

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

ESTRUCTURA DE DATOS Y ALGORITMOS II

---

**PROYECTO 3.**  
**PROGRAMACIÓN PARALELA**

---

*ALUMNOS:*

MURRIETA VILLEGAS ALFONSO  
REZA CHAVARRÍA SERGIO GABRIEL  
VALDESPINO MENDIETA JOAQUÍN

*PROFESOR:*

EDGAR TISTA GARCÍA

CIUDAD UNIVERSITARIA, CD. MX., 21 DE NOVIEMBRE 2018

# Índice

<b>1. OBJETIVOS DEL PROYECTO</b>	<b>3</b>
<b>2. INTRODUCCIÓN</b>	<b>3</b>
<b>3. ANTECEDENTES</b>	<b>3</b>
<b>4. DESCRIPCIÓN DEL ALGORITMO SELECCIONADO</b>	<b>5</b>
4.1. INSERTIONSORT . . . . .	5
<b>5. PARALELIZACIÓN DEL ALGORITMO</b>	<b>5</b>
5.1. TIPO DE PARALELIZACIÓN . . . . .	5
5.2. MÉTRICAS DE DESEMPEÑO . . . . .	5
5.3. FORMAS DE COMUNICACIÓN . . . . .	6
<b>6. IMPLEMENTACIÓN Y RESULTADOS DEL ALGORITMO</b>	<b>6</b>
6.1. PARTE 1 . . . . .	6
6.1.1. ANÁLISIS . . . . .	7
6.2. PARTE 2 . . . . .	8
<b>7. CONCLUSIONES</b>	<b>9</b>
<b>8. REFERENCIAS</b>	<b>9</b>

## 1. OBJETIVOS DEL PROYECTO

Que el alumno ponga en práctica los conceptos de la programación paralela a través de la implementación de un algoritmo paralelo y una réplica grupal de dicha implementación; así mismo, que el alumno ponga en práctica sus habilidades de expresión oral ante el grupo

## 2. INTRODUCCIÓN

La programación paralela se define como un nuevo paradigma de programación. Es aquel formado por varios procesos que se ejecutan en varios procesadores conectados entre sí mediante una red de comunicación de manera simultánea para la resolución de problemas, los cuales requieren un tiempo bastante elevado para su finalización.

La programación paralela maneja procesos o métodos en ejecución simultáneos y este puede describir el comportamiento de un objeto. Existen ventajas como desventajas para el uso de este tipo de paradigma.

Como ventajas de uso son:

- Resolución de problemas con un solo procesador
- Disminuye los tiempos de procesos al ejecutar varios procesos a la vez
- Puede convertir o disminuir problemas de una complejidad mayor
- Permite la ejecución de código de manera más rápida
- Obtienen los resultados en menor tiempo que un programa secuencial

Aunque pueda ser la mejor opción realizar un programa en paralelo existen bastantes problemas para su ejecución.

- Consumo de energía en la ejecución
- Encontrar una división de tareas equilibrada a la hora de escribir el programa
- Falta de sincronización y comunicación entre procesos
- Dependencia de procesos por los resultados de otros trabajos
- Los mismos recursos (Hardware y Software) pueden también ocasionar problemas para realizar los procesos.

Para realizar la programación paralela se utiliza el recurso de los hilos. Los hilos son secuencias de tareas encadenadas que pueden ser ejecutadas por un sistema operativo. Los hilos son independientes, estos llevan información de estado y pueden interactuar solo a través de mecanismos de comunicación dados por el sistema.

## 3. ANTECEDENTES

A continuación, se presentan algunos conceptos elementales para la mejor comprensión de este proyecto:

TIPOS DE PROCESO:

• Disjuntos: Programa que realiza sus procesos en manera que diferentes bloques de un programa se realicen. En este caso los múltiples procesos no tiene ningún tipo de comunicación entre sí.

• Cooperativos: Este tipo de proceso comparte recursos de las tareas para la realización de una tarea, para esto los procesos necesitan estar sincronizados y también comunicarse entre sí.

