



## **MPI (Message Passing Interface) y openMP Filtros Activos**

Carlos Andrés Mata Calderón

Escuela de Ingeniería en Computadores

CE 1112 — Taller de Señales Mixtas, Especificación Taller 2

Profesor: Luis Alonso Barboza Artavia

II Semestre 2024

## *Indice*

<b>Preguntas.....</b>	<b>3</b>
1. ¿Qué es Message Passing Interface (MPI)?.....	3
2. ¿Qué es un rank en un proceso?.....	3
3. ¿Cómo se establece el código que ejecuta el nodo raíz y aquellos nodos que están conectados a éste?.....	3
4. ¿Qué es MPICH?.....	4
5. ¿Qué es OpenMP?.....	4
6. Respecto a OpenMP, ¿qué permiten las variables de entorno y funciones: OMP_NUM_THREADS y omp_get_max_threads()?.....	4
7. ¿Qué es un pragma? Específicamente en OpenMP, ¿qué hacen: omp parallel, omp single, omp for, reduction?.....	5
8. ¿Cómo se puede permitir la sincronización y medición de tiempo en OpenMP?.....	6
<b>Análisis.....</b>	<b>7</b>
MPI_Send.....	7
MPI_Recv.....	7
MPI_Finalize.....	8
Compilación y Ejecución en el Clúster.....	8
Master.....	9
Este resultado muestra que el programa distribuyó la suma de los primeros 1000 números entre los tres procesos. Cada proceso calculó una suma parcial y el maestro sumó todas las sumas parciales para obtener el total final.....	11
Esclavo 2 (slave2).....	11
<b>Ejercicios prácticos.....</b>	<b>13</b>
1. Multiplicación de dos matrices de 4x4.....	13
2. Filtro de imágenes Sobel.....	14
Sobel Serial.....	15
Sobel con OPENMP 1 Thread.....	16
OPEN MPI.....	17
Análisis de Filtro Sobel.....	18
<b>Conclusiones.....</b>	<b>19</b>

## Preguntas

### 1. ¿Qué es Message Passing Interface (MPI)?

MPI es un estándar para la programación de aplicaciones paralelas en sistemas distribuidos. Es una interfaz que define un conjunto de funciones y protocolos para la comunicación entre procesos que pueden estar ejecutándose en diferentes nodos de un clúster o supercomputadora. MPI permite que los procesos intercambien datos mediante el envío y recepción de mensajes, facilitando el desarrollo de aplicaciones paralelas y distribuidas de alto rendimiento.

### 2. ¿Qué es un rank en un proceso?

En MPI, un "rank" es un identificador único asignado a cada proceso dentro de un comunicador. El rank se utiliza para identificar y distinguir cada proceso en el conjunto de procesos que participan en una ejecución paralela. Por ejemplo, el proceso con rank 0 suele considerarse el proceso raíz o maestro, y los demás procesos tienen ranks asignados desde 1 hasta el número total de procesos menos uno.

### 3. ¿Cómo se establece el código que ejecuta el nodo raíz y aquellos nodos que están conectados a éste?

En un programa MPI, todos los procesos ejecutan el mismo código. Sin embargo, es posible diferenciar el comportamiento de cada proceso utilizando el rank. Utilizando la función **`MPI_Comm_rank`**, cada proceso obtiene su propio rank y, con estructuras condicionales (**`if`**), se puede establecer qué código ejecuta el proceso raíz (generalmente el de rank 0) y qué código ejecutan los demás procesos. De esta manera, el proceso raíz puede encargarse de tareas como la

lectura de datos o la coordinación, mientras que los procesos trabajadores realizan cálculos paralelos.

#### **4. ¿Qué es MPICH?**

MPICH es una implementación de código abierto del estándar MPI. Es una biblioteca que permite desarrollar y ejecutar aplicaciones paralelas utilizando MPI en diferentes plataformas y sistemas operativos. MPICH se enfoca en proporcionar un rendimiento alto y portabilidad, facilitando el desarrollo de aplicaciones paralelas escalables en clústeres y supercomputadoras.

#### **5. ¿Qué es OpenMP?**

OpenMP es una API (Interfaz de Programación de Aplicaciones) para la programación de aplicaciones paralelas en sistemas con memoria compartida. Proporciona un conjunto de directivas, funciones y variables de entorno que permiten al programador parallelizar bucles y secciones de código en lenguajes como C, C++ y Fortran. OpenMP simplifica la creación de programas paralelos al abstraer los detalles de gestión de hilos y sincronización.

