

Trabajo Práctico N° 3

Programación en pasaje de mensajes / Programación híbrida

Características del hardware y del software usados:

- Hardware
 - Procesador: Intel (R) Core (TM) i5- 2310 CPU
 - Núcleos: 4
 - Arquitectura: x86_64
 - Memoria RAM: 7.7 GB
- Software
 - Sistema Operativo: Ubuntu 22.04.1
 - Compilador: GCC version 11.3.0
- Cluster
 - Multicore (partición Blade) conformado por 16 nodos.
 - Cada nodo posee 8GB de RAM
 - 2 procesadores Intel Xeon E5405 de cuatro 4 cores cada uno que operan a 2.0GHz.

Enunciado

Parte 1:

2) Los códigos *blocking.c* y *non-blocking.c* siguen el patrón master-worker, donde los procesos worker le envían un mensaje de texto al master empleando operaciones de comunicación bloqueantes y no bloqueantes, respectivamente. - Compile y ejecute ambos códigos usando $P=\{4,8,16\}$ (no importa que el número de núcleos sea menor que la cantidad de procesos). ¿Cuál de los dos retorna antes el control? - En el caso de la versión no bloqueante, ¿qué sucede si se elimina la operación `MPI_Wait()` (línea 52)? ¿Se imprimen correctamente los mensajes enviados? ¿Por qué?

3) Los códigos *blocking-ring.c* y *non-blocking-ring.c* comunican a los procesos en forma de anillo empleando operaciones bloqueantes y no bloqueantes, respectivamente. Compile y ejecute ambos códigos empleando $P=\{4,8,16\}$ (no importa que el número de núcleos sea

menor que la cantidad de procesos) y $N=\{10000000, 20000000, 40000000, \dots\}$. ¿Cuál de los dos algoritmos requiere menos tiempo de comunicación? ¿Por qué? Nota: Para el caso de $P=16$, agregue la línea `--overcommit` al script de de SLURM y el flag `-- oversubscribe` al comando `mpirun`.

2) blocking

4:

```
Tiempo transcurrido 0.000002 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 1)
Tiempo transcurrido 2.000175 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 1
Tiempo transcurrido 2.000188 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 2)
Tiempo transcurrido 4.000129 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 2
Tiempo transcurrido 4.000141 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 3)
Tiempo transcurrido 6.000100 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 3
Tiempo total = 0.000000 (s)
```

8:

```
Tiempo transcurrido 0.000002 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 1)
Tiempo transcurrido 2.000069 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 1
Tiempo transcurrido 2.000238 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 2)
Tiempo transcurrido 4.000073 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 2
Tiempo transcurrido 4.000085 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 3)
Tiempo transcurrido 6.000047 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 3
Tiempo transcurrido 6.000725 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 4)
Tiempo transcurrido 7.999951 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 4
Tiempo transcurrido 7.999963 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 5)
Tiempo transcurrido 10.000223 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 5
Tiempo transcurrido 10.000238 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 6)
Tiempo transcurrido 12.000136 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 6
Tiempo transcurrido 12.000149 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 7)
Tiempo transcurrido 13.999980 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 7
Tiempo total = 0.000000 (s)
```

16: #SBATCH - -overcommit

```
Tiempo transcurrido 0.000001 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 1)
Tiempo transcurrido 2.000225 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 1
Tiempo transcurrido 2.000238 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 2)
Tiempo transcurrido 4.000000 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 2
Tiempo transcurrido 4.000012 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 3)
Tiempo transcurrido 6.000713 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 3
Tiempo transcurrido 6.000725 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 4)
Tiempo transcurrido 8.000042 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 4
Tiempo transcurrido 8.000055 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 5)
Tiempo transcurrido 9.999993 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 5
Tiempo transcurrido 10.000006 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 6)
Tiempo transcurrido 12.000081 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 6
Tiempo transcurrido 12.000095 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 7)
Tiempo transcurrido 14.000069 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 7
Tiempo transcurrido 14.000083 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 8)
Tiempo transcurrido 15.999910 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 8
Tiempo transcurrido 15.999923 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 9)
Tiempo transcurrido 17.999991 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 9
Tiempo transcurrido 18.000005 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 10)
Tiempo transcurrido 20.000035 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 10
Tiempo transcurrido 20.000049 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 11)
Tiempo transcurrido 22.000284 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 11
Tiempo transcurrido 22.000299 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 12)
Tiempo transcurrido 24.000100 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 12
Tiempo transcurrido 24.000171 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 13)
Tiempo transcurrido 26.000187 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 13
Tiempo transcurrido 26.000202 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 14)
Tiempo transcurrido 28.000356 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 14
Tiempo transcurrido 28.000371 (s): proceso 0, llamando a MPI_Recv() [bloqueante] (fuente rank 15)
Tiempo transcurrido 30.000134 (s): proceso 0, MPI_Recv() devolvio control con mensaje: Hola Mundo! Soy el proceso 15
Tiempo total = 0.000000 (s)
```

non-blocking

4:

```
Tiempo transcurrido 0.000001 (s): proceso 0, llamando a MPI_IRecv() [no bloqueante] (fuente rank 1)
Tiempo transcurrido 0.000053 (s): proceso 0, MPI_IRecv() devolvio el control..
..pero el mensaje no fue aun recibido..
Tiempo transcurrido 2.000221 (s): proceso 0, operacion receive completa con mensaje: Hola Mundo! Soy el proceso 1
Tiempo transcurrido 2.000235 (s): proceso 0, llamando a MPI_IRecv() [no bloqueante] (fuente rank 2)
Tiempo transcurrido 2.000253 (s): proceso 0, MPI_IRecv() devolvio el control..
..pero el mensaje no fue aun recibido..
Tiempo transcurrido 4.000190 (s): proceso 0, operacion receive completa con mensaje: Hola Mundo! Soy el proceso 2
Tiempo transcurrido 4.000202 (s): proceso 0, llamando a MPI_IRecv() [no bloqueante] (fuente rank 3)
Tiempo transcurrido 4.000209 (s): proceso 0, MPI_IRecv() devolvio el control..
..pero el mensaje no fue aun recibido..
Tiempo transcurrido 6.000309 (s): proceso 0, operacion receive completa con mensaje: Hola Mundo! Soy el proceso 3
Tiempo total = 0.000000 (s)
```