PROGRAMACIÓN CONCURRENTES, PARALELA Y DISTRIBUIDA

Para clasificar a los programas paralelos por medio de hilos o hebras se tienen 3 clasificaciones

- Programación concurrente: Un programa de este tipo consiste en que varios procesos se ejecutan por medio de un solo procesador, realizando el proceso de paralelismo temporal. El paralelismo temporal consiste en que el procesador realiza pausas entre los procesos para simular una sincronización.
- Programación paralela: Programa que utiliza varios procesadores para realizar las tareas de manera simultánea. Estos necesitan comunicarse entre sí, para esto se conectan entre sí por medio de una red local.
- Programación distribuida: Este tipo de programa sigue el mismo concepto del paralelo, pero la diferencia consiste en que la red de comunicación es de manera global, o sea los procesadores se pueden encontrar en diferentes partes del mundo.

## MEMORIA

Para el uso de datos en un programa paralelo se debe de cuidar la entrada y salida de datos durante la ejecución de diferentes procesos. Ya que debido al mal manejo del acceso de memoria puede provocar fallos o errores en los procesos del programa.

Para el uso de la memoria se puede utilizar la memoria compartida, en esta forma de uso, los procesadores pueden entrar o acceder a cualquier localidad de memoria. Para el acceso, la red es controlada por el hardware. Para esto se utilizan varios métodos de control de acceso algunos de ellos son los semáforos, regiones críticas y monitores

Otra forma de utilizar la memoria es como si se trabaja de manera distribuida. De esta forma cada procesador utiliza su propia memoria y puede acceder a la memoria de otros procesadores. Los procesadores deben de reconocer el acceso a sus propias localidades de memoria y de los otros procesadores. También se utilizan formas de control como los canales o llamadas de procesos remotos.

## GRANULARIDAD

La granularidad se refiere a la cantidad de trabajo que cada procesador hace de manera independiente. Existen 2 granularidades. La granularidad gruesa consiste cuando se dividen las partes lógicas de los procesos secuenciales con poca sincronización o comunicación entre los procesos. Y la granularidad fina los procesadores ejecutan pocas instrucciones y aumenta la cantidad entre procesadores.

Para un programa paralelo se necesita un punto medio de granularidad. Si es gruesa, existe poca comunicación entre los procesos, esto puede ocasionar que el mismo proceso se tienda a ser un programa de tipo secuencial. Y si es fina la cantidad de tiempo entre la comunicación de los procesos puede aumentar el tiempo de ejecución.

## MÉTRICAS DE DESEMPEÑO

Debido a que un programa secuencial puede llegar a tener su forma paralela. Con este cambio puede llegarse a un proceso más rápido se da el concepto de métricas que determinan como se utiliza el tiempo de ejecución de un problema. La métrica de tiempo de proceso y comunicación: Esta métrica muestra que el tiempo puede subdividirse entre el tiempo de proceso y el tiempo de comunicación.

Speedup: Referido a la relación del tiempo de ejecución de un programa ejecutándose en un solo procesador sobre el tiempo de ejecución del mismo programa con  $n$  procesadores. Solo considera aspecto temporal, o sea no cuenta con otros aspectos como la topología establecida, carga o granularidad.

Eficiencia: Es asociada a la idea de que el Speed up realizado por  $n$  procesadores por la cantidad de procesadores Fracción Serial: Esta métrica trata de tomar diferentes factores respecto al tiempo.

## CLASIFICACIÓN DE PARALELISMO

**Paralelismo algorítmico:** Basado en las tareas independientes que ejecutan secciones del código o algoritmo. Esta clasificación se modela por el flujo de datos ya que el proceso se conduce a partir de la disponibilidad de datos. **Paralelismo geométrico:** Basado en la forma de paralelizar tareas independientes. Para la paralelización consiste en dividir de manera simétrica los problemas con respecto a una distribución uniforme con respecto a una región de datos para su manejo.