#### **6. Respecto a OpenMP, ¿qué permiten las variables de entorno y funciones:**

##### **OMP\_NUM\_THREADS y omp\_get\_max\_threads()?**

**OMP\_NUM\_THREADS:** Es una variable de entorno que establece el número de hilos que se utilizarán en la ejecución de las regiones paralelas de un programa OpenMP. Al definir esta variable antes de ejecutar el programa, se puede controlar el grado de paralelismo sin modificar el código fuente.

**omp\_get\_max\_threads()**: Es una función de la biblioteca OpenMP que devuelve el número máximo de hilos disponibles para ejecutar regiones paralelas. Esto permite al programa conocer cuántos hilos puede utilizar y ajustar dinámicamente su comportamiento según el número de hilos disponibles.

## 7. ¿Qué es un pragma? Específicamente en OpenMP, ¿qué hacen: **omp parallel**, **omp single**, **omp for**, **reduction**?

Un pragma es una directiva para el compilador que proporciona instrucciones adicionales sobre cómo compilar el código. En OpenMP, los pragmas se utilizan para indicar al compilador cómo parallelizar ciertas secciones del código.

- **#pragma omp parallel**: Indica que el bloque de código siguiente debe ejecutarse en paralelo por un equipo de hilos. Cada hilo ejecuta una copia del bloque de código.
- **#pragma omp single**: Dentro de una región paralela, especifica que el bloque de código siguiente debe ser ejecutado por un solo hilo, generalmente el primero que llega a esa sección. Los demás hilos esperan a que el hilo termine esa sección, a menos que se use la cláusula **nowait**.
- **#pragma omp for**: Se utiliza para parallelizar bucles **for**. Divide las iteraciones del bucle entre los hilos disponibles para que cada hilo ejecute una parte del bucle.
- **reduction**: Es una cláusula que se utiliza junto con **omp parallel for** para especificar que una operación de reducción (como suma, multiplicación, mínimo, máximo) debe aplicarse a una variable compartida. Esto permite combinar resultados parciales de cada hilo en un resultado final de manera segura y eficiente.

## 8. ¿Cómo se puede permitir la sincronización y medición de tiempo en OpenMP?

Para la sincronización en OpenMP, se utilizan diversas directivas y funciones:

- **Barreras de sincronización:** `#pragma omp barrier` fuerza a que todos los hilos esperen en ese punto hasta que todos hayan alcanzado la barrera.
- **Secciones críticas:** `#pragma omp critical` define una sección del código que debe ser ejecutada por un solo hilo a la vez, evitando condiciones de carrera.
- **Bloques atómicos:** `#pragma omp atomic` asegura que una operación específica sobre una variable sea atómica, es decir, que se realice sin interrupción.

Para la medición de tiempo, OpenMP proporciona la función `omp_get_wtime()`, que devuelve el tiempo en segundos desde un punto arbitrario en el pasado (generalmente el inicio de la ejecución del programa). Al tomar marcas de tiempo antes y después de una sección de código, es posible calcular el tiempo transcurrido.

## Análisis

### **MPI\_Send**

La función **MPI\_Send** se utiliza para enviar datos de un proceso a otro en un entorno MPI. En este programa, el proceso maestro (root) utiliza **MPI\_Send** para:

- Enviar el número de filas a procesar: Le indica a cada proceso esclavo cuántos elementos del arreglo debe sumar.
- Enviar la porción del arreglo: Envía la sección específica del arreglo que cada esclavo procesará.

Por ejemplo:

```
// Envío del número de filas a cada esclavo
ierr = MPI_Send(&num_rows_to_send, 1, MPI_INT, an_id, send_data_tag,
MPI_COMM_WORLD);
// Envío de la porción del arreglo
ierr = MPI_Send(&array[start_row], num_rows_to_send, MPI_INT, an_id,
send_data_tag, MPI_COMM_WORLD);
```

Aquí, el maestro está comunicando a los esclavos la cantidad de datos que recibirán y luego les envía esos datos.

### **MPI\_Recv**

La función **MPI\_Recv** se utiliza para recibir datos enviados por otros procesos. En este programa:

- **Procesos esclavos:** Usan **MPI\_Recv** para recibir del maestro el número de filas a procesar y la porción del arreglo correspondiente.

```
// Recepción del número de filas a procesar
ierr = MPI_Recv(&num_rows_to_receive, 1, MPI_INT, root_process,
send_data_tag, MPI_COMM_WORLD, &status);
// Recepción de la porción del arreglo
ierr = MPI_Recv(&array2, num_rows_to_receive, MPI_INT, root_process,
send_data_tag, MPI_COMM_WORLD, &status);
```

- **Proceso maestro:** Usa **MPI\_Recv** para recibir las sumas parciales calculadas por los esclavos.

```
// Recepción de la suma parcial de un esclavo
ierr = MPI_Recv(&partial_sum, 1, MPI_LONG, MPI_ANY_SOURCE,
return_data_tag, MPI_COMM_WORLD, &status);
```

## **MPI\_Finalize**

La función **MPI\_Finalize** es esencial para terminar correctamente el programa MPI.