8:

```
Tiempo transcurrido 0.000002 (s): proceso 0, llamando a MPI_IRecv() [no bloqueante] (fuente rank 1)
Tiempo transcurrido 0.000044 (s): proceso 0, MPI_IRecv() devolvio el control..
..pero el mensaje no fue aun recibido..
Tiempo transcurrido 2.000111 (s): proceso 0, operacion receive completa con mensaje: Hola Mundo! Soy el proceso 1
Tiempo transcurrido 2.000124 (s): proceso 0, llamando a MPI_IRecv() [no bloqueante] (fuente rank 2)
Tiempo transcurrido 2.000132 (s): proceso 0, MPI_IRecv() devolvio el control..
..pero el mensaje no fue aun recibido..
Tiempo transcurrido 4.000053 (s): proceso 0, operacion receive completa con mensaje: Hola Mundo! Soy el proceso 2
Tiempo transcurrido 4.000065 (s): proceso 0, llamando a MPI_IRecv() [no bloqueante] (fuente rank 3)
Tiempo transcurrido 4.000073 (s): proceso 0, MPI_IRecv() devolvio el control..
..pero el mensaje no fue aun recibido..
Tiempo transcurrido 5.999954 (s): proceso 0, operacion receive completa con mensaje: Hola Mundo! Soy el proceso 3
Tiempo transcurrido 5.999968 (s): proceso 0, llamando a MPI_IRecv() [no bloqueante] (fuente rank 4)
Tiempo transcurrido 5.999976 (s): proceso 0, MPI_IRecv() devolvio el control..
..pero el mensaje no fue aun recibido..
Tiempo transcurrido 8.000041 (s): proceso 0, operacion receive completa con mensaje: Hola Mundo! Soy el proceso 4
Tiempo transcurrido 8.000054 (s): proceso 0, llamando a MPI_IRecv() [no bloqueante] (fuente rank 5)
Tiempo transcurrido 8.000062 (s): proceso 0, MPI_IRecv() devolvio el control..
..pero el mensaje no fue aun recibido..
Tiempo transcurrido 10.000102 (s): proceso 0, operacion receive completa con mensaje: Hola Mundo! Soy el proceso 5
Tiempo transcurrido 10.000116 (s): proceso 0, llamando a MPI_IRecv() [no bloqueante] (fuente rank 6)
Tiempo transcurrido 10.000123 (s): proceso 0, MPI_IRecv() devolvio el control..
..pero el mensaje no fue aun recibido..
Tiempo transcurrido 11.999954 (s): proceso 0, operacion receive completa con mensaje: Hola Mundo! Soy el proceso 6
Tiempo transcurrido 11.999966 (s): proceso 0, llamando a MPI_IRecv() [no bloqueante] (fuente rank 7)
Tiempo transcurrido 11.999975 (s): proceso 0, MPI_IRecv() devolvio el control..
..pero el mensaje no fue aun recibido..
Tiempo transcurrido 13.999827 (s): proceso 0, operacion receive completa con mensaje: Hola Mundo! Soy el proceso 7
Tiempo total = 0.000000 (s)
```

16: #SBATCH - -overcommit

Si borramos la línea 52 MPI_Wait()

```
Tiempo transcurrido 0.000002 (s): proceso 0, llamando a MPI_IRecv() [no bloqueante] (fuente rank 1)
Tiempo transcurrido 0.000050 (s): proceso 0, MPI_IRecv() devolvio el control..
..pero el mensaje no fue aun recibido..
Tiempo transcurrido 0.000074 (s): proceso 0, operacion receive completa con mensaje: No deberia estar leyendo esta frase.
Tiempo transcurrido 0.000085 (s): proceso 0, llamando a MPI_IRecv() [no bloqueante] (fuente rank 2)
Tiempo transcurrido 0.000098 (s): proceso 0, MPI_IRecv() devolvio el control..
..pero el mensaje no fue aun recibido..
Tiempo transcurrido 0.000113 (s): proceso 0, operacion receive completa con mensaje: No deberia estar leyendo esta frase.
Tiempo transcurrido 0.000123 (s): proceso 0, llamando a MPI_IRecv() [no bloqueante] (fuente rank 3)
Tiempo transcurrido 0.000133 (s): proceso 0, MPI_IRecv() devolvio el control..
..pero el mensaje no fue aun recibido..
Tiempo transcurrido 0.000144 (s): proceso 0, operacion receive completa con mensaje: No deberia estar leyendo esta frase.
Tiempo transcurrido 0.000155 (s): proceso 0, llamando a MPI_IRecv() [no bloqueante] (fuente rank 4)
Tiempo transcurrido 0.000165 (s): proceso 0, MPI_IRecv() devolvio el control..
..pero el mensaje no fue aun recibido..
Tiempo transcurrido 0.000176 (s): proceso 0, operacion receive completa con mensaje: No deberia estar leyendo esta frase.
Tiempo transcurrido 0.000185 (s): proceso 0, llamando a MPI_IRecv() [no bloqueante] (fuente rank 5)
Tiempo transcurrido 0.000237 (s): proceso 0, MPI_IRecv() devolvio el control..
..pero el mensaje no fue aun recibido..
```

Ambos algoritmos tardan lo mismo (una diferencia de milésimas, siendo non-blocking un poco más rápido) en cuanto a tiempo de ejecución, esto se da porque son equivalentes en cuanto a resolver el problema. Sin embargo, si hablamos del control, el algoritmo non-blocking retorna antes porque no tiene que esperar a establecer una comunicación, sino que solo deposita en el buffer y sigue con su trabajo. En cambio en el algoritmo bloqueante (blocking.c), un proceso se bloquea hasta que terminan las operaciones de envío o recepción. Por esta razón, el maestro debe esperar a que cada proceso envíe su mensaje antes de poder recibirlo.

En el caso de que se borre del código non-blocking la línea 52 que tiene la operación MPI_Wait() lo que ocurre es que va a haber una gran mejora del tiempo pero imprimiría mal, ya que en vez de imprimir los mensajes enviados por los procesos se puede leer la frase "No se debería estar leyendo esta frase", que es el valor inicial asignado al buffer. Esto pasa porque al no esperar la llegada del mensaje con MPI_Wait() el proceso master imprime inmediatamente el contenido del buffer que no se actualizó con el mensaje esperado.

Capturas tiempos blocking-ring y non-blocking-ring →  CAPTURAS c/ SCRIPTS PARTE 1

3) blocking-ring.c

Tiempos de comunicación:

	10000000	20000000	40000000
4	0.250177	0.496232	0.983008

8	0.567713	1.134899	2.257469
16	2.379903	4.664497	-

non-blocking-ring.c

Tiempos de comunicación:

	10000000	20000000	40000000
4	0.220835	0.443037	0.878137
8	0.296604	0.588943	1.179638
16	1.661709	3.432739	-

Los algoritmos no bloqueantes (como el "non-blocking-ring") son más eficientes en términos de tiempos de comunicación al utilizar operaciones como MPI_Irecv() y MPI_Isend(). Estas permiten realizar otras tareas mientras las operaciones de comunicación se ejecutan en segundo plano, evitando demoras causadas por la espera completa de datos. En contraste, los algoritmos bloqueantes, que emplean MPI_Recv() y MPI_Send(), requieren que el proceso espere hasta que la operación de comunicación se complete antes de continuar, lo que puede generar tiempos de inactividad.