**Farm:** Clasificación formada por tareas independientes la cual procesa información de cualquier orden. Se utiliza cuando una solución puede dividirse en subtareas independientes. Con la independencia de los datos se requiere un coordinador para encargarse de la distribución del trabajo. Niveles de paralelismo

**Instrucción:** Los procesos se pueden ejecutar en diferentes procesadores. Las tareas dependientes se dividen en dependencia de flujo, el cual un trabajo necesita de datos de otro proceso para continuar, de la anti dependencia, en la cual cuando un proceso utiliza un registro o dato que se obtiene de otro proceso, y de salida, la cual si 2 procesos necesitan de un mismo registro para su almacenamiento en la memoria.

**Datos:** Distribución de datos de manera equitativa entre los procesadores utilizados en el problema. Se realizan las operaciones con los elementos dados.

**Ciclos:** En el tal caso de usar un ciclo y los datos no tienen dependencia entre si por las iteraciones entonces las operaciones se pueden hacer de manera arbitraria y de manera paralela hecha por los procesadores.

**Funcional:** Paralelismo que tiene varias instrucciones y que esos conjuntos se puedan paralelizar. Este implica una calendarización para resolver posibles dependencias entre las instrucciones.

## 4. DESCRIPCIÓN DEL ALGORITMO SELECCIONADO

### 4.1. INSERTIONSORT

El algoritmo de estudio es Insertion Sort. Insertion sort es un método de ordenamiento de datos. Este método considera un elemento en cada caso, insertándolo en el lugar apropiado entre aquellos que ya se encuentran ordenados. Al algoritmo requiere en la comparación de datos y el reacomodo de estos al insertar nuevos datos en la posición correspondiente. Requiere operaciones para ordenar una lista de  $n$  elementos

$$(O(n^2))$$

Por esto se decidió elegir el algoritmo para poder analizar y probar si puede disminuir el tiempo de ejecución.

## 5. PARALELIZACIÓN DEL ALGORITMO

### 5.1. TIPO DE PARALELIZACIÓN

Para el código paralelo de Insertion Sort se tomó en cuenta que se necesita tener la información del arreglo en la mayor parte del proceso para realizar los cambios. Los hilos tienen que cambiar o modificar el mismo arreglo a ordenar. Debido a que comparten recursos se clasifica como un programa concurrente.

### 5.2. MÉTRICAS DE DESEMPEÑO

Tiempos de procesamiento y comunicación:

Se da uso de la comunicación para el ordenamiento del arreglo, esto es para acelerar el proceso. Dependiendo de los hilos utilizados en el código de Insertion Sort. Dependerá el tiempo de comunicación.

**Speed Up:**

La relación del tiempo con 10000 datos del programa fue:

$$\frac{0,1280001}{0,1140001} = 1,122806$$

El cálculo se realizó con respecto a la prueba media de procesos, el cual fue 6.

**Eficiencia:**

La eficiencia contará como parametro la relación de Speed up sacada anterior mente entre los procesos realizados:

$$E(n) = \frac{S(n)}{n} = \frac{1,12280691}{6} = 0,1871344$$

**Fracción Serial:**

$$F = \frac{\frac{1}{1,122806} - \frac{1}{6}}{1 - \frac{1}{6}} = 0,86875096$$

### 5.3. FORMAS DE COMUNICACIÓN

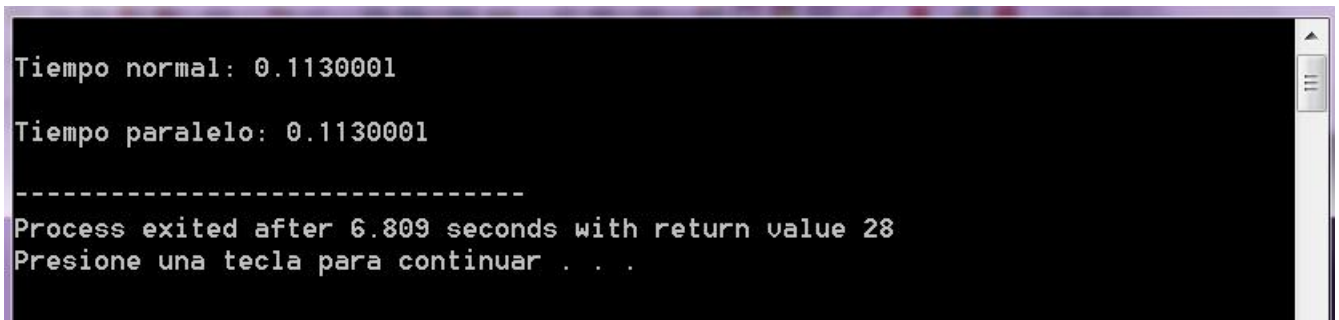
La comunicación en el código se realiza cuando se deben de compartir datos para el acomodo de los datos por cada hilo, esto se hace con la cláusula shared para saber que datos deben de compartir los hilos. También existen restricciones para las variables a usar con la cláusula private. No se modificarán esos valores, se crearán copias de los datos para su uso temporal.

## 6. IMPLEMENTACIÓN Y RESULTADOS DEL ALGORITMO

Con lo visto anteriormente se realizaron las pruebas. En el código anexado se utilizaron 2 arreglos de números enteros idénticos, estos arreglos tienen la información de números aleatorios del 1 al 1000. Ambos arreglos tienen tamaño fijo de 10000. Durante las diferentes pruebas se cambiaron la cantidad de hilos que trabajaron en el bloque de código paralelo.

A continuación, se muestran las salidas de los programas con sus correspondientes capturas de pantalla:

### 6.1. PARTE 1



```

Tiempo normal: 0.1130001
Tiempo paralelo: 0.1130001
-----
Process exited after 6.809 seconds with return value 28
Presione una tecla para continuar . . .

```

Figura 1: Ordenamiento por Insertion Sortc con 2 hilos a un arreglo de 10000 datos enteros.

```
Tiempo normal: 0.1340001
Tiempo paralelo: 0.1150001
-----
Process exited after 7.222 seconds with return value 28
Presione una tecla para continuar . . .
```

Figura 2: Ordenamiento por Insertion Sortc con 4 hilos a un arreglo de 10000 datos enteros.

```
E:\EDA 2\Monsalvo_Reza_G5_P1213_V1\Ejercicios\insertion1.exe
Tiempo normal: 0.1150001
Tiempo paralelo: 0.1140001
-----
Process exited after 6.929 seconds with return value 28
Presione una tecla para continuar . . .
```

Figura 3: Ordenamiento por Insertion Sort con 6 hilos a un arreglo de 10000 datos enteros.

```
E:\EDA 2\Monsalvo_Reza_G5_P1213_V1\Ejercicios\insertion1.exe
Tiempo normal: 0.1120001
Tiempo paralelo: 0.1200001
-----
Process exited after 6.225 seconds with return value 28
Presione una tecla para continuar . . .
```

Figura 4: Ordenamiento por Insertion Sort con 8 hilos a un arreglo de 10000 datos enteros.

```
Tiempo normal: 11.2590001
Tiempo paralelo: 11.2420001
-----
Process exited after 45.95 seconds with return value 29
Presione una tecla para continuar . . .
```

Figura 5: Ordenamiento por Insertion Sortc on 10 hilos a un arreglo de 10000 datos enteros.

### 6.1.1. ANÁLISIS

Al declarar una pequeña cantidad de hilos a usar en el programa se vio que se da un procedimiento de igual tiempo con respecto al código normal.

Al utilizar más hilos se da una mayor comunicación entre los mismos, por lo que da por consecuencia el aumento de tiempo con respecto al proceso normal.

Para tener una mejora en el tiempo se debe de tomar en cuenta la cantidad de hilos para el ordenamiento.

## 6.2. PARTE 2

```
Tiempo normal: 11.6050001
Tiempo paralelo: 11.1600001
-----
Process exited after 29.5 seconds with return value 29
Presione una tecla para continuar . . .
```

Figura 6: Ordenamiento por Insertion Sortc con 2 hilos a un arreglo de 100000 datos enteros.

```
Tiempo normal: 0.1280001
Tiempo paralelo: 0.1140001
-----
Process exited after 6.345 seconds with return value 28
Presione una tecla para continuar . . .
```

Figura 7: Ordenamiento por Insertion Sortc con 4 hilos a un arreglo de 100000 datos enteros.

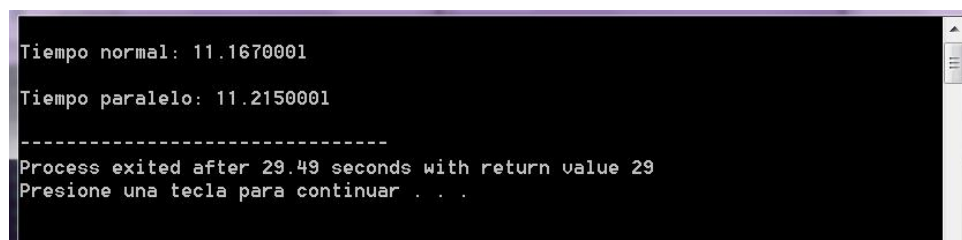
```
Tiempo normal: 11.2030001
Tiempo paralelo: 11.1510001
-----
Process exited after 43.41 seconds with return value 29
Presione una tecla para continuar . . .
```

Figura 8: Ordenamiento por Insertion Sortc con 6 hilos a un arreglo de 100000 datos enteros.

```
Tiempo normal: 11.1500001
Tiempo paralelo: 11.1210001
-----
Process exited after 45.66 seconds with return value 29
Presione una tecla para continuar . . .
```

Figura 9: Ordenamiento por Insertion Sortc con 8 hilos a un arreglo de 100000 datos enteros.





```
Tiempo normal: 11.1670001
Tiempo paralelo: 11.2150001
-----
Process exited after 29.49 seconds with return value 29
Presione una tecla para continuar . . .
```

Figura 10: Ordenamiento por Insertion Sortc con 10 hilos a un arreglo de 100000 datos enteros.

## 7. CONCLUSIONES

A lo largo de este proyecto se aplicaron las distintas directivas y constructores de OpenMp que previamente vimos en la práctica 12 y 13 del laboratorio, además de que aplicamos el paradigma de programación en paralelo donde se pudo demostrar que los tiempos de ejecución realmente eran distintos a los de la versión serial del algoritmo de insertion sort.

Este proyecto tuvo como propósito demostrar que la versión paralela de los algoritmos puede ser totalmente buena en tiempo de ejecución debido a la forma en que se distribuye la carga de tareas en los procesadores.

Sin embargo, cabe destacar que pese los resultado como fueron vistos previamente fueron mejores a los de la versión serial, realmente tampoco aportaron muchísima ventaja respecto a tiempos y esto se debe en gran parte a como fue paralelizado.

Sin duda alguna, se pudieron aplicar los conceptos vistos en clase con respecto a la programación paralela y como aplicarlos a un algoritmo visto en el curso.

La programación paralela necesita de varios factores como el tipo de datos a manejar y sobre todo un buen manejo de los recursos para no ocasionar defectos en el procedimiento como en el tiempo de ejecución.

## 8. REFERENCIAS

- (2007). Introduction to parallel programming, Intel Software collage.
- Galvin Baer Peter, Gagne Greg . Fundamentos de Sistemas operativos. MacGraw Hill.
- Elba Karen Saenz García. (2017). Manual de Prácticas del laboratorio de Estructuras de Datos y algoritmos 1I. UNAM, Facultad de Ingeniería.
- Recuperado el 18 de noviembre de 2018, de <https://www.experts-exchange.com/questions/23904907/Parallel-Insertion-Sort.html>