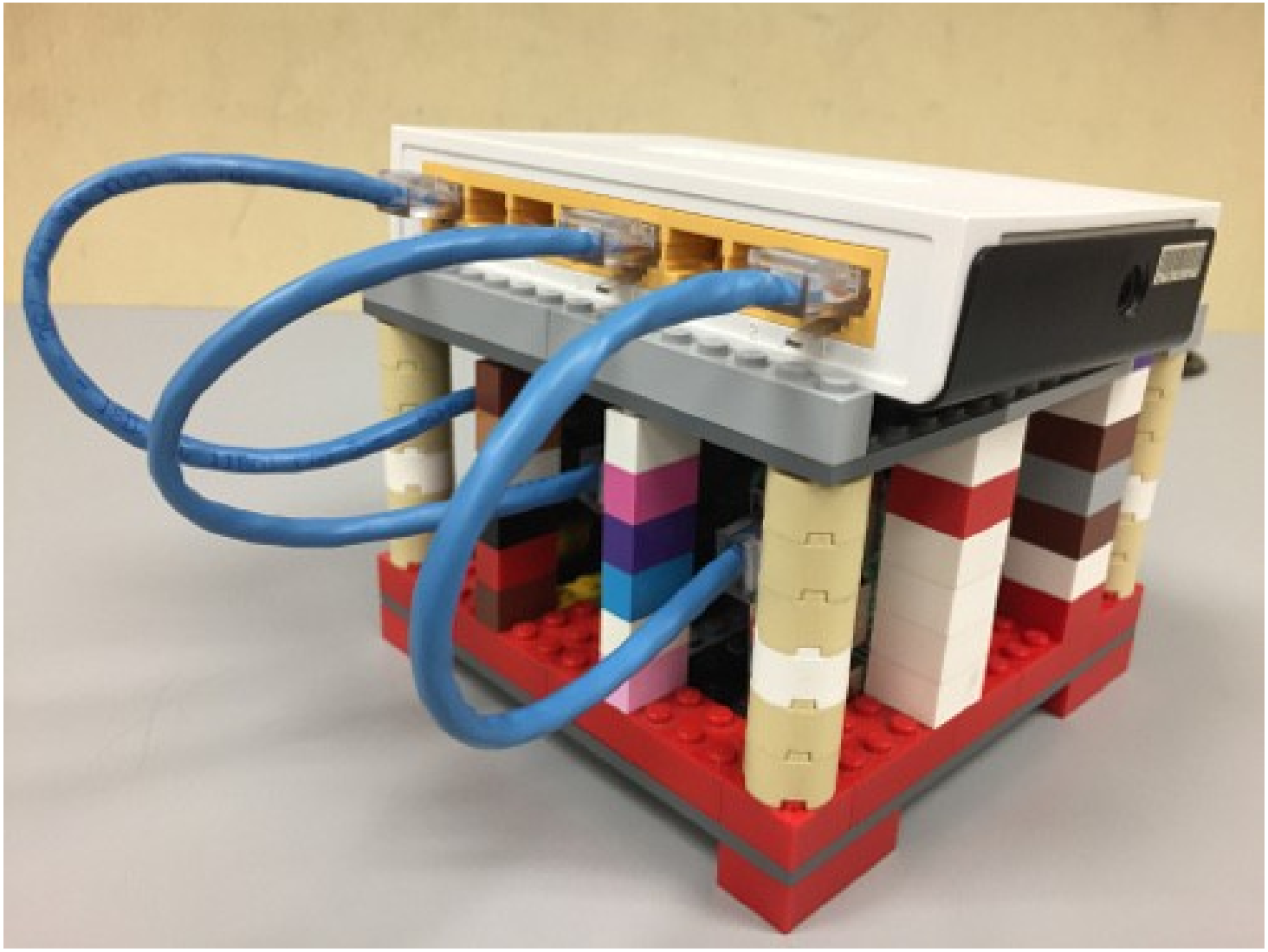
MPI - “Programación”



Rogelio Ferreira Escutia

Funciones básicas

MPI - Biblioteca

- **Cargar biblioteca**

```
#include <mpi.h>
```

MPI - Inicializar

- **Inicializar:**

```
// Initialize the MPI environment
MPI_Init(NULL, NULL);
```

"MPI Hello World", http://mpitutorial.com/tutorials/mpi-hello-world/, noviembre 2018

MPI - Detectar

- **Detectar el número de procesadores:**

```
MPI_Comm_size(
    MPI_Comm communicator,
    int* size)
```

MPI - Identificador

- **Detectar el número lógico que corresponde a cada procesador. Este valor siempre empieza en cero y alcanza un valor máximo igual al número de procesadores menos uno.**

```
MPI_Comm_rank(
        MPI_Comm communicator,
        int* rank)
```

MPI - Enviar

- **Envía un mensaje a otro procesador. El procesador origen espera que el procesador destinatario haya recibido el mensaje antes de continuar trabajando:**

```
MPI_Send(&buf, count, datatype, dest, tag, comm)
```

MPI - Recibir

- **Envía un mensaje a otro procesador. El procesador origen espera que el procesador destinatario haya recibido el mensaje antes de continuar trabajando:**

```
MPI_Recv(&buf, count, datatype, source, tag, comm, &status)
```