	Ventajas	Desventajas
Bloqueantes (MPI_Send y MPI_Recv)	-Simplicidad de Programación -Seguridad en Buffers	-Tiempos Ociosos -Riesgo de Deadlocks
No Bloqueantes (MPI_Irecv y MPI_Isend)	-Mayor Paralelismo -Reducción de Deadlocks	-Mayor Complejidad de Programación

Podemos observar que hay un costo al aplicar mensajes bloqueantes, a medida que van en aumento la cantidad de procesos disminuye la eficiencia, esto se debe a que estamos utilizando valores de N "bajos", si N fuera aún más grande el tiempo de comunicación sería bajo comparado con el tiempo de cómputo, por lo que obtendremos en esos casos una buena eficiencia con un alto número de procesos. En el caso del algoritmo no bloqueante, el tiempo no es tan alto como su contraparte bloqueante, el resultado es parecido en cuanto a que obtenemos una mejor eficiencia con un número bajo de procesos. En este caso se puede ver mejor como la eficiencia de este algoritmo con respecto a la cantidad de

P (procesos) que se usen para la ejecución va a estar relacionado a la entrada N. Cabe destacar que se ve un aumento al pasar de utilizar 1 nodo a 2 nodos, este es el caso de, por ejemplo, usar 4 y 8 procesos, que se ejecutan en un solo nodo, a pasar a 16 procesos que se ejecutan 8 en cada nodo. Probamos esto con el algoritmo blocking-ring.c, ejecutando 4 procesos, 2 en cada nodo. El caso en el que se ejecutan los procesos en nodos separados tiene un tiempo mayor a su contraparte de usar 4 procesos en un solo nodo, por lo que podríamos decir que también hay un costo extra en la comunicación entre nodos, hay un aumento en el overhead significativo.

Como conclusión podríamos decir que con el algoritmo no bloqueante (non-blocking-ring.c) obtenemos mejoras con respecto al overhead, gastos de cómputo y ejecución, pero dejamos de lado la simpleza en el código, tenemos que llegar a una solución en la que haya sincronización entre los procesos mediante la semántica del algoritmo. El no bloqueante tiene mayor performance que el bloqueante, por lo que dejando de lado los casos en los que realmente necesitemos aplicar una comunicación bloqueante y no tengamos otra opción, nos convendría más establecer comunicación no bloqueante, para permitir al proceso emisor poder seguir con su operación local.

Cabe destacar que si utilizamos la comunicación no bloqueante se debe estar controlando de no provocar deadlock debido a un mal orden del flujo de instrucciones.

Parte 2:

Dada la siguiente expresión:

$$R = \frac{(MaxA \times MaxB - MinA \times MinB)}{PromA \times PromB} \times [A \times B] + [C \times B^T]$$

- ***Donde A, B, C y R son matrices cuadradas de NxN con elementos de tipo double.***
- ***MaxA, MinA y PromA son los valores máximo, mínimo y promedio de la matriz A, respectivamente.***
- ***MaxB, MinB y PromB son los valores máximo, mínimo y promedio de la matriz B, respectivamente.***
- ***B^T Es la matriz transpuesta de B.***

Desarrolle 2 algoritmos que computen la expresión dada:

1. Algoritmo paralelo empleando MPI

Para todos los algoritmos planteados seguimos usando el nivel de optimización -O3

La implementación del algoritmo MPI sigue el modelo SPMD (Single Program Multiple Data), donde todos los procesos ejecutan el mismo código pero realizan tareas diferenciadas según su identificador (rank). A continuación se mencionan las instrucciones MPI empleadas:

-Inicialización y configuración:

- MPI_Init(), MPI_Comm_size() y MPI_Comm_rank(): inicializan el entorno MPI, determinan el número de procesos y asignan un identificador único a cada proceso.
- MPI_Barrier(): para sincronizar los procesos antes de medir los tiempos.

-Distribución de trabajo:

- MPI_Scatter(): distribuye porciones de las matrices A y C a cada proceso, permitiendo la paralelización de cálculos.
- MPI_Bcast(): se utiliza para compartir las matrices B y BT completas con todos los procesos, necesarias para los cálculos.

-Cálculos locales:

- Cada proceso calcula los valores mínimos, máximos y promedio de sus porciones asignadas de las matrices A y B.
- La multiplicación de matrices se realiza localmente en bloques para optimizar el uso de memoria y mejorar el rendimiento en grandes tamaños de matrices.

-Reducción y sincronización:

- MPI_Reduce(): combina los resultados de todos los procesos (mínimos, máximos, promedios) en el proceso coordinador.
- El escalar calculado por el coordinador se transmite a todos los procesos usando MPI_Bcast().

-Recolección de resultados:

- MPI_Gather(): recolecta las porciones de la matriz R generadas por cada proceso y las ensambla en el proceso coordinador.

-Medición de tiempos:

- MPI_Wtime(): para medir tiempos de comunicación y cálculo en distintas etapas del algoritmo (distribución inicial, recolección min/max/prom, transmisión escalar y recolección final de resultados)

