Cómputo Paralelo y Distribuido.
Ejercicios de MPI
ITESM
Junio 2015

Ejercicios.

1. Iniciando MPI. Hola Mundo con MPI.

```
/*
Parallel and Distributed Computing Class
MPI

Practice 1 : Hola Mundo
Name      :
*/


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

int main( argc, argv )
int argc;
char **argv;
{
MPI_Init( &argc, &argv );
printf( "Hello world with MPI\n" );
MPI_Finalize();
return 0;
}
```

* compilarlo como: mpicc fuente.c –o ejecutable
* ejecutarlo como: mpirun -np 2 ./ejecutable

2. Hola mundo desde el procesador, rango, total de procesos y nombre del procesador.

```
/*
Parallel and Distributed Computing Class
MPI

Practice 2 : Hola Mundo con varios procesadores
Name      :
*/
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char** argv) {
MPI_Init(NULL, NULL);
  // Get the number of processes //procesos
  int world_size1;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size1);
```

```
// Get the rank of the process //
  int world_rank1;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank1);
  // Get the name of the processor
  char processor_name1[MPI_MAX_PROCESSOR_NAME];
  int name_len;
  MPI_Get_processor_name(processor_name1, &name_len);
  // Print off a hello world message
  printf("Hola Mundo!!! desde el procesador %s, rank %d out of %d
processors\n",
         processor_name1, world_rank1, world_size1);
  // Finalize the MPI environment. No more MPI calls can be made after this
  MPI_Finalize();
}
```

Preguntas de reflexión:
*¿Para que sirve la instrucción MPI_Get_processor_name?

3. Envío y Recepción de un número entre dos procesos (Bloqueante).

```
/*
Parallel and Distributed Computing Class
MPI

Practice 3 : Envío y Recepción de un número entre dos procesos
Name      :
*/

 #include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

 int main(int argc, char** argv) {
  // Initialize the MPI environment
  MPI_Init(NULL, NULL);
  // Find out rank, size
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);
// We are assuming at least 2 processes for this task
  if (world_size < 2) {
    printf("World size must be greater than 1 for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
  }
int number;
```

```
  if (world_rank == 0) {
    // If we are rank 0, set the number to -1 and send it to process 1
    number = -1;
    MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
  } else if (world_rank == 1) {
    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
    printf("Process 1 received number %d from process 0\n", number);
  }
  MPI_Finalize();
}
```

Preguntas de reflexión.
* Probar el programa con más de 2 procesos. Que sucede?

4. Envío y Recepción de un número entre dos procesos de forma No bloqueante. Cuál es la diferencia entre procesos bloqueantes y no bloqueantes?".

5. Ping-Pong. Probarlo con 2 y 4 procesos y obtener tiempo de procesamiento en cada uno de ellos con MPI_Wtime(). Que resultados obtienes?.

```
 /*
Parallel and Distributed Computing Class
MPI

Practice 3 : Envío y Recepción de un número entre dos procesos
Name      :
*/
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define PING_PONG_LIMIT 4

int main(int argc, char** argv) {
MPI_Init(NULL, NULL);

  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  // We are assuming at least 2 processes for this task
  if (world_size != 2) {
    printf("World size must be two for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
```

```
    }

    int ping_pong_count = 0;
    int partner_rank = (world_rank + 1) % 2;
    while (ping_pong_count < PING_PONG_LIMIT) {
      if (world_rank == ping_pong_count % 2) {
        // Increment the ping pong count before you send it
        ping_pong_count++;
        MPI_Send(&ping_pong_count, 1, MPI_INT, partner_rank, 0,
  MPI_COMM_WORLD);
        printf("%d sent and incremented ping_pong_count %d to %d\n",
  world_rank, ping_pong_count, partner_rank);
      } else {
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0,
  MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);
        printf("%d received ping_pong_count %d from %d\n", world_rank,
  ping_pong_count, partner_rank);
      }
    }
    MPI_Finalize();
  }
```

El resultado debe ser similar a:

```
0 sent and incremented ping_pong_count 1 to 1
1 received ping_pong_count 1 from 0
1 sent and incremented ping_pong_count 2 to 0
1 received ping_pong_count 3 from 0
1 sent and incremented ping_pong_count 4 to 0
0 received ping_pong_count 2 from 1
0 sent and incremented ping_pong_count 3 to 1
0 received ping_pong_count 4 from 1
```

6. Que hacen las funciones: MPI_Scatter, MPI_Gather, MPI_Allreduce, MPI_Allgather, MPI_Barrier y MPI_Reduce?. Menciona en que casos se utiliza. Cuál es la función prototipo de cada una de ellas?.

7. Calcular la suma de números de un arreglo unidimensional utilizando MPI_Scatter y MPI_Gather. Pruébalo con 2 y 4 procesos. ¿Qué resultados obtienes?.

8. Puedes calcular la suma de números del ejemplo 7 utilizando la función MPI_Reduce para obtener el resultado ?, Si la respuesta es sí ¿Cómo sería?, de lo contrario fundamenta tu respuesta.

Bibliografía:

- Kendall, Wesley (2013). Beginning MPI, An introduction in C.

  Tiene su página web en: http://mpitutorial.com