# SISTEMAS COMPUTADORES DE ALTAS PRESTACIONES

## PRACTICA 3


ALEJANDRO GORDILLO PEDRAZA

# Contenido

# RESUMEN

En esta practica vamos a tratar de recrear un benchmark y ver su escalabilidad en el tiempo, haciendo uso de procesadores multinúcleo y más potentes que en las practicas anteriores. También para la misma haremos uso de MSMPI, que es la implementación en C de Microsoft de la interfaz estándar del paso de mensajes de MPI.

Para realizar la Practica necesitaremos un ordenador con Windows, SDK Microsoft MPI y Visual Studio.

# Configurar proyecto con MSMPI en Visual Studio

## Instalar SDK Microsoft MPI

Para empezar con nuestro proyecto, lo primero que deberemos hacer es descargar el kit de desarrollo MPI de Microsoft. Para ello iremos a su pagina oficial y lo descargaremos (link), y seguimos los pasos de la instalación.

## Instalación del entorno de ejecución Microsoft MPI

Para poder ejecutar los programas con la librería MSMPI, deberemos tener instalado el entorno de ejecución MPI. Más en concreto el programa mpiexec.exe.

Este programa es el lanzador de aplicaciones MPI que nos permitirá iniciar las aplicaciones que desarrollemos en nuestra práctica.

Para conseguirlo deberemos descargarlo (link) e instalarlo, para instalarlo ejecuta el archivo y sigue las instrucciones.

## Configuración del Proyecto en Visual Studio

Una vez tenemos las librerías y el programa para la ejecución de estas, pasaremos a hacer uso de visual studio. Que para quien no lo tenga aun en su computador dejo el link de descarga. Visual Studio

Llegados a este punto ya podremos crear y desarrollar nuestro proyecto. Para ello vamos a abrir visual studio y comenzamos:

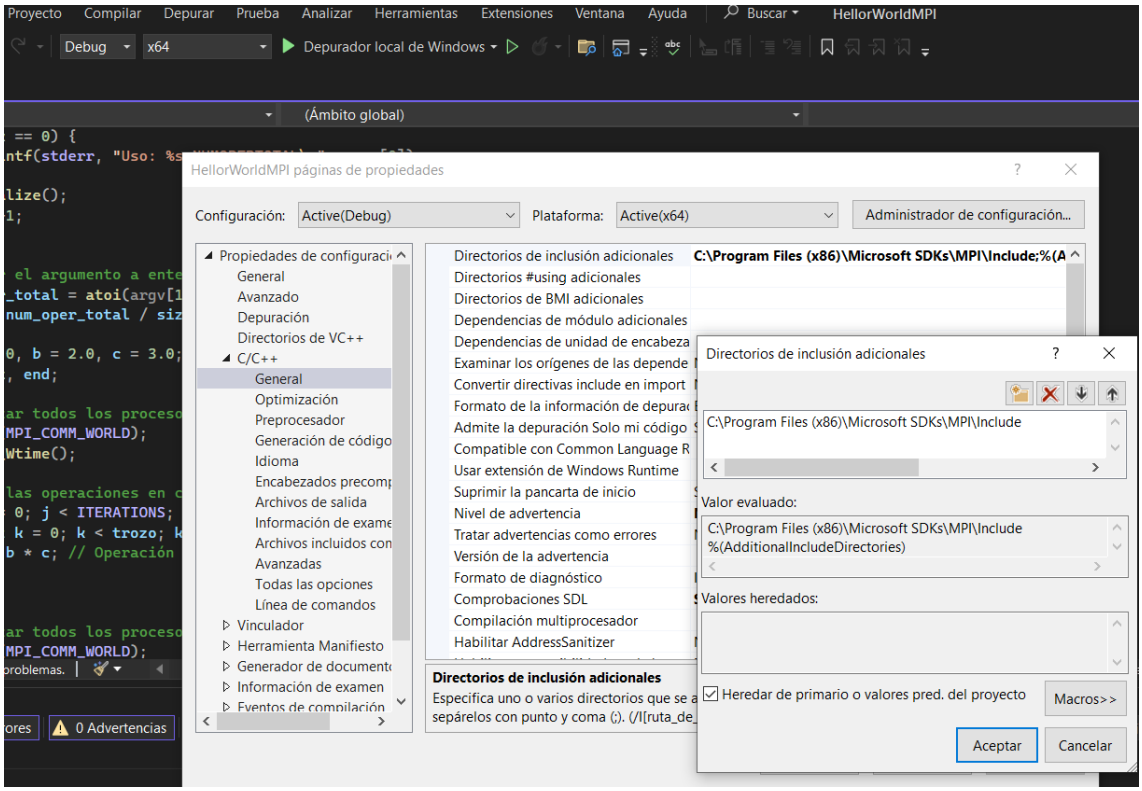
1º Crear nuevo proyecto:

> Elegiremos un proyecto de aplicación de consola, y le pondremos de nombre HelloWorldMPI. Seleccionaremos la casilla de colocar solución y proyecto en el mismo directorio, y crear.