ScriptMPI.sh (8 procesos)

```
#!/bin/bash

#SBATCH -N 1 //un nodo con 8 procesos

#SBATCH --exclusive

#SBATCH --tasks-per-node=8

#SBATCH -o /nethome/spusuariol4/Entrega3/Parte2/outputMPI.txt

#SBATCH -e /nethome/spusuariol4/Entrega3/Parte2/erroresMPI.txt

mpirun salida_mpiV2 $1
```

```
spusuariol4@frontend:~/Entrega3/Parte2$ sbatch ./miScriptMPI.sh 512
Submitted batch job 158778
spusuariol4@frontend:~/Entrega3/Parte2$ squeue
      JOBID PARTITION    NAME    USER ST       TIME  NODES NODELIST(REASON)
spusuariol4@frontend:~/Entrega3/Parte2$ cat outputMPI.txt
N=512 P=8 Tiempo total=0.093338 Tiempo comunicacion=0.009830
spusuariol4@frontend:~/Entrega3/Parte2$ sbatch ./miScriptMPI.sh 1024
Submitted batch job 158781
spusuariol4@frontend:~/Entrega3/Parte2$ squeue
      JOBID PARTITION    NAME    USER ST       TIME  NODES NODELIST(REASON)
    158780      Blade sE2_sec. sdyp27  R       0:29      1 nodol
spusuariol4@frontend:~/Entrega3/Parte2$ cat outputMPI.txt
N=1024 P=8 Tiempo total=0.638879 Tiempo comunicacion=0.034044
spusuariol4@frontend:~/Entrega3/Parte2$ sbatch ./miScriptMPI.sh 2048
Submitted batch job 158783
spusuariol4@frontend:~/Entrega3/Parte2$ squeue
      JOBID PARTITION    NAME    USER ST       TIME  NODES NODELIST(REASON)
    158780      Blade sE2_sec. sdyp27  R       1:02      1 nodol
    158783      Blade miScript spusuari  R       0:02      1 nodo2
spusuariol4@frontend:~/Entrega3/Parte2$ squeue
      JOBID PARTITION    NAME    USER ST       TIME  NODES NODELIST(REASON)
    158780      Blade sE2_sec. sdyp27  R       1:05      1 nodol
spusuariol4@frontend:~/Entrega3/Parte2$ cat outputMPI.tx
cat: outputMPI.tx: No existe el fichero o el directorio
spusuariol4@frontend:~/Entrega3/Parte2$ cat outputMPI.txt
N=2048 P=8 Tiempo total=4.583101 Tiempo comunicacion=0.133629
spusuariol4@frontend:~/Entrega3/Parte2$ sbatch ./miScriptMPI.sh 4096
Submitted batch job 158784
spusuariol4@frontend:~/Entrega3/Parte2$ squeue
      JOBID PARTITION    NAME    USER ST       TIME  NODES NODELIST(REASON)
    158780      Blade sE2_sec. sdyp27  R       1:52      1 nodol
    158784      Blade miScript spusuari  R       0:02      1 nodo2
spusuariol4@frontend:~/Entrega3/Parte2$ squeue
      JOBID PARTITION    NAME    USER ST       TIME  NODES NODELIST(REASON)
    158780      Blade sE2_sec. sdyp27  R       1:58      1 nodol
    158784      Blade miScript spusuari  R       0:08      1 nodo2
spusuariol4@frontend:~/Entrega3/Parte2$ squeue
      JOBID PARTITION    NAME    USER ST       TIME  NODES NODELIST(REASON)
    158780      Blade sE2_sec. sdyp27  R       2:49      1 nodol
spusuariol4@frontend:~/Entrega3/Parte2$ cat outputMPI.txt
N=4096 P=8 Tiempo total=34.843580 Tiempo comunicacion=2.621063
```

ScriptMPI.sh (16 procesos)

```
#!/bin/bash

#SBATCH -N 2 //tengo 2 nodos con 4 procesos cada uno

#SBATCH --exclusive

#SBATCH --tasks-per-node=8

#SBATCH -o /nethome/spusuariol4/Entrega3/Parte2/outputMPI.txt

#SBATCH -e /nethome/spusuariol4/Entrega3/Parte2/erroresMPI.txt

mpirun salida_mpiV2 $1
```

```
spusuariol4@frontend:~/Entrega3/Parte2$ sbatch ./miScriptMPI.sh 512
Submitted batch job 158769
spusuariol4@frontend:~/Entrega3/Parte2$ cat outputMPI.txt
N=512 P=16 Tiempo total=0.107202 Tiempo comunicacion=0.047345
spusuariol4@frontend:~/Entrega3/Parte2$ sbatch ./miScriptMPI.sh 1024
Submitted batch job 158772
spusuariol4@frontend:~/Entrega3/Parte2$ cat outputMPI.txt
N=1024 P=16 Tiempo total=0.670899 Tiempo comunicacion=0.184878
spusuariol4@frontend:~/Entrega3/Parte2$ sbatch ./miScriptMPI.sh 2048
Submitted batch job 158774
spusuariol4@frontend:~/Entrega3/Parte2$ squeue
      JOBID PARTITION    NAME    USER ST       TIME  NODES NODELIST(REASON)
      158773      Blade sE2_sec. sdyp27  R        0:06       1 nodol
      158774      Blade miScript spusuari  R        0:01       2 nodo[2-3]
spusuariol4@frontend:~/Entrega3/Parte2$ squeue
      JOBID PARTITION    NAME    USER ST       TIME  NODES NODELIST(REASON)
      158774      Blade miScript spusuari  CG       0:05       2 nodo[2-3]
      158773      Blade sE2_sec. sdyp27  R        0:10       1 nodol
spusuariol4@frontend:~/Entrega3/Parte2$ squeue
      JOBID PARTITION    NAME    USER ST       TIME  NODES NODELIST(REASON)
      158773      Blade sE2_sec. sdyp27  R        0:12       1 nodol
spusuariol4@frontend:~/Entrega3/Parte2$ cat outputMPI.txt
N=2048 P=16 Tiempo total=3.641806 Tiempo comunicacion=0.831323
spusuariol4@frontend:~/Entrega3/Parte2$ sbatch ./miScriptMPI.sh 4096
Submitted batch job 158775
spusuariol4@frontend:~/Entrega3/Parte2$ squeue
      JOBID PARTITION    NAME    USER ST       TIME  NODES NODELIST(REASON)
      158773      Blade sE2_sec. sdyp27  R       1:08       1 nodol
      158775      Blade miScript spusuari  R        0:03       2 nodo[2-3]
spusuariol4@frontend:~/Entrega3/Parte2$ squeue
      JOBID PARTITION    NAME    USER ST       TIME  NODES NODELIST(REASON)
      158773      Blade sE2_sec. sdyp27  R       1:41       1 nodol
spusuariol4@frontend:~/Entrega3/Parte2$ cat outputMPI.txt
N=4096 P=16 Tiempo total=24.341929 Tiempo comunicacion=7.130385
spusuariol4@frontend:~/Entrega3/Parte2$
```