Se asegura de que todos los procesos hayan concluido y libera los recursos utilizados por MPI.

```
ierr = MPI_Finalize();
```

Debe ser llamada por todos los procesos al finalizar su ejecución.

## **Compilación y Ejecución en el Clúster**

Para probar el código, construí un clúster utilizando máquinas en laboratorios de computadores, donde tenemos 1 master y 2 esclavos. Aquí está lo que hice:

## Master

Se configuro el master de esta forma.

```

curso@soporte-OptiPlex-7050:~$ neofetch
curso@soporte-OptiPlex-7050:~$ neofetch
[...]
curso@soporte-OptiPlex-7050:~$ ifconfig
[...]
curso@soporte-OptiPlex-7050:~$ cat /etc/hosts
[...]

```



Note que el master es tiene una **master\_ip=172.18.170.56**, los esclavos son

**slave2\_ip=172.18.101.239** y **slave1\_ip=172.18.124.86**, que se guardan en el archivo

/etc/hosts/ para que estos se puedan conectar al master, observe que el host tiene la capacidad de 2 procesos y los slaves de 1 proceso.

Se utilizo el comando **mpicc** este comando se encarga de compilar el programa.

```
mpicc -o SUM_MPI sum_mpi.c
```

Se utilizo el comando **mpirun** este comando se encarga de ejecutar el programa y le decimos que queremos 3 nodos, osea el master y 2 esclavos. Y ejecutamos las sumas.

```
mpirun --hosts /etc/hosts/ -np 3 ./SUM_MPI
```

Donde el resultado fue:

```
mpirun --hosts /etc/hosts -np 3 ./SUM_MPI
please enter the number of numbers to sum: 1000
sum 168334 calculated by root process
Partial sum 166667 returned from process 1
Partial sum 166999 returned from process 2
The grand total is: 502000
```

Este resultado muestra que el programa distribuyó la suma de los primeros 1000 números entre los tres procesos. Cada proceso calculó una suma parcial y el maestro sumó todas las sumas parciales para obtener el total final.

### **Esclavo 2 (slave2)**

Note que el slave tiene la **172.18.101.239** y este en el archivo **/etc/hosts/** tiene el master que tenia la **172.18.170.56**

```

curso@soporte-OptiPlex-7050:~$ sudo mount 172.18.170.56:/home/
[sudo] password for curso:
curso@soporte-OptiPlex-7050:~$ ifconfig
en0: flags=4163<IP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 172.18.101.239 netmask 255.255.0.0 broadcast 172.18.0.255
                ether 6c:2b:59:de:1b:fc txqueuelen 1000 (Ethernet)
                  RX packets 193542 bytes 38138167 (38.1 MB)
                  RX errors 0 dropped 643 overruns 0 frame 0
                  TX packets 32832 bytes 7619671 (7.6 MB)
                  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
                  device interrupt 16 memory 0x92d00000-92d20000
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
            loop txqueuelen 1000 (Local Loopback)
              RX packets 5578 bytes 1636616 (1.6 MB)
              RX errors 0 dropped 0 overruns 0 frame 0
              TX packets 5578 bytes 1636616 (1.6 MB)
              TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
curso@soporte-OptiPlex-7050:~$ cat /etc/hosts
127.0.0.1      localhost
172.18.170.56  master
#127.0.1.1      soporte-OptiPlex-7050

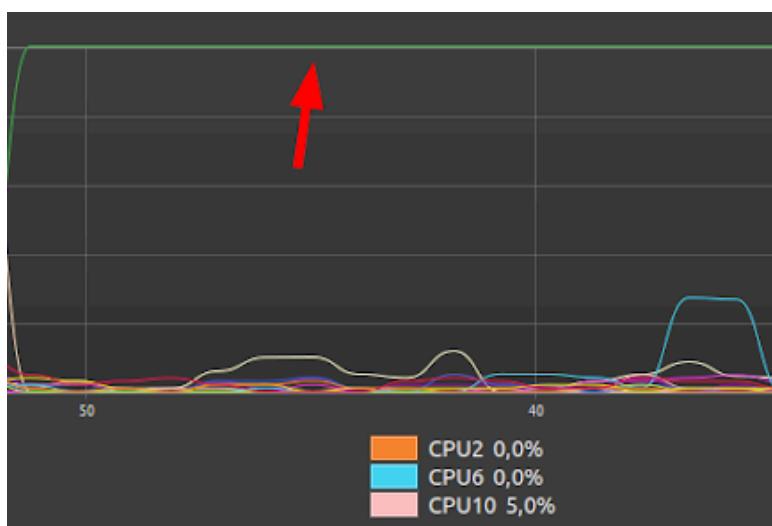
# The following lines are desirable for IPv6 capable hosts
#:1      ip6-localhost ip6-loopback
#fe00::0 ip6-localnet
#ff00::0 ip6-mcastprefix
#ff02::1 ip6-allnodes
#ff02::2 ip6-allrouters

```

Ahora bien note que esta computadora se esta corriendo un **SUM\_MPI** este fue el archivo que compilamos y ejecutamos desde el

PTD	USER	PR	NT	VTRT	RES	SHR	S	%CPUL	%MEM	TIME+	COMMAND
44079	curso	20	0	174592	9492	7236	R	99,7	0,1	23:27.48	SUM_MPI
1022	curso	20	0	401564	374048	100370	S	0,2	2,1	0:07.15	gnome-shell
1791	curso	20	0	589284	103076	60396	S	1,0	0,6	66:10.05	Xorg
1778	curso	P	-11	3247324	20996	16080	S	0,7	0,1	33:45.24	pulseaudio
14	root	20	0	0	0	0	I	0,3	0,0	0:49.42	rcu_sched
1893	curso	20	0	7384	4268	3808	S	0,3	0,0	0:00.22	dbus-daemon
2494	curso	20	0	4298524	568172	270996	S	0,3	3,5	11:04.15	firefox
39104	curso	20	0	1488948	84968	52520	S	0,3	0,5	0:06.49	evince
39481	curso	20	0	2450204	102144	88756	S	0,3	0,6	0:00.99	Isolated Web Co
1	root	20	0	168620	11972	8496	S	0,0	0,1	0:03.56	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.01	kthreadd
3	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_par_gp
5	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	slub_flushwq
6	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	netns
8	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	kworker/0:0-H-events_highpri
10	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	mm_percpu_wq
11	root	20	0	0	0	0	S	0,0	0,0	0:00.00	rcu_tasks_rude_
12	root	20	0	0	0	0	S	0,0	0,0	0:00.00	rcu_tasks_trace
13	root	20	0	0	0	0	S	0,0	0,0	0:00.97	ksoftirqd/0
15	root	rt	0	0	0	0	S	0,0	0,0	0:00.12	migration/0
16	root	-51	0	0	0	0	S	0,0	0,0	0:00.00	idle_inject/0
17	root	20	0	0	0	0	I	0,0	0,0	0:04.41	kworker/0:1-events
18	root	20	0	0	0	0	S	0,0	0,0	0:00.00	cpuhp/0
19	root	20	0	0	0	0	S	0,0	0,0	0:00.00	cpuhp/1
20	root	-51	0	0	0	0	S	0,0	0,0	0:00.00	idle_inject/1
21	root	rt	0	0	0	0	S	0,0	0,0	0:00.23	migration/1
22	root	20	0	0	0	0	S	0,0	0,0	0:00.30	kssoftirqd/1
24	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	kworker/1:0-H-events_highpri
25	root	20	0	0	0	0	S	0,0	0,0	0:00.00	cpuhp/2
26	root	-51	0	0	0	0	S	0,0	0,0	0:00.00	idle_inject/2
27	root	rt	0	0	0	0	S	0,0	0,0	0:00.21	migration/2

Aqui podemos el consumo de proceso cuando se ejecutó:



## Ejercicios prácticos

### 1. Multiplicación de dos matrices de 4x4

```
curso@soporte-OptiPlex-7050:~/Desktop/sharedfolder/Taller 2 MPI(1)/src$ mpirun -hostfile /etc/hosts -np 4 MATRICES_MULTIPLICACION
>>> Proceso [0] Reporte de Métricas <<
Memoria Máxima Usada: 11628 KB
Tiempo de Cálculo: 0.000000 segundos
Tiempo de Comunicación: 0.000002 segundos
Datos Envíados: 16 bytes
Datos Recibidos: 16 bytes
-----
===== Resultado de la Multiplicación de Matrices =====
 98 108 110 120
262 228 254 280
314 356 398 440
426 484 542 600

>>> Proceso [1] Reporte de Métricas <<
Memoria Máxima Usada: 11604 KB
Tiempo de Cálculo: 0.000000 segundos
Tiempo de Comunicación: 0.000001 segundos
Datos Envíados: 16 bytes
Datos Recibidos: 16 bytes
-----
>>> Proceso [2] Reporte de Métricas <<
Memoria Máxima Usada: 11544 KB
Tiempo de Cálculo: 0.000000 segundos
Tiempo de Comunicación: 0.000004 segundos
Datos Envíados: 16 bytes
Datos Recibidos: 16 bytes
-----
>>> Proceso [3] Reporte de Métricas <<
Memoria Máxima Usada: 11580 KB
Tiempo de Cálculo: 0.000001 segundos
Tiempo de Comunicación: 0.000016 segundos
Datos Envíados: 16 bytes
Datos Recibidos: 16 bytes
-----
```

nov 7 18:26

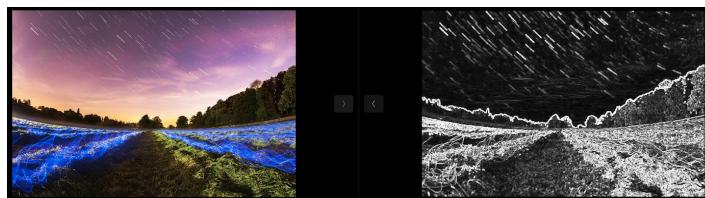
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
50310	curso	20	0	173812	9400	7148	S	7,3	0,1	0:04.52	MATRICES_MULTIPLICACION
1922	curso	20	0	4875028	349140	108744	S	4,3	2,2	9:59.74	gnome-shell
1791	curso	20	0	586500	105040	61328	S	2,3	0,6	66:44.85	Xorg
2851	curso	20	0	2732092	280152	121972	S	1,7	1,7	4:01.49	Isolated Web Co
1524	mssql	20	0	16,6g	917332	58072	S	1,3	5,7	8:05.70	sqlservr
1778	curso	9	-11	4033756	21008	16092	S	1,3	0,1	34:05.00	pulseaudio
2378	curso	20	0	826640	58164	40020	S	1,3	0,4	0:31.63	gnome-terminal
2494	curso	20	0	4553564	567956	274792	S	1,0	3,5	13:14.03	firefox
5974	curso	20	0	2464936	111712	90544	S	1,0	0,7	0:10.07	Isolated Web Co
7493	curso	20	0	2453280	104476	89580	S	0,7	0,6	0:08.36	Isolated Web Co
19475	curso	20	0	2604780	224220	127264	S	0,7	1,4	0:24.98	Isolated Web Co
39751	curso	20	0	2698352	230568	121960	S	0,7	1,4	0:16.59	Isolated Web Co
46435	curso	20	0	14892	4416	3320	R	0,0	0:05.83	top	
902	avahi	20	0	12044	7408	3476	S	0,3	0,0	2:19.18	avahi-daemon
929	root	20	0	2060844	34268	21248	S	0,3	0,2	0:11.32	snapd
7538	curso	20	0	2451628	103616	89660	S	0,3	0,6	0:08.61	Isolated Web Co
7545	curso	20	0	2449552	99168	85844	S	0,3	0,6	0:08.57	Isolated Web Co
45713	root	20	0	0	0	0	I	0,3	0,0	0:00.70	kworker/u24:0-even+
48528	gnome	20	0	1157,7g	108552	77012	S	0,2	1,2	0:02.88	gnome

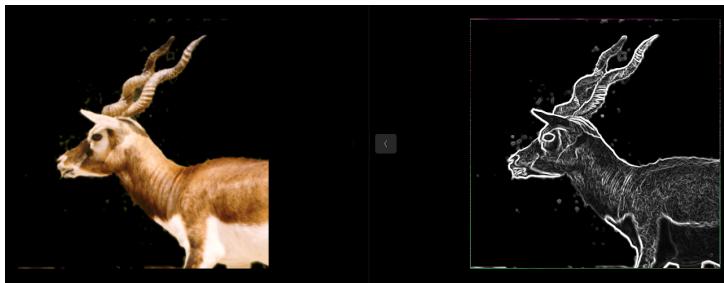
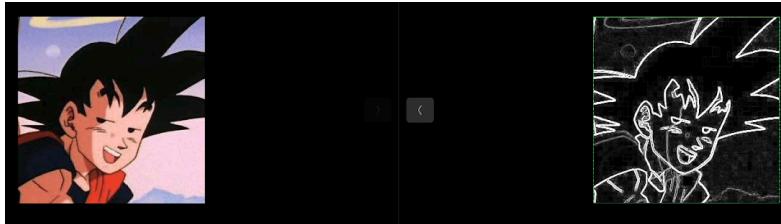
Cada proceso usó aproximadamente 11,500 KB de memoria, lo que indica un consumo moderado suficiente para almacenar las matrices y los datos temporales necesarios. El tiempo de cálculo fue prácticamente nulo en todos los procesos, lo cual sugiere que el

cálculo de la multiplicación fue muy rápido, probablemente debido al tamaño reducido de las matrices. Los tiempos de comunicación fueron igualmente bajos, variando entre 0.000001 y 0.000016 segundos, lo que refleja una comunicación eficiente y de baja latencia entre los procesos. Cada proceso envió y recibió solo 16 bytes, sugiriendo una comunicación mínima, probablemente limitada a la sincronización o al intercambio de pequeños fragmentos de las matrices. Finalmente, la matriz 4x4 resultante confirma que cada proceso calculó una parte específica de la matriz final, logrando el resultado esperado de la multiplicación.

## 2. Filtro de imágenes Sobel

Este es el resultado de las imágenes 5 imágenes aplicando sobel es:





### Sobel Serial

```
Prestone Enter para finalizar...
curso@soporte-OptiPlex-7050:~/Desktop/sharedfolder/Taller 2 MPI(1)/src$ gcc -o sobel_serial
sobel_serial.c -lm
curso@soporte-OptiPlex-7050:~/Desktop/sharedfolder/Taller 2 MPI(1)/src$ ./sobel_serial
Imagen 1 procesada. Memoria utilizada: 1635894 bytes
Imagen 2 procesada. Memoria utilizada: 23040054 bytes
Imagen 3 procesada. Memoria utilizada: 12441654 bytes
Imagen 4 procesada. Memoria utilizada: 540054 bytes
Imagen 5 procesada. Memoria utilizada: 1572918 bytes
Presione Enter para finalizar...
curso@soporte-OptiPlex-7050:~/Desktop/sharedfolder/Taller 2 MPI(1)/src$ ./sobel_serial
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2573	curso	39	19	1129060	44160	16536	S	24,3	0,3	1:58.04	tracker-miner-f
57471	curso	20	0	18028	15724	1636	S	10,3	0,1	0:00.31	sobel_serial
2001	curso	20	0	5751756	356252	108536	S	6,0	2,2	8:59.39	gnome-shell
47198	curso	20	0	1487396	102508	51352	S	4,0	0,6	1:12.82	nautilus
1809	curso	20	0	728088	149552	97964	S	3,7	0,9	10:04.16	Xorg
1707	curso	9	-11	4033756	21012	16100	S	3,3	0,1	41:29.56	pulseaudio
10249	curso	20	0	593076	47912	15492	S	2,0	0,3	0:26.66	tracker-store
57013	curso	20	0	953580	101564	82112	S	1,7	0,6	0:12.49	eog
1666	mssql	20	0	16,8g	917844	58272	S	1,3	5,7	9:04.31	sqlservr
2842	curso	20	0	835456	68688	42200	S	0,7	0,4	1:36.88	gnome-terminal
42461	curso	20	0	2609668	229520	125276	S	0,7	1,4	2:00.01	Isolated Web Co
43933	curso	20	0	679420	60828	44776	S	0,7	0,4	1:40.09	gnome-system-mo
1743	curso	20	0	10092	7156	3820	S	0,3	0,0	0:02.83	dbus-daemon
2180	curso	20	0	162848	7468	6680	S	0,3	0,0	0:02.69	at-spi2-registr
2238	curso	20	0	238248	6372	5788	S	0,3	0,0	0:00.10	gvfs-mtp-volume
2860	curso	20	0	164988	6724	5920	S	0,3	0,0	0:00.33	gvfsd-metadata
7671	curso	20	0	1159,6g	290652	102500	S	0,3	1,8	4:43.86	code
7718	curso	20	0	1157,7g	208268	77920	S	0,3	1,3	0:39.17	code
14131	curso	20	0	4912416	730056	306652	S	0,3	4,5	13:22.02	firefox
41321	curso	20	0	2724124	256580	131444	S	0,3	1,6	1:07.16	Isolated Web Co
41648	curso	20	0	2605528	227348	124268	S	0,3	1,4	0:41.51	Isolated Web Co
47275	root	20	0	0	0	0	I	0,3	0,0	0:01.89	kworker/u24:1-e
57481	curso	20	0	14676	4156	3324	R	0,3	0,0	0:00.10	top
57411	root	20	0	0	0	0	I	0,3	0,0	0:00.02	kworker/9:2-eve
1	root	20	0	168632	11904	8420	S	0,0	0,1	0:02.91	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.01	kthreadd
3	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_par_gp
5	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	slub_flushwq
6	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	netns
8	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	kworker/0:0H-eve
10	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	mm_percpu_wq
11	root	20	0	0	0	0	S	0,0	0,0	0:00.00	rcu_tasks_cude

### Sobel con OPENMP 1 Thread