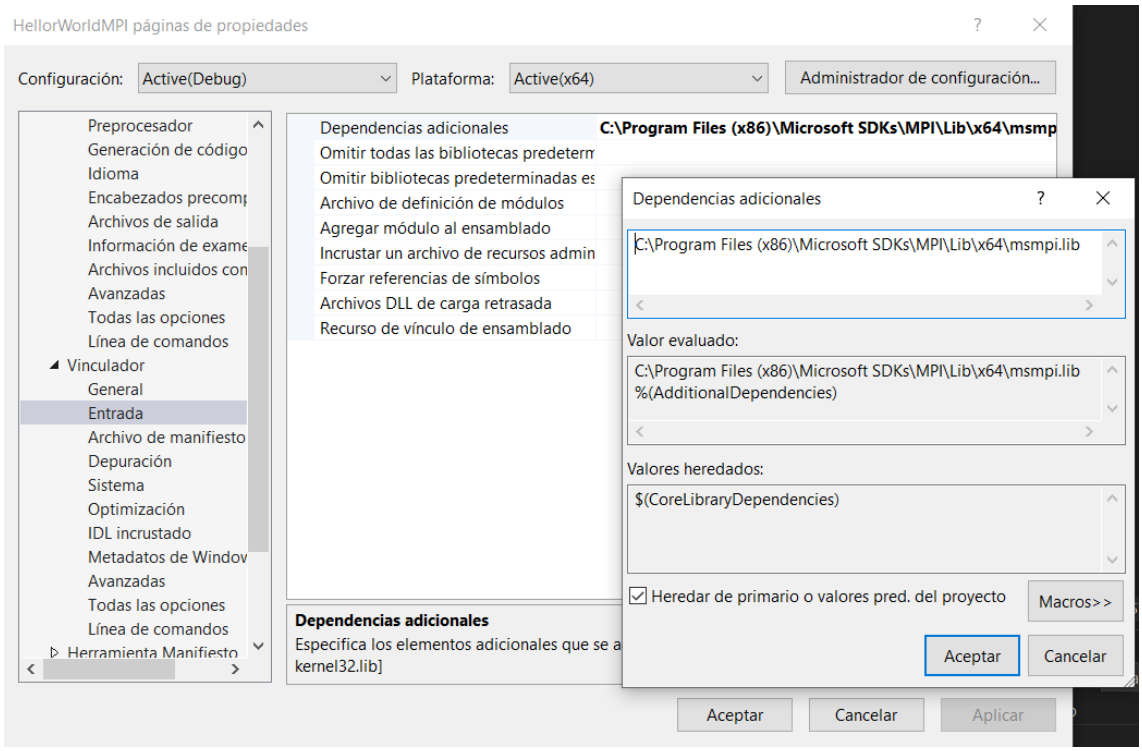
2º Añadir dependencias del Proyecto con MSMPI

> Para este paso deberemos poner el Debug en x64, lo encontraremos en la barra superior del programa.

Después deberemos irnos a proyecto-propiedades de proyecto-C/C++-General-Directorios de Inclusión adicional y añadiremos la ruta donde tenemos instalado el SDKs.




Por último, añadiremos la ruta a la librería de msmpi.lib en la misma venta de propiedades de proyecto Vinculador-Entrada- Dependencias adicionales



## Configuración del proyecto Hello World

Vamos a implementar el archivo fuente del proyecto. Para ello vamos a sustiuir el código que nos viene por defecto, por el código que facilitare a continuación, que ya implementa las funciones multiproceso al hello world.

```
#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

int main(int argc, char** argv) {

  // Inicializar MPI

  MPI_Init(&argc, &argv);

  // Obtener el número de procesos

  int num_procs;

  MPI_Comm_size(MPI_COMM_WORLD, &num_procs);


  // Obtener el rango del proceso actual

  int rank;

  MPI_Comm_rank(MPI_COMM_WORLD, &rank);


  // Imprimir el número de núcleos por proceso

  printf("Hola Mundo desde el Proceso %d de %d totales\n", rank,
num_procs);


  // Finalizar MPI

  MPI_Finalize();

  return 0; }
```

## Compilación del Proyecto

Para compilar nuestro proyecto en visual studio y posteriormente ejecutarlo con mpi, deberemos seguir los siguientes pasos:

1º Vamos a la barra de herramientas y clicamos en Compilar- Compilar Solución

2º Buscamos la carpeta donde se ha compilado nuestro proyecto, en nuestro caso se encuentra en el siguiente directorio:
**C:\Users\Rebu\source\repos\HellorWorldMPI\x64\Debug**

3º Abrimos la powershel o el cmd, en nuestro caso usamos la powershell por temas de gustos.