ScriptMPI.sh (32 procesos)

```
#!/bin/bash

#SBATCH -N 4 //tengo 4 nodos con 8 procesos cada uno

#SBATCH --exclusive

#SBATCH --tasks-per-node=8

#SBATCH -o /nethome/spusuariol4/Entrega3/Parte2/outputMPI.txt

#SBATCH -e /nethome/spusuariol4/Entrega3/Parte2/erroresMPI.txt

mpirun salida_mpiV2 $1
```

```
spusuariol4@frontend:~/Entrega3/Parte2$ sbatch ./miScriptMPI.sh 512
Submitted batch job 158787
spusuariol4@frontend:~/Entrega3/Parte2$ squeue
      JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
      158780      Blade sE2_sec.  sdyp27  R        4:49        1 nodol
spusuariol4@frontend:~/Entrega3/Parte2$ cat outputMPI.txt
N=512 P=32 Tiempo total=0.161252 Tiempo comunicacion=0.186679
spusuariol4@frontend:~/Entrega3/Parte2$ sbatch ./miScriptMPI.sh 1024
Submitted batch job 158788
spusuariol4@frontend:~/Entrega3/Parte2$ squeue
      JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
      158780      Blade sE2_sec.  sdyp27  R        5:42        1 nodol
spusuariol4@frontend:~/Entrega3/Parte2$ cat outputMPI.txt
N=1024 P=32 Tiempo total=0.628848 Tiempo comunicacion=0.433595
spusuariol4@frontend:~/Entrega3/Parte2$ sbatch ./miScriptMPI.sh 2048
Submitted batch job 158789
spusuariol4@frontend:~/Entrega3/Parte2$ squeue
      JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
      158780      Blade sE2_sec.  sdyp27  R        6:22        1 nodol
      158789      Blade miScript spusuari  R        0:02        4 nodo[2-5]
spusuariol4@frontend:~/Entrega3/Parte2$ squeue
      JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
      158780      Blade sE2_sec.  sdyp27  R        6:35        1 nodol
spusuariol4@frontend:~/Entrega3/Parte2$ cat outputMPI.txt
N=2048 P=32 Tiempo total=3.039804 Tiempo comunicacion=0.914409
spusuariol4@frontend:~/Entrega3/Parte2$ sbatch ./miScriptMPI.sh 4096
Submitted batch job 158791
spusuariol4@frontend:~/Entrega3/Parte2$ squeue
      JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
      158780      Blade sE2_sec.  sdyp27  R        7:27        1 nodol
      158791      Blade miScript spusuari  R        0:04        4 nodo[2-5]
spusuariol4@frontend:~/Entrega3/Parte2$ squeue
      JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
      158780      Blade sE2_sec.  sdyp27  R        7:52        1 nodol
spusuariol4@frontend:~/Entrega3/Parte2$ cat outputMPI.txt
N=4096 P=32 Tiempo total=17.365256 Tiempo comunicacion=6.117186
spusuariol4@frontend:~/Entrega3/Parte2$
```

MPI

Tiempo ejecución:

	<i>N = 512</i>	<i>N = 1024</i>	<i>N = 2048</i>	<i>N = 4096</i>
<i>P=8</i>	0.091027	0.637466	4.579412	34.531997
<i>P=16</i>	0.114295	0.634839	3.459959	21.845668
<i>P=32</i>	0.150907	0.641705	2.851908	16.685637

Tiempo comunicacion:

	<i>N = 512</i>	<i>N = 1024</i>	<i>N = 2048</i>	<i>N = 4096</i>
<i>P=8</i>	0.035467	0.171986	0.828539	4.527408
<i>P=16</i>	0.117004	0.423141	1.622554	6.931908
<i>P=32</i>	0.225553	0.786924	2.08888	10.748482

Híbrido

Tiempo ejecución:

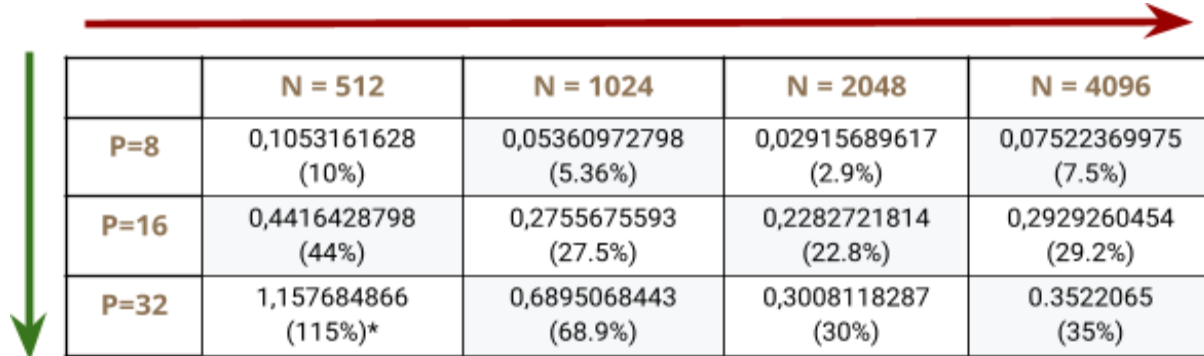
	<i>N = 512</i>	<i>N = 1024</i>	<i>N = 2048</i>	<i>N = 4096</i>
<i>P=16</i>	0.097724	0.573023	3.465727	21.0493750
<i>P=32</i>	0.108342	0.536492	2.491953	16.161110

Tiempo comunicacion:

	<i>N = 512</i>	<i>N = 1024</i>	<i>N = 2048</i>	<i>N = 4096</i>
<i>P=16</i>	0.030656	0.132438	0.480968	1.829034
<i>P=32</i>	0.050826	0.191111	0.705682	8.215382

MPI - Overhead de Comunicación:

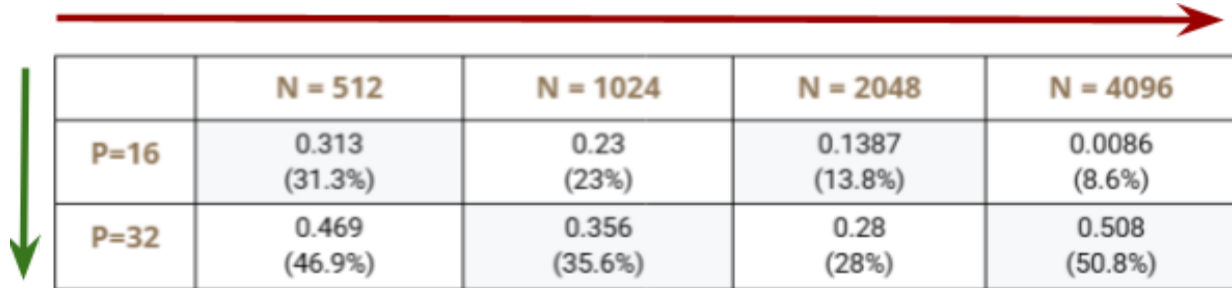
Es el porcentaje del tiempo que un proceso paso comunicándose en lugar de realizar trabajo útil. → Tiempo comunicación/ tiempo total



	N = 512	N = 1024	N = 2048	N = 4096
P=8	0,1053161628 (10%)	0,05360972798 (5.36%)	0,02915689617 (2.9%)	0,07522369975 (7.5%)
P=16	0,4416428798 (44%)	0,2755675593 (27.5%)	0,2282721814 (22.8%)	0,2929260454 (29.2%)
P=32	1,157684866 (115%)*	0,6895068443 (68.9%)	0,3008118287 (30%)	0.3522065 (35%)

*El valor 115% superior al 100% indica que la suma de los tiempos de comunicación en todas las fases supera el tiempo total de ejecución. Esto puede ocurrir debido a: solapamiento de comunicaciones: diferentes procesos realizan comunicaciones simultáneamente y/o tiempos de espera: operaciones bloqueantes introducen demoras que forman parte del tiempo total. En este caso específico, se verificó con el docente que este comportamiento es válido y no representa un error.

