



**BUAP**

**BENEMÉRITA UNIVERSIDAD  
AUTÓNOMA DE PUEBLA**

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

**MANUAL DE  
FUNCIONAMIENTO**

EQUIPO:

MARTHA ELENA CARRERA SANCHEZ

EDUARDO LÓPEZ AGUILAR

ERICK LIEVANA POY

JORGE BLANCARTE LÓPEZ

JOSE ANTONIO BARRERA OROPEZA

Abril 2021

## INTRODUCCION

En este manual podrá encontrar cual es el funcionamiento y la organización de los programas que se desarrollaron para hacer el análisis de los tiempos y memorias con los diferentes tipos y plataformas para la multiplicación de matrices.

LA organización de nuestros programas fue de la siguiente manera:

- \* Multiplicación de matrices secuencial
- \* Multiplicación de matrices implementando algoritmo fox en MPI
- \* Multiplicación de matrices implementando algoritmo fox en Open MP

Para el código se utilizaron diferentes tipos de dato, los cuales se dividen en:

- \* Enteros cortos (short int)
- \* Enteros largos (long int)
- \* Reales (float)
- \* Doubles

También se consideraron ejecutar estos programas con diferentes número de procesos y núcleos. Principalmente se ejecutan con: 1 , 4 procesos y 1 y 4 núcleos.

## OpenMP

Para la programación multiproceso de memoria compartida en múltiples plataformas. Permite añadir concurrencia a los programas escritos en C, C++ y Fortran sobre la base del modelo de ejecución fork\_join. Está disponible en muchas arquitecturas, incluidas las plataformas de Unix y de Microsoft Windows. Se compone de un conjunto de directivas de compilador, rutinas de biblioteca, y variables de entorno que influyen el comportamiento en tiempo de ejecución.

Definido conjuntamente por proveedores de hardware y de software, OpenMP es un modelo de programación portable y escalable que proporciona a los programadores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas, para plataformas que van desde las computadoras de escritorio hasta supercomputadoras. Una aplicación construida con un modelo de programación paralela híbrido se puede ejecutar en un cluster de computadoras utilizando OpenMP y MPI, o a través de las extensiones de OpenMP para los sistemas de memoria distribuida.

## PROCEDIMIENTO

Aplicación de Algoritmo Fox con MPI

**MPI\_Comm\_rank:** Determina el rango (identificador) del proceso que lo llama dentro del comunicador seleccionado

**MPI\_Comm\_size:** Determina el tamaño del comunicador seleccionado, es decir, el número de procesos que están actualmente asociados a este.

**MPI\_Send:** Realiza el envío de un mensaje de un proceso fuente a otro destino.

**MPI\_Recv:** Rutina de recibimiento de un mensaje desde un proceso.

**MPI\_COMM\_WORLD:** Identificador del comunicador al que pertenecen todos los procesos de una ejecución MPI.

Especificación de tipos de datos

MPI_INT	Tipo de dato int.
MPI_SHORT	Tipo de dato short
MPI_FLOAT	Tipo de dato float.
MPI_DOUBLE	Tipo de dato double.

**MPI\_Sendrecv :** Envía y recibe un mensaje en la misma operación.

Una vez que se tengan instalados todos los programas y requerimientos que se mencionan en Manual de instalación procederemos a abrir nuestra terminal en nuestro sistema operativo.

Accedemos a la ubicación de los programas que ejecutaremos. Probaremos de forma individual cada programa y posteriormente ejecutaremos los scripts para poder automatizar las ejecuciones y finalmente obtener los resultados.

Multiplicación de Matrices secuencial  
Compilamos nuestro programa con el comando:

```
mpicc enteroFox.c -o EnteroFox
```

Para ejecutar el programa se tendrá que escribir el siguiente comando:

```
mpirun -np 4 ./enterosFox 16
```

Donde el tercer parámetro indica el número de procesos que se ejecutará el programa y el quinto parámetro nos indica el tamaño de la matriz, cabe mencionar que este dato debe ser un entero y debe ser positivo.

## ALGORITMO FOX

El algoritmo implementado es FOX el cual está definido de la siguiente forma:  
La complejidad del Algoritmo está dada por  $N^3/P$

Está propuesto para un arreglo bidimensional de elementos de procesamiento, interconectados como una malla y además con los extremos de cada fila y columna interconectados entre sí, es decir formando la estructura denominada toro.

Una vez más, la distribución de los datos de las matrices A, B, C es similar a la definida antes, es decir que si enumeran los procesadores de acuerdo a su posición en el arreglo bidimensional, el procesador  $P_{ij}$  ( $0 \leq i, j \leq p-1$ ), contiene los elementos o bloques de la posición  $ij$  ( $0 \leq i, j \leq p-1$ ), de las matrices, A, B, y C, en este caso no se define ningún paso inicial de reubicación de los datos, sino que a partir de aquí se define iterativa mente el algoritmo. En la iteración o paso K

\*El bloque datos de la matriz A almacenado en el proceso de la posición  $i, i+k \bmod P$  (fila  $i$ , columna  $i+k \bmod P$ ) Se envía a todos los procesadores de la fila  $i$  (Su misma fila) en broadcast Por fila del arreglo bidimensional.

\*Se multiplican en cada procesador los datos de A recibidos con los datos de B almacenados localmente, obteniendo un resultado parcial de la matriz C.

\* Se rotan los datos de B hacia arriba

Una vez que se realicen las pruebas se pueden ejecutar los scripts para poder automatizar los procesos, en general el script realiza las siguientes operaciones:

```
#!/bin/bash
function mpi()
for i in 1..100
do
mpirun np 1 ./foxMPI $1;
done
mv datosTiempo.txt tiempoMPI$1.txt
mv datosMemoria.txt memoriaMPI$1.txt
./lectura memoriaMPI1.txttiempoMPI1.txt promediosMPI
function sec()
for i in 1..100
do
mpirun ./secuencial $1;
done
mv datosTiempo.txt tiempoSec$1.txt
mv datosMemoria.txt memoriaSec$1.txt
./lectura memoriaSec1.txttiempoSec1.txt promediosSec
function omp()
```

```

for i in 1..100
do
mpirun -np 1 ./foxOMP $1 0;
done
mv datosTiempo.txt tiempoOMP$1.txt
mv datosMemoria.txt memoriaOMP$1.txt
./lectura memoriaOMP1.txttiempoOMP1.txt promediosOMP
FILE=datosTiempo.txt
if [[ -f "$FILE" ]]; then
rm datosTiempo.txt
fi

FILE=datosMemoria.txt
if [[ -f "$FILE" ]]; then
rm datosMemoria.txt
fi

#mpi "64"
#mpi "128"
#mpi "256"
#mpi "512"
#mpi "1024"
mpi "2048"
#mpi "4096"
#mpi "8192"

#sec "64"
#sec "128"
#sec "256"
#sec "512"
#sec "1024"
#sec "2048"
#sec "4096"
#sec "8192"

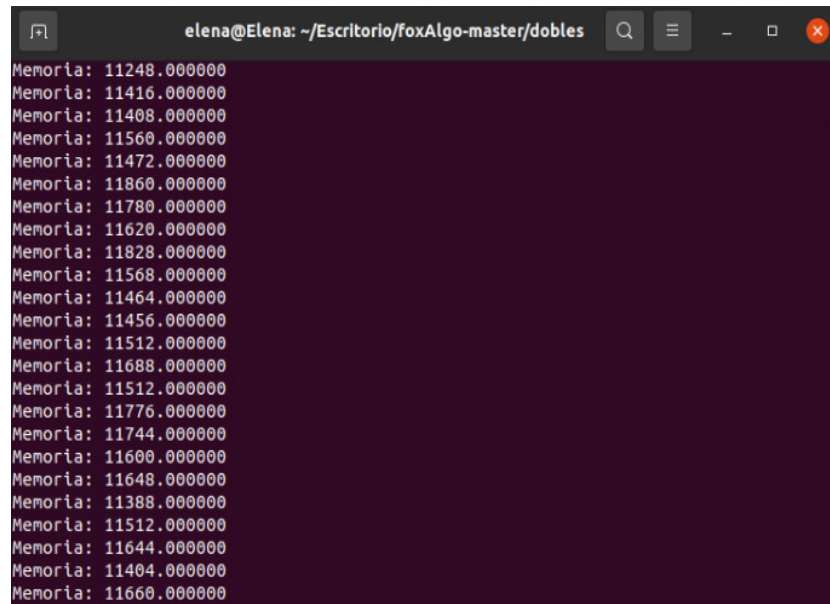
#omp "64"
#omp "128"
#omp "256"
#omp "512"
#omp "1024"
omp "2048"
#omp "4096"
#omp "8192"

```

```
#paste promediosMPITiempo.txt promediosOMPTiempo.txt  
promediosSecTiempo.txt ordenes.txt & promediosTiempo.txt  
#paste promediosMPIMemoria.txt promediosOMPMemoria.txt  
promediosSecMemoria.txt ordenes.txt & promediosMemoria.txt  
  
#gnuplot graficaTiempo.ptl  
#gnuplot graficaMemoria.ptl
```

Ejemplo:

Como se observa al usar el scrips nos genera los siguientes resultados de memoria en la terminal

A terminal window with a dark purple background. The title bar shows the user 'elena@Elena' and the directory '~/Escritorio/foxAlgo-master/dobles'. The terminal displays a list of 20 memory usage entries, each starting with 'Memoria:' followed by a numerical value. The values range from 11248.000000 to 11660.000000.

```
Memoria: 11248.000000  
Memoria: 11416.000000  
Memoria: 11408.000000  
Memoria: 11560.000000  
Memoria: 11472.000000  
Memoria: 11860.000000  
Memoria: 11780.000000  
Memoria: 11620.000000  
Memoria: 11828.000000  
Memoria: 11568.000000  
Memoria: 11464.000000  
Memoria: 11456.000000  
Memoria: 11512.000000  
Memoria: 11688.000000  
Memoria: 11512.000000  
Memoria: 11776.000000  
Memoria: 11744.000000  
Memoria: 11600.000000  
Memoria: 11648.000000  
Memoria: 11388.000000  
Memoria: 11512.000000  
Memoria: 11644.000000  
Memoria: 11404.000000  
Memoria: 11660.000000
```

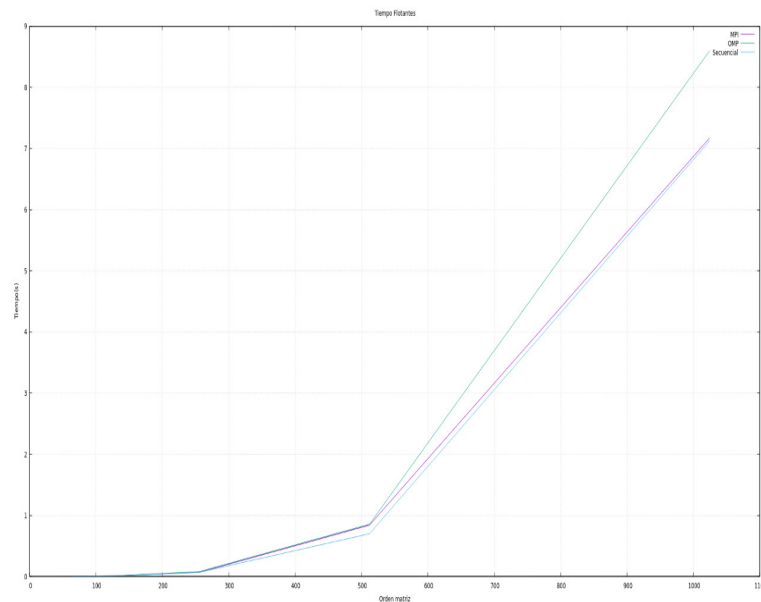
Con esto nos generará los archivos con los promedios de las ejecuciones de cada programa para que posteriormente, se puedan ejecutar en gnuplot y así poder determinar que programa y con qué tipo de dato es más eficiente en cuestión de tiempo y de memoria.

## GRAFICACIÓN DE LOS DATOS DE TIEMPO Y MEMORIA OBTENIDA

Para poder graficar en gnuplot los datos obtenidos, se realiza un script en cual une los puntos en una gráfica 2D que nos muestra el tamaño de la matriz y el tiempo que tarda según los datos obtenidos.

```
set title "Uso de memoria con 1 core(s) datos Cortos"
set title font "Helvetica,14"
set xlabel "Orden matriz"
set ylabel "Memoria(kb)"
set grid
plot "promediosMemoria.txt" using 4:1 with linespoints lw 3 title "MPI"
replot "promediosMemoria.txt" using 4:2 with linespoints lw 3 title "OMP"
replot "promediosMemoria.txt" using 4:3 with linespoints lw 3 title "Secuen-
cial"
replot "promediosMemoria.txt" using 4:1:(sprintf("(%d, %f)", $1,$2)) with
labels notitle
#replot "promediosMemoria.txt" using 4:2:(sprintf("(%d,%f)", $1, $3)) with
labels notitle
#replot promediosMemoria.txt using 4:3:(sprintf("(%d, %f)", $1, $4)) with
labels notitle
pause -1
```

Posteriormente al ejecutar el scripts nos mostrara el tamaño en la columna x muestra el tamaño de la matriz y en la columna y muestra la memoria utilizada en kb



Aqui se observa otra grafica con el scripts teniendo en cuenta el nombre del archivo, su tamaño de matriz, promedios de memoria y su regilla de datos.