MPI - Finalizar

- **Cierra el ambiente de trabajo en paralelo una vez finalizado el trabajo:**

```
MPI_Finalize()
```

"MPI Hello World", http://mpitutorial.com/tutorials/mpi-hello-world/, noviembre 2018

Programación con MPI

MPI

- **Hola Mundo:**

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name, &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d processor
s\n",
            processor_name, world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```

"MPI Hello World", http://mpitutorial.com/tutorials/mpi-hello-world/, noviembre 2018

MPI

- **Tipos de datos:**

MPI	C	MPI	Fortran
MPI_CHAR	signed char	MPI_INTEGER	INTEGER
MPI_SHORT	signed short int	MPI_REAL	REAL
MPI_INT	signed int	MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_LONG	signed long int	MPI_COMPLEX	COMPLEX
MPI_UNSIGNED_CHAR	unsigned char	MPI_LOGICAL	LOGICAL
MPI_UNSIGNED_SHORT	unsigned short int	MPI_CHARACTER	CHARACTER
MPI_UNSIGNED	unsigned int	MPI_BYTE	
MPI_UNSIGNED_LONG	unsigned long int	MPI_PACKED	
MPI_FLOAT	float		
MPI_DOUBLE	double		
MPI_LONG_DOUBLE	long double		
MPI_BYTE			
MPI_PACKED			

MPI – Encontrar número mayor (1)

- **Inicialización:**

```
1  #include <mpi.h> /*Biblioteca de MPI*/
2  #include <stdio.h>
3  #include <string.h>
4
5  int main(int argc, char **argv) {
6
7      int numeros[50] =
       {1,9,8,6,4,3,2,12,322,45,67,98,21,23,34,33,231,647,987,348,549,11,70,77,194,237,470,5,58,729,498,315,94,6
       29,641,861,302,27,57,13,77,564,209,44,873,846,298,741,138,500};
8
9      int id_proceso;
10     int proceso;
11     int total;
12     int particion;
13     int mayor = 0;
14     int enteromaquina;
15
16     char respuesta[50];
17     char maquina[MPI_MAX_PROCESSOR_NAME];
18
19     MPI_Status estado;
20
21     MPI_Init(&argc, &argv); /*Iniciar MPI*/
22     MPI_Comm_rank(MPI_COMM_WORLD, &id_proceso); /*Obtener el identificador del proceso*/
23     MPI_Comm_size(MPI_COMM_WORLD, &total); /*Obtener el número total de procesos*/
24     MPI_Get_processor_name(maquina, &enteromaquina);
```

MPI – Encontrar número mayor (2)

- **Dividir el arreglo y procesar localmente:**

```
25
26      int longitud_arreglo = (sizeof(numeros)/sizeof(numeros[0]));
27
28      particion = (id_proceso == 0) ? (longitud_arreglo / total) + (longitud_arreglo % total) : (int)
        longitud_arreglo / total;
29
30      int inicio = particion * id_proceso;
31      int fin = inicio + particion;
32
33      printf("Proceso %d corriendo en %s analizando %d numeros\n", id_proceso, maquina, particion);
34
35      for(int i = inicio; i < fin; i++)
36      {
37          int aux = mayor;
38          mayor = (mayor > numeros[i]) ? mayor : numeros[i];
39          printf("Comparando %d y %d. El mayor es: %d\n", aux, numeros[i], mayor);
40      }
```

MPI – Encontrar número mayor (3)

- **Procesamiento local:**

```
41
42      if(id_proceso == 0)
43      {
44          printf("Mayor en el proceso %d: %d\n", id_proceso, mayor);
45
46          for(proceso = 1; proceso < total; proceso++)
47          {
48              MPI_Recv(respuesta,  sizeof(respuesta), MPI_BYTE, proceso, 1, MPI_COMM_WORLD, &estado);
49
50              int valor;
51              int aux = mayor;
52
53              sscanf(respuesta, "%d", &valor);
54
55              mayor = (mayor > valor) ? mayor : valor;
56
57              printf("Comparaciones finales. Comparando %d y %d. EL mayor es: %d\n", aux, valor, mayor);
58          }
59
60          printf("El numero mayor es: %d\n", mayor);
61      }
```

MPI – Encontrar número mayor (4)

- **Procesamiento en nodos remotos:**

```
62      else
63      {
64          printf("Mayor en el proceso %d: %d\n", id_proceso, mayor);
65
66          sprintf(respuesta, "%d", mayor);
67
68          MPI_Send(respuesta, strlen(respuesta) + 1, MPI_BYTE, 0, 1, MPI_COMM_WORLD);
69      }
70
71      MPI_Finalize(); /*Terminar MPI*/
72
73      return 0;
74  }
```

Rogelio Ferreira Escutia

Instituto Tecnológico de Morelia
Departamento de Sistemas y Computación

Correo: ***rogelio@itmorelia.edu.mx***
rogeplus@gmail.com

Página Web: ***http://sagitario.itmorelia.edu.mx/~rogelio/***
http://www.xumarhu.net/

Twitter: ***http://twitter.com/rogeplus***
Facebook: ***http://www.facebook.com/groups/xumarhu.net/***