Híbrido - Overhead de Comunicación:



	N = 512	N = 1024	N = 2048	N = 4096
P=16	0.313 (31.3%)	0.23 (23%)	0.1387 (13.8%)	0.0086 (8.6%)
P=32	0.469 (46.9%)	0.356 (35.6%)	0.28 (28%)	0.508 (50.8%)

Overhead de comunicación:

En el caso de MPI puro lo que vemos es que cuando N aumenta, la proporción de tiempo de comunicación respecto al tiempo total baja significativamente. Esto refleja que el tiempo de cómputo útil aumenta más rápido que el tiempo de comunicación. En cambio si dejamos fijo N y aumentamos P la proporción del tiempo de comunicación crece, ya que más procesos implican más sincronización y comunicación.

En el caso del híbrido, el overhead de comunicación es menor en general para el mismo número de procesos y tamaño del problema. Por ejemplo: para N=512 y P=16, el overhead de comunicación en MPI puro es del 44%, mientras que en híbrido es del 31.3%. Esto refleja que el uso de hilos (OpenMP) en lugar de más procesos MPI reduce las comunicaciones entre nodos.

Conclusión:

El algoritmo híbrido (MPI+OpenMP) es superior al MPI puro en cuanto a:

- Overhead de comunicación más bajo, especialmente para valores pequeños de N y configuraciones de muchos procesos.
- Mejor escalabilidad, gracias al balance entre procesos MPI e hilos OpenMP.
- Menor costo de comunicación por mantener más cómputo dentro de cada nodo y reducir la frecuencia de sincronizaciones entre nodos.

Enunciado

"... En el caso de $P=8$, compare el rendimiento del algoritmo MPI con el de Pthreads/OpenMP (el que mejor rendimiento haya tenido) del Trabajo Práctico N° 2. "

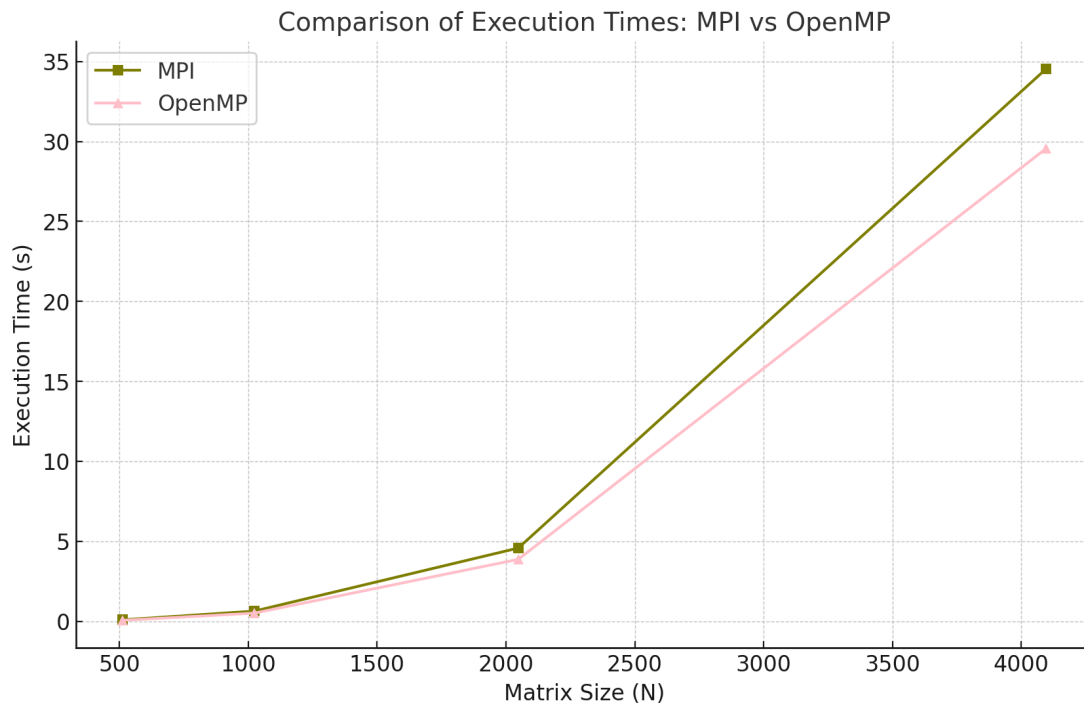
Comparación entre MPI y OpenMP en tiempos de ejecución:

MPI:

	$N = 512$	$N = 1024$	$N = 2048$	$N = 4096$
$P=8$	0.091027	0.637466	4.579412	34.531997

OpenMP:

	$N = 512$	$N = 1024$	$N = 2048$	$N = 4096$
$T=8$	0.060604	0.511298	3.871394	29.558310



La implementación con MPI es más costosa en términos de tiempo de ejecución que las implementaciones con OpenMP para los tamaños N analizados.

Esto puede explicarse por:

- Overhead de comunicación: MPI involucra más sincronización y envío de datos entre procesos, incluso en una sola máquina. En cambio, OpenMP aprovecha la memoria compartida y reduce los costos de comunicación.
- Granularidad del paralelismo: OpenMP opera sobre hilos del mismo proceso, lo que permite una coordinación más eficiente y una sobrecarga menor.
- Escalabilidad: Para valores bajos de P, como 8, los modelos basados en hilos suelen escalar mejor, pero esto puede cambiar con más procesos o distintos entornos de hardware (como múltiples nodos reales).

Enunciado

“...En el caso de $P=\{16,32\}$, compare el rendimiento del algoritmo MPI con el del híbrido.”