4º cd y el directorio, cd C:\Users\Rebu\source\repos\HellorWorldMPI\x64\Debug

5º hacemos uso de la herramienta mpiexec que instalamos anteriormente

6º mpiexec -n 4 HellorWorldMPI , donde 4 es el numero de procesos y HellorWorldMPI el nombre de nuestro programa compilado.

7º lo ejecutamos y nos aparecerá en la terminal algo como lo siguiente:

```
PS C:\Users\Rebu\source\repos\HellorWorldMPI\x64\Debug> mpiexec -n 4 HellorWorldMPI
Tiempo total: 0.070102 segundos
```

Una vez ejecutado este nos devolver un tiempo total, correspondiente a lo que le ha llevado hacer la ejecución de todo nuestro programa.

Con ese dato y algunas pruebas podremos resolver algunas series de preguntas para entender mejor el funcionamiento de esta práctica con mpi.

## Experimentos con el Codigo Hello World

Realizaremos 10 ejecuciones del código para resolver una serie de cuestiones, que nos propone nuestro profesor. Y así intentar entender mejor el objetivo de la practica y la forma de trabajar, de las cpu multinúcleo.

1 Hola Mundo desde el Proceso 3 de 4 totales. Hola Mundo desde el Proceso 1 de 4 totales Hola Mundo desde el Proceso 2 de 4 totales Hola Mundo desde el Proceso 0 de 4 totales

2 Hola Mundo desde el Proceso 3 de 4 totales Hola Mundo desde el Proceso 1 de 4 totales Hola Mundo desde el Proceso 2 de 4 totales Hola Mundo desde el Proceso 0 de 4 totales

3 Hola Mundo desde el Proceso 3 de 4 totales Hola Mundo desde el Proceso 2 de 4 totales Hola Mundo desde el Proceso 0 de 4 totales Hola Mundo desde el Proceso 1 de 4 totales

4 Hola Mundo desde el Proceso 1 de 4 totales Hola Mundo desde el Proceso 0 de 4 totales Hola Mundo desde el Proceso 2 de 4 totales Hola Mundo desde el Proceso 3 de 4 totales

5 Hola Mundo desde el Proceso 2 de 4 totales Hola Mundo desde el Proceso 3 de 4 totales Hola Mundo desde el Proceso 1 de 4 totales Hola Mundo desde el Proceso 0 de 4 totales

6 Hola Mundo desde el Proceso 0 de 4 totales Hola Mundo desde el Proceso 3 de 4 totales Hola Mundo desde el Proceso 2 de 4 totales Hola Mundo desde el Proceso 1 de 4 totales

7 Hola Mundo desde el Proceso 1 de 4 totales Hola Mundo desde el Proceso 3 de 4 totales Hola Mundo desde el Proceso 2 de 4 totales Hola Mundo desde el Proceso 0 de 4 totales

8 Hola Mundo desde el Proceso 1 de 4 totales Hola Mundo desde el Proceso 3 de 4 totales Hola Mundo desde el Proceso 0 de 4 totales Hola Mundo desde el Proceso 2 de 4 totales

9 Hola Mundo desde el Proceso 2 de 4 totales Hola Mundo desde el Proceso 3 de 4 totales Hola Mundo desde el Proceso 1 de 4 totales Hola Mundo desde el Proceso 0 de 4 totales

10 Hola Mundo desde el Proceso 1 de 4 totales Hola Mundo desde el Proceso 3 de 4 totales Hola Mundo desde el Proceso 2 de 4 totales Hola Mundo desde el Proceso 0 de 4 totales

Con los resultados obtenidos, responderemos las siguientes preguntas:

**¿Se produce siempre el mismo resultado?**

No. Aunque cada proceso siempre imprime su mensaje correctamente, el en que aparecen los mensajes cambia en cada ejecución. Esto es debido a que en un entorno paralelo como MPI, la ejecución de los procesos es concurrente. Esto significa que la planificación de los procesos y la salida pueden variar debido a factores como la carga del sistema o la latencia de comunicación entre procesos.

**¿Por qué se ejecuta el mensaje completo que escribe cada proceso, y estos no se entremezclan?**

MPI gestiona la salida de cada proceso de manera que los mensajes no se entremezclan. Aunque el orden de los mensajes puede variar, por lo dicho anteriormente, MPI y el SO manejan el buffering de la salida estándar de manera que cada línea se completa antes de pasar al siguiente proceso.

**¿Está garantizado que cada proceso se ejecute en un núcleo de procesamiento distinto?**

No está garantizado que se ejecute en núcleos distintos, debido a que la asignación de procesos a núcleos es controlada por el sistema operativo, y no necesariamente cada proceso MPI se ejecuta en un núcle distinto.

MPI no tiene control directo sobre esto, aunque nuestro SO intentara balancear la carga.

**¿La cantidad de memoria que ocuparán los procesos, será proporcional al número de procesos que se lancen?**

En nuestro caso, cada proceso es independiente y ejecuta una copia del mismo código, por lo que la memoria total utilizada será aproximadamente proporcional al número de procesos. Sin embargo, en programas MPI más complejos, la memoria usada puede depender de diferentes factores.

## Implementación de un benchmark sintético usando MSMPI

Llegamos a la ultima parte de la práctica, y en esta parte vamos a utilizar el código de la practica 2 para convertilo en un código adaptado a MSMPI.

Codigo adaptado:

```
#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>


#define ITERATIONS 100


int main(int argc, char* argv[]) {
  MPI_Init(&argc, &argv);


  int rank, size;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);


  if (argc < 2) {
    if (rank == 0) {
      fprintf(stderr, "Uso: %s NUMOPERTOTAL\n", argv[0]);
    }
    MPI_Finalize();
    return -1;
  }


  int num_oper_total = atoi(argv[1]);
  int trozo = num_oper_total / size;
```

```
    float a = 1.0, b = 2.0, c = 3.0;
    double start, end;

    // Sincronizar procesos y medir tiempo
    MPI_Barrier(MPI_COMM_WORLD);
    start = MPI_Wtime();

    for (int j = 0; j < ITERATIONS; j++) {
        for (int k = 0; k < trozo; k++) {
            a = b * c;
        }
    }

    // Sincronizar procesos y medir tiempo
    MPI_Barrier(MPI_COMM_WORLD);
    end = MPI_Wtime();

    if (rank == 0) {
        printf("Tiempo total: %f segundos\n", end - start);
    }

    MPI_Finalize();
    return 0;
}
```

Después de convertir nuestro código, deberemos hacer un análisis de los componentes que utiliza nuestro ordenador. Para así posteriormente hacer una estimación teórica de la ganancia en velocidad que esperaremos de nuestro sistema bajo las demandas siguientes.

Cpu:

Ryzen 7 a 3.2Ghz, 8 nucleos y 16 hilos

Cache: L1 512Kb, L2 4Mb, L3 16Mb

Ram:

16Gb, 2 modulos a 3200Mhz

Ahora aprovechando las características que hemos sacado de nuestro pc, y uan prueba que vamos a hacer a continuación. Vamos a calcula la posible capacidad de ganancia en velocidad esperada en nuestro sistema.

```
PS C:\Users\Rebu\source\repos\HellorWorldMPI\x64\Debug> mpiexec -n 4 HellorWorldMPI 1000000
Tiempo total: 0.019042 segundos
```

Esta es la prueba que hemos realizado. Siendo 4 procesos del programa anterior mencionado en MPI y 1000000 repeticiones.

Para realizar el cálculo de la capacidad de aumento de velocidad de procesamiento, utilizaremos el concepto de Speedup en la computación paralela.

El Speedup es la relación entre el tiempo de ejecución de la mejor implementación secuencial y el tiempo de ejecución de la implementación paralela utilizando “n” nucleos.

Formula de Speedup:

$S(n)=Ts/Tp(n)$

$Tp(4)=0.019042$ segundos con 4 procesadores.

Asumimos $Tp(4)=0.019042$ es el tiempo de ejecución con 1 procesador para el mismo número de operaciones, pero no tenemos Ts. Vamos a calcularlo estimando Ts con la ley de Amdahl para n=4.

Ley de Amdahl:

La ley de Amdahl se utiliza para encontrar el Speedup máximo teórico, considerando que una fracción del programa (f) es paralelizable y el resto (1−f) es secuencial.

S(n)=1/(1-f)+f/n

Para simplificar, si asumimos que el programa es completamente paralelizable (f=1), la fórmula se reduce a:

S(n)=n

Estimacion de Ts:

Tp=0.019042 segundos

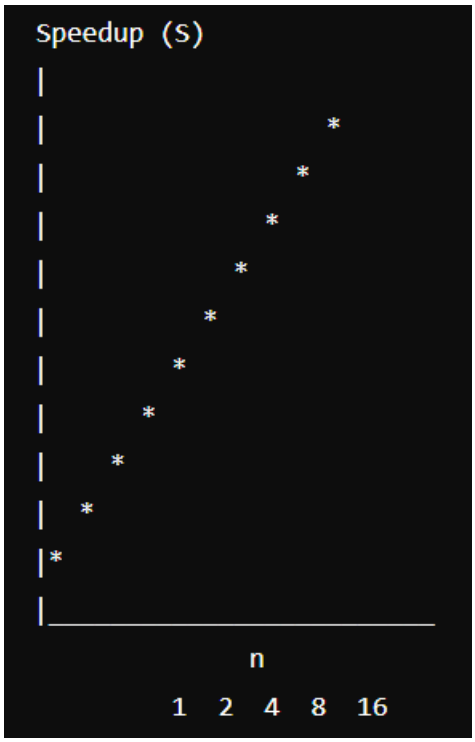
Si el Speedup S(4) es 4:

S(4)=T8/Tp(4)

T8= 0.076168 segundos

Tabla de Speedup:

| Número de Procesadores (n) | Speedup (S) | Tiempo de Ejecución (Tp) |
|---|---|---|
| 1 | 1 | 0.076168 |
| 2 | 2 | 0.038084 |
| 4 | 4 | 0.019042 |
| 8 | 8 | 0.009521 |
| 16 | 16 | 0.004760 |



La gráfica y los cálculos muestran que, **teóricamente**, el Speedup es lineal respecto al número de procesadores en una situación ideal donde el programa es completamente paralelizable y el overhead de la paralelización es despreciable. En la práctica, el Speedup puede ser menor debido a factores como la comunicación entre procesadores y el overhead de gestión de procesos.

Por último vamos a realizar diferentes pruebas con nuestro benchmark, incrementando en 1 el número de procesos. Iniciando en 1 y acabando en 16. Haremos esto para ver si de verdad se cumple nuestra teoría del speedup y que factores pueden estar afectando a nuestro sistema.

| Número de Procesos (n) | Tiempo de Ejecución (segundos) | Speedup |
|---|---|---|
| 1 | 0.076647 | 1.00 |
| 2 | 0.039574 | 1.94 |
| 3 | 0.025980 | 2.95 |
| 4 | 0.027886 | 2.75 |
| 5 | 0.015556 | 4.92 |
| 6 | 0.015550 | 4.92 |
| 7 | 0.021303 | 3.60 |
| 8 | 0.016092 | 4.76 |
| 9 | 0.014384 | 5.33 |
| 10 | 0.015394 | 4.97 |
| 11 | 0.014551 | 5.27 |
| 12 | 0.013428 | 5.71 |
| 13 | 0.012921 | 5.94 |
| 14 | 0.013102 | 5.85 |
| 15 | 0.014458 | 5.29 |
| 16 | 0.016136 | 4.75 |

Como podemos apreciar en la tabla cuando empezamos a realizar pruebas con 1 solo proceso y vamos incrementando el numero de procesos, el speedup se incrementa en gran medida en la mayoría de los casos. Pero cuando llegamos a cierto punto este comienza a disminuir. Este punto podría considerarse como el punto de saturación, donde agregar mas procesos no nos proporciona una mejora significativa en la velocidad de ejecución.

Basándonos en los datos obtenidos, podríamos decir que a partir de 9 o 10 procesos, el aumento de el speedup ya no es tan significativo. Llegando a disminuir en algunos de los casos.

Hay que tener en cuenta, que estas pruebas están realizadas en un entorno muy diferente, a lo descrito como entorno ideal. Y este entorno ideal es el que planteábamos en la cuestión anterior. Es por esto que las pruebas reales quedan tan distantes de la estimación que habíamos calculado.

En estas pruebas pueden verse afectadas, por procesos en segundo plano ejecutados por nuestro sistema operativo e incluso en primer plano. Ya que no tenemos el sistema preparado solo para este tipo de prueba y cerrado todo lo que no concierne a esto.

Entonces mi teoría es que lo más probable que al llegar a la capacidad máxima de procesos de la cpu ya estuviéramos topándonos con procesos que estuvieran en uso y estos limitaban la velocidad de trabajo de la maquina.