Universidad Nacional de San Antonio Abad del Cusco

Departamento Académico de Informática

ALGORITMOS PARALELOS Y DISTRIBUIDOS

Práctica Nº 5

CONFIGURACION Y EJECUCION DE PROGRAMAS PARALELOS EN MPI

1. OBJETIVO.

- Configurar la librería MPI de Microsoft para desarrollar programas paralelos en C/C++ con Visual Studio.
- Conocer la estructura básica de un programa paralelo con MPI.
- Conocer las funciones básicas de un programa paralelo con MPI.

2. INTRODUCCION.

MPI (Message Passing Interface) es un estándar para programación paralela basada en el envío de mensajes en arquitecturas con memoria distribuida. Proporciona una librería de funciones para C, C++ o Fortran que son empleadas en los programas para comunicar datos entre procesos. MPI es la primera librería de paso de mensajes estándar y portable, especificada por consenso por el MPI Forum (http://www.mpi-forum.org) con unas 40 organizaciones participantes, definiendo un modelo que permita desarrollar programas que puedan ser migrados a diferentes computadores paralelos.

2.1. CARACTERÍSTICAS DE MPI.

Siguiendo el modelo SPMD, el usuario escribirá su aplicación como un proceso secuencial del que se lanzarán varias instancias que cooperan entre sí. Los procesos invocan diferentes funciones MPI que permiten iniciar, gestionar y finalizar procesos MPI comunicar datos entre dos procesos, realizar operaciones de comunicación entre grupos de procesos crear tipos arbitrarios de datos

2.2. FUNCIONES BASICAS DE MPI.

Cualquier programa paralelo con MPI puede implementarse con tan sólo 6 funciones, aunque hay muchas más funciones para aspectos avanzados. Todas ellas empiezan por MPI_ y obligan a que los programas MPI tengan #include "mpi.h".

Los programas MPI deben ser obligatoriamente inicializados y finalizados en MPI (MPI_Init, MPI_Finalize).

Los procesos en ejecución pueden saber cuántos procesos MPI forman parte de un grupo de procesos -communicator en la terminología MPI- (MPI_Comm_size) y qué número de orden -empezando por 0- tiene en ese grupo de procesos (MPI_Comm_rank).

Los mensajes punto a punto deben ser enviados explícitamente por el emisor y recibidos explícitamente por el receptor (más adelante se explicarán las operaciones de comunicación colectivas), para lo cual pueden emplearse dos funciones básicas (MPI_Send y MPI_Recv).

- **Archivo cabecera:**

  #include <mpi.h>

- **Formato de las funciones:**

  codigo_error = MPI_nombre(parámetros ...)

- **Inicialización:**

  int MPI_Init (int *argc , char **argv)

- **Comunicador:**

  Conjunto de procesos que se intercomunican. Por defecto podemos utilizar MPI_COMM_WORD , en cuyo caso el grupo de procesos es el conjunto de procesos lanzados conjuntamente para resolver un problema

- **Identificación de procesos:**

  MPI_Comm_rank (MPI_Comm comm , int *rank)

- **Procesos en el comunicador:**

  MPI_Comm_size (MPI_Comm comm , int *size)

- **Finalización:**

  int MPI_Finalize ( )

- **Mensajes punto a punto**

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

  MPI_Recv(buf, count, datatype, source, tag, comm)

3. DESARROLLO DE LA PRÁCTICA

3.1. CONFIGURACION DE MS MPI.

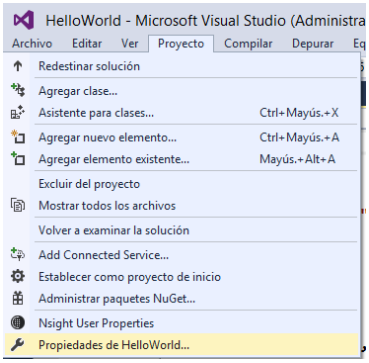
1. Descargar los instaladores de MS-MPI https://msdn.microsoft.com/en-us/library/bb524831.aspx. Éstos son:
   - msmpisetup.exe
   - msmpisdk.msi

   luego, instalarlos en los directorios por defecto.

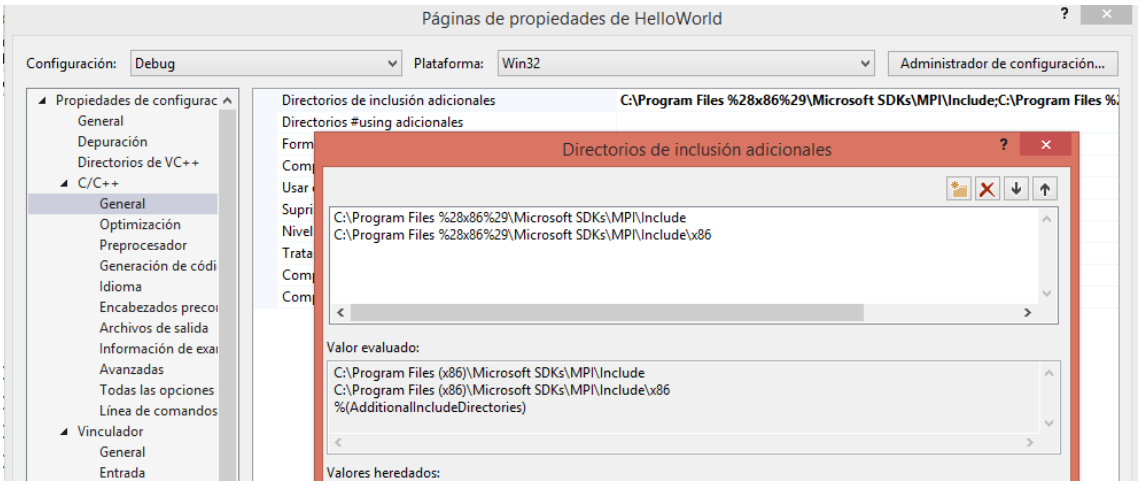
2. Agregar a la variable PATH del sistema la siguiente dirección:
   C:\Program Files\Microsoft MPI\Bin

3. Abrir Visual Studio y crear un proyecto de Aplicación de Consola en C/C++ vacío. Nombre al proyecto “HelloWorld”.

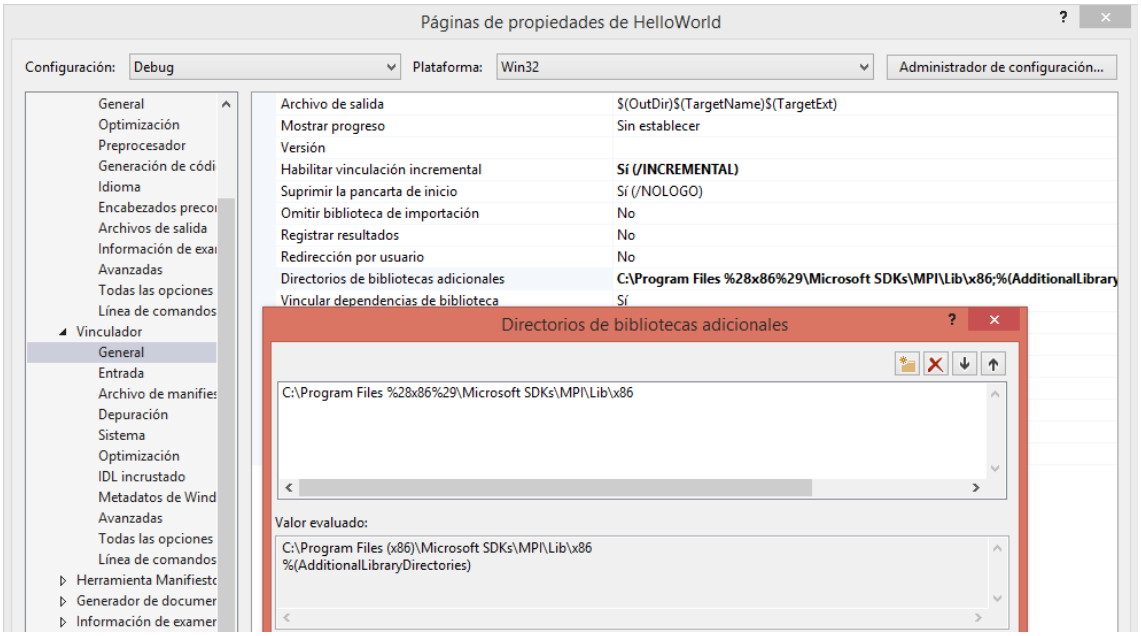
4. Ingrese al menú Proyecto –> Propiedades de HelloWorld, como se muestra en la figura.



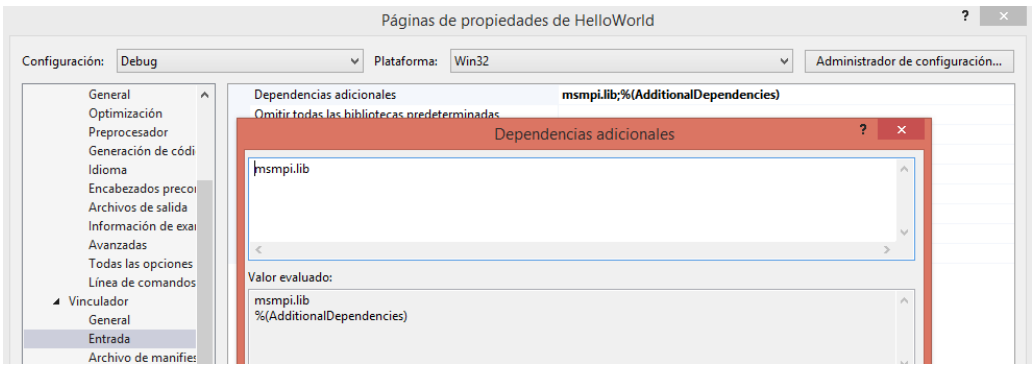
5. Configure los directorios de inclusión adicionales como se muestra en la figura.



6. Configure Directorio de Biblioteca adicionales como se muestra en la figura siguiente.



7. Configurar Dependencias Adicionales digitando msmpi.lib como se muestra en la figura.



3.2. COMPILAR Y EJECUTAR UN PROGRAMA PARALELO CON MPI

1. Digitar el siguiente código y compilar.

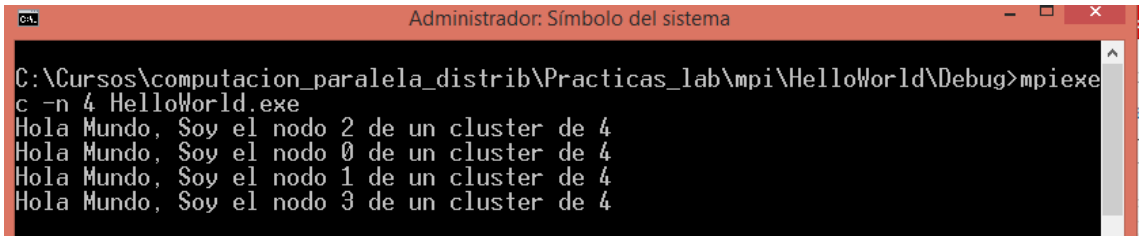
```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char **argv)
{
   int rank;
   int size;
   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   printf("Hola Mundo, Soy el nodo %d de un cluster de %d\n",
              rank,size);
   MPI_Finalize();
   return 0;

 }
```

2. Para ejecutar el programa paralelo abrir una ventana de CMD y ubicarse en el directorio del programa ejecutable. Luego digitar:

   mpiexec –n 4 HelloWorld.exe

   y obtendremos el resultado siguiente:



3.3. Ejercicio. El siguiente ejercicio muestra la ejecución del ciclo for paralelo. En cada proceso se ejecuta el ciclo completo.

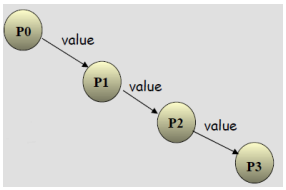
```
#include <mpi.h>
#include <stdio.h>
```

```
#include <iostream>
using namespace std;
int main(int argc, char **argv)
{
   int i, pid, npr;
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &npr);
   MPI_Comm_rank(MPI_COMM_WORLD, &pid);
   for (i = 0; i < 5; i++)
      cout << "Proceso: " << pid << " iteracion: " << i << endl;
    MPI_Finalize();
    return 0;
}
```

3.4. Ejercicio. En este ejercicio se muestra la comunicación punto a punto entre dos procesos utilizando MPI_Send y MPI_Recv bloqueante.

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv)
{
  int rank, value, size; MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  if (rank == 0)
  {
    value = 100;
    MPI_Send(&value, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
    printf("Envie valor %d", value);
  }
  else
  {
    MPI_Recv(&value, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
    printf("soy el proceso %d de %d y recibi el valor %d", rank, size,
           value);
  }
   MPI_Finalize();
   return 0;
}
```

3.5. Ejercicio. El siguiente programa paralelo solicita ingresar un entero por teclado, el proceso 0 envía este dato al proceso 1, éste lo recibe y lo envía al proceso 2 y así sucesivamente, el último proceso sólo recibe el dato.




```
#include <stdio.h>
#include "mpi.h"
#include <iostream>
int main(int argc, char **argv)
{
      int rank, value, size; MPI_Status status;
      MPI_Init(&argc, &argv);
      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

```
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        if (rank == 0)
        {
          printf("Ingrese un nro entero: ");
          std::cin >> value;
          printf("Soy el proceso %d y envio el valor %d al proceso %d",
                  rank, value, rank + 1);
          MPI_Send(&value, 1, MPI_INT, rank + 1, 0, MPI_COMM_WORLD);
        }
        else
        {
          MPI_Recv(&value, 1, MPI_INT, rank - 1, 0, MPI_COMM_WORLD,
                   &status);
          printf("Soy el proceso %d y recibi el valor %d de %d \n", rank,
                   value, rank-1);
          if (rank < size - 1)
          {
           printf("Soy el proceso %d y envio el valor %d al proceso %d",
                    rank, value, rank + 1);
           MPI_Send(&value, 1, MPI_INT, rank + 1, 0, MPI_COMM_WORLD);
          }
        }
        MPI_Finalize();
        return 0;
}
```

4. TAREA

4.1. Escribir un programa paralelo en MPI en el cada proceso calcule la suma de los elementos de un vector de tamaño N y muestre su resultado indicando el número del proceso que ha hecho la suma .

4.2. Escribir un programa paralelo para que el proceso 0 envíe dos números reales al proceso 1 y éste devuelva el resultado de la suma al proceso 0 que lo imprimirá.

5. BIBLIOGRAFIA.

- Castro Rodero Ivan, G. B. F. Programación y Computación Paralelas.: Universidad Oberta de Catalunya.
- Gropp William, L. E., Skjellum Anthony. (2014). Using MPI Portable Parallel Programming with the
- J., Q. M. (2003). Parallel Programming in C with MPI and OpenMP. Singapore: Mc Graw Hill.
- Peter, P. (1997). Parallel Programming with MPI. San Francisco California: Morgan Kaufmann Publishers.
- Peter, P. (2011). An Introduction to Parallel Programming. Burlington USA: Elsevier Inc.
- Wilkinson Barry, A. M. (2004). Parallel Programming (Second Edition): Prentice Hall.

Cusco diciembre de 2019