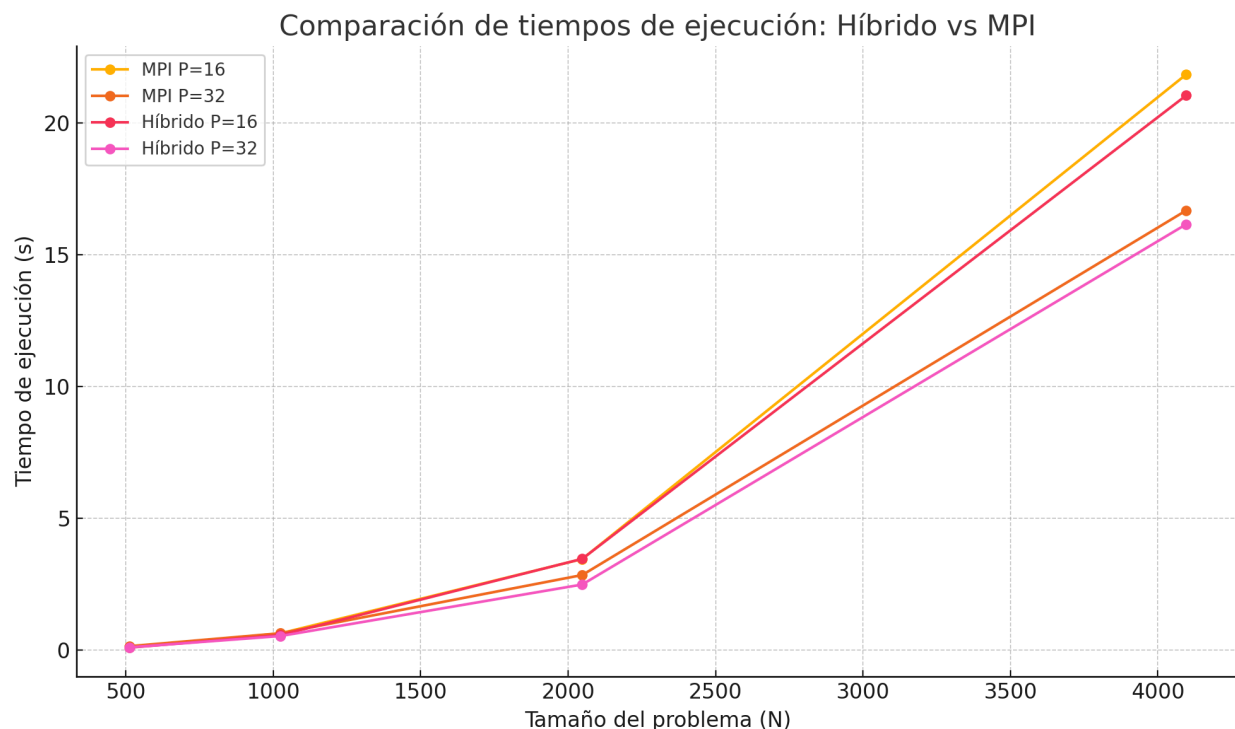
Comparación entre Híbrido y MPI en tiempos de ejecución:

MPI:

	$N = 512$	$N = 1024$	$N = 2048$	$N = 4096$
$P=16$	0.114295	0.634839	3.459959	21.845668
$P=32$	0.150907	0.641705	2.851908	16.685637

Híbrido:

	$N = 512$	$N = 1024$	$N = 2048$	$N = 4096$
$P=16$	0.097724	0.573023	3.465727	21.0493750
$P=32$	0.108342	0.536492	2.491953	16.161110



El enfoque híbrido supera al enfoque MPI en tiempos de ejecución, principalmente porque reduce el overhead asociado al paso de mensajes dentro de un mismo nodo. Al combinar MPI con hilos (como OpenMP), se evita la comunicación innecesaria entre procesos locales, lo que mejora la eficiencia en arquitecturas multinúcleo.

Speedup y eficiencia- Resultados obtenidos



Secuencial (tiempo ejecución)

	<i>N = 512</i>	<i>N = 1024</i>	<i>N = 2048</i>	<i>N = 4096</i>
<i>T=1</i>	0.428034	3.501337	28.008172	226.518864

Speedup: cuantifica cuántas veces más rápido pudimos resolver un problema utilizando un algoritmo paralelo con p unidades de procesamiento, en comparación con el mejor algoritmo secuencial para el mismo problema.



Eficiencia: nos indica el porcentaje de tiempo que un proceso está ejecutando cómputo útil.

MPI - Speedup:





<i>P</i>	<i>N = 512</i>	<i>N = 1024</i>	<i>N = 2048</i>	<i>N = 4096</i>
8	4,585849279	5,513621318	6,111183672	6,501021537
16	3,992779985	5,218873482	7,690736959	9,305707202
32	2,654441495	5,567859006	9,213808522	13,04436928

MPI - Eficiencia:





<i>P</i>	<i>N = 512</i>	<i>N = 1024</i>	<i>N = 2048</i>	<i>N = 4096</i>
8	0,5732311599 (57.3%)	0,6892026647 (68.9%)	0,763897959 (76%)	0,8126276921 (81%)
16	0,2495487491 (24.9%)	0,3261795926 (32.6%)	0,4806710599 (48%)	0,5816067001 (58%)
32	0,08295129673 (8.2%)	0,1739955939 (17.3%)	0,2879315163 (28.7%)	0,40763654 (40%)

Híbrido - Speedup:



	<i>N = 512</i>	<i>N = 1024</i>	<i>N = 2048</i>	<i>N = 4096</i>
P=16	4.3800294	6.110290	8.081470	10.76131
P=32	3.9507670	6.526354	11.23944	14.01629