```
SOBEL_OPENMP_C.c: In function 'main':
curso@soporte-OptiPlex-7050:~/Desktop/sharedfolder/Taller 2 MPI(1)/src$ gcc -o SOBEL_OPENMP
sobel_openmp.c -lm -fopenmp
curso@soporte-OptiPlex-7050:~/Desktop/sharedfolder/Taller 2 MPI(1)/src$ ./SOBEL_OPENMP
Imagen 1 procesada con OpenMP. Memoria utilizada: 1635894 bytes
Imagen 2 procesada con OpenMP. Memoria utilizada: 23040054 bytes
Imagen 3 procesada con OpenMP. Memoria utilizada: 12441654 bytes
Imagen 4 procesada con OpenMP. Memoria utilizada: 540054 bytes
Imagen 5 procesada con OpenMP. Memoria utilizada: 1572918 bytes
```



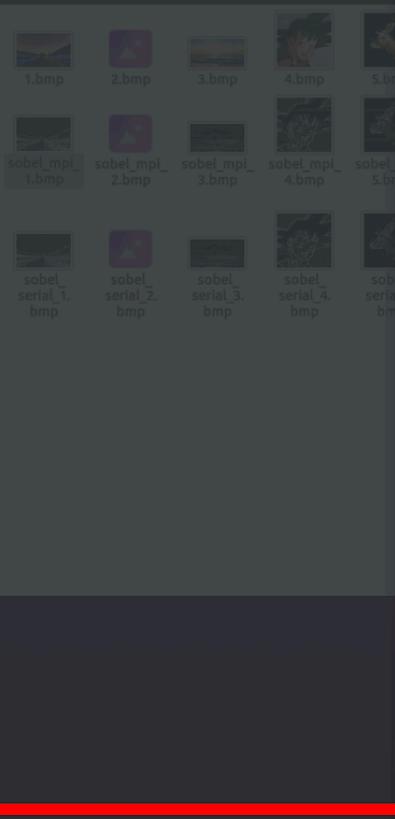
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
58465	curso	20	0	108624	14448	2052	S	64,5	0,1	0:01.96	SOBEL_OPENMP
2573	curso	39	19	1138192	44996	16536	S	16,8	0,3	2:04.14	tracker-miner-f
2091	curso	20	0	5752020	356292	108536	S	10,5	2,2	9:54.81	gnome-sell
58499	curso	39	19	1281328	36724	24560	S	9,9	0,2	0:00.30	tracker-extract
1809	curso	20	0	711688	151940	100352	S	8,2	0,9	11:36.57	g
2842	curso	20	0	836096	69384	42200	S	3,9	0,4	1:42.43	gnome-terminal-
1707	curso	9	-11	3771612	21028	16116	S	3,3	0,1	41:47.83	pulseaudio
10249	curso	20	0	593076	47924	15492	S	2,0	0,3	0:28.52	tracker-store
47198	curso	20	0	1487396	102544	51352	S	2,0	0,6	1:33.43	nautilus

## OPEN MPI

```

MPI-1 SOBEL
MPI-MPI OPENMPI
ursos@optiPlex-7050:~/Desktop/sharedfolder/Taller 2 MPI(1)/src$ mpirun --hostfile /etc/hosts
np 4 SOBEL_MPI
>> Proceso [1] Reporte de Métricas para imagen 1 <<
memoria Máxima Usada: 11528 KB
tiempo de Cómputo: 0.004914 segundos
tiempo de Comunicación: 0.006268 segundos
atos Enviados: 205440 bytes
atos Recibidos: 205440 bytes
-----
>> Proceso [2] Reporte de Métricas para imagen 1 <<
memoria Máxima Usada: 11567 KB
tiempo de Cómputo: 0.010836 segundos
tiempo de Comunicación: 0.000969 segundos
atos Enviados: 203520 bytes
atos Recibidos: 203520 bytes
-----
>> Proceso [3] Reporte de Métricas para imagen 1 <<
memoria Máxima Usada: 11512 KB
tiempo de Cómputo: 0.006827 segundos
tiempo de Comunicación: 0.005277 segundos
atos Enviados: 203520 bytes
atos Recibidos: 203520 bytes
-----
>> Proceso [0] Reporte de Métricas para imagen 1 <<
memoria Máxima Usada: 12128 KB
tiempo de Cómputo: 0.004948 segundos
tiempo de Comunicación: 0.035501 segundos
atos Enviados: 205440 bytes
atos Recibidos: 205440 bytes
-----
>> Proceso [1] Reporte de Métricas para imagen 2 <<
memoria Máxima Usada: 16092 KB
tiempo de Cómputo: 0.044221 segundos
tiempo de Comunicación: 0.215262 segundos
atos Enviados: 2880000 bytes
atos Recibidos: 2880000 bytes
-----
>> Proceso [2] Reporte de Métricas para imagen 2 <<
memoria Máxima Usada: 16308 KB
tiempo de Cómputo: 0.042599 segundos
tiempo de Comunicación: 0.206035 segundos
atos Enviados: 2880000 bytes
atos Recibidos: 2880000 bytes
-----
>> Proceso [3] Reporte de Métricas para imagen 2 <<

```



```

402 total, 1 running, 400 sleeping, 0 stopped, 1 zombie
Cpu(s): 1,7 us, 1,1 sy, 0,0 ni, 97,0 id, 0,1 wa, 0,0 hi, 0,0 si, 0,0 st
kB Mem : 15783,5 total, 2872,8 free, 5310,9 used, 7599,9 buff/cache
kB Swap: 2048,0 total, 2048,0 free, 0,0 used, 9068,8 avail Mem

```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
59630	curso	20	0	183076	13188	7428	S	9,0	0,1	0:03.60	SOBEL_MPI
2091	curso	20	0	5753680	356224	108536	S	6,6	2,2	10:10.86	gnome-shell

### *Análisis de Filtro Sobel*

Imagen	Tamaño de la Imagen (bytes)	Memoria Usada Serial y OpenMP (KB)	Memoria Usada MPI (KB)	Tiempo de Cómputo MPI (s)	Tiempo de Comunicación MPI (s)
1	1,635,894	1,598	46,844	0.010836	0.035501
2	23,040,054	22,500	86,600	0.045974	0.303044
3	12,441,654	12,150	86,600	0.023722	0.212114
4	540,054	527	86,600	0.000974	0.013333
5	1,572,918	1,536	86,600	0.002853	0.034731

En la versión serial y OpenMP, el uso de memoria es el mismo, ya que ambos procesan la imagen en un solo nodo. En MPI, la memoria reportada suma la máxima usada por cada proceso, lo que resulta en un mayor consumo, especialmente evidente en imágenes pequeñas.

En tiempos de procesamiento, MPI muestra tiempos de cómputo menores gracias al paralelismo, aunque no tenemos datos comparativos con serial y OpenMP. La comunicación en MPI añade sobrecarga, especialmente en imágenes grandes, lo que puede impactar el rendimiento.

En resumen, serial y OpenMP son más eficientes en memoria y simples de implementar, mientras que MPI permite paralelismo en varios nodos, aunque con mayor consumo de memoria y sobrecarga de comunicación.

## Conclusiones

En conclusión, Message Passing Interface (MPI) y OpenMP son herramientas fundamentales para la programación paralela, pero se aplican en diferentes contextos. Mientras que MPI se adapta a sistemas distribuidos con múltiples nodos, permitiendo el intercambio de datos mediante mensajes, OpenMP está diseñado para sistemas con memoria compartida, donde se facilita la paralelización de tareas sin necesidad de manejar la comunicación explícitamente entre procesos.

La implementación de programas con MPI, como el ejemplo de la suma distribuida o la multiplicación de matrices, muestra cómo se pueden distribuir eficientemente las tareas en un clúster, mejorando los tiempos de cómputo mediante el uso de múltiples procesos. Sin embargo, el costo de comunicación puede ser significativo, como se observó en el análisis de filtros de imágenes con Sobel, donde la sobrecarga de comunicación y el mayor uso de memoria en MPI representan factores críticos a considerar.

Por otro lado, OpenMP ofrece una solución más directa y eficiente en términos de uso de memoria para problemas que pueden ser resueltos en un único nodo, lo que lo convierte en una opción más sencilla de implementar para ciertas aplicaciones. En general, la elección entre MPI y OpenMP depende del contexto y los recursos disponibles, balanceando la complejidad de implementación, el costo de comunicación y los beneficios del paralelismo en cada caso.