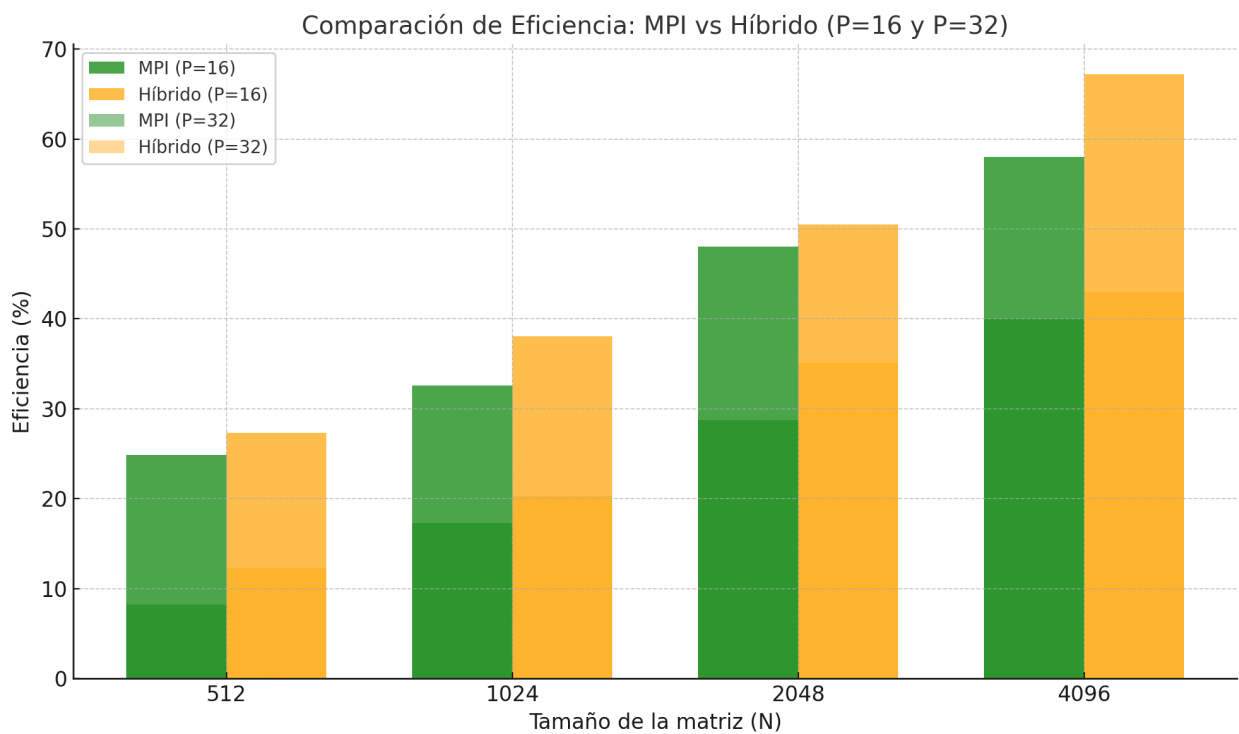
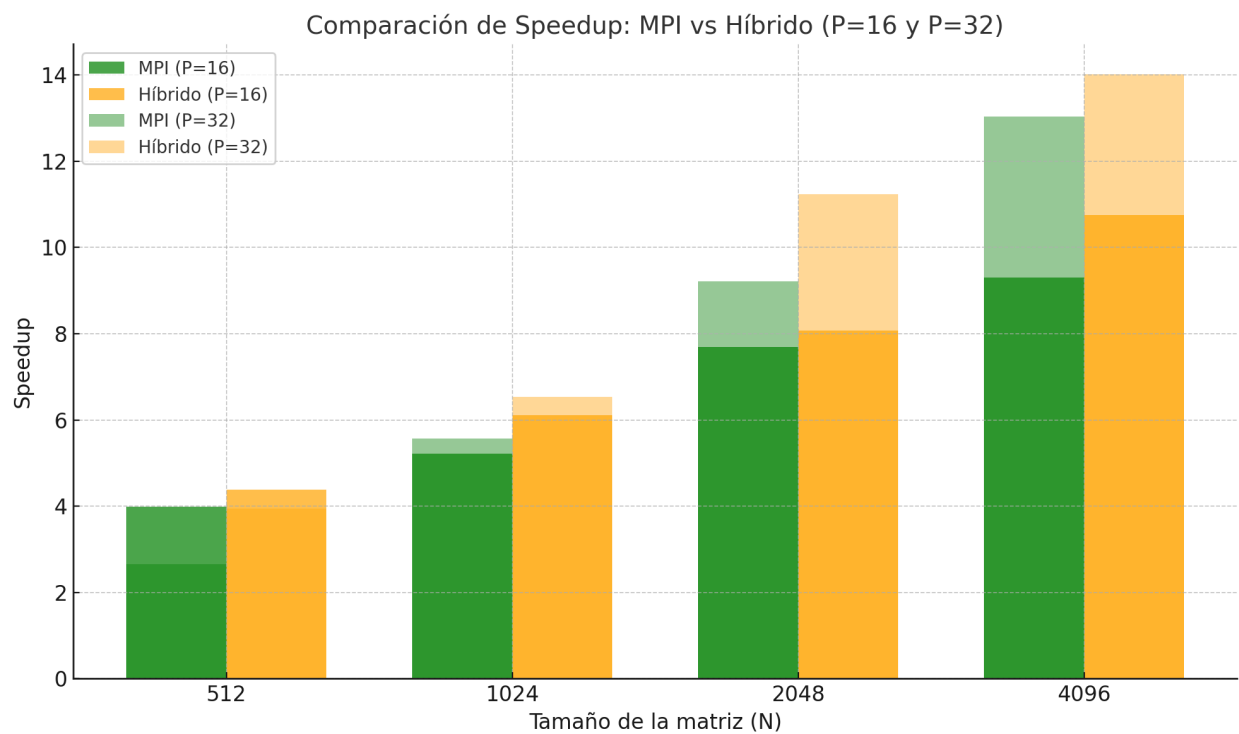
Híbrido - Eficiencia:



	<i>N = 512</i>	<i>N = 1024</i>	<i>N = 2048</i>	<i>N = 4096</i>
P=16	0.2737 (27.3%)	0.3818 (38.1%)	0.505 (50.5%)	0.672 (67.2%)
P=32	0.123 (12.3%)	0.203 (20.3%)	0.351 (35.1%)	0.43 (43%)

Conclusión

- El híbrido logra un mejor Speedup en todos los casos, especialmente para $P=32$, debido a su capacidad de aprovechar los hilos de OpenMP dentro de cada nodo y reducir el overhead de comunicación entre procesos.
- La diferencia es más notoria a medida que aumenta N , donde el híbrido logra aprovechar más el cómputo útil.
- Incremento del Speedup con N : Ambos algoritmos muestran un incremento del Speedup conforme aumenta N , ya que el mayor tamaño de datos amortigua el impacto del overhead de comunicación.
- Diferencia en $P=32$: En $P=32$, el híbrido tiene una clara ventaja para tamaños mayores ($N=2048$ y $N=4096$), logrando un Speedup cercano al ideal para esa cantidad de núcleos.
- El híbrido tiene una eficiencia consistentemente más alta que MPI, especialmente para valores bajos de N , lo que demuestra un mejor balance entre comunicación y cómputo.
- Al aumentar N , la eficiencia mejora significativamente en ambos algoritmos, pero el híbrido sigue siendo superior.
- Para $P=32$, el híbrido mitiga mejor la pérdida de eficiencia que ocurre debido al mayor overhead de comunicación en MPI.



Cálculos speedup, eficiencia y overhead MPI: [+ Entrega 3 SP](#)

2. Algoritmo paralelo híbrido empleando MPI+OpenMP

La implementación del algoritmo híbrido combina el modelo de paso de mensajes (MPI) para la comunicación entre nodos, con el modelo de memoria compartida (OpenMP) para el paralelismo dentro de cada nodo. Esto sigue el modelo SPMD a nivel de procesos MPI, donde cada proceso MPI ejecuta el mismo código pero gestiona sus propios hilos OpenMP para la paralelización local.

Vamos a ejecutar un proceso por cada nodo y cada proceso creará el número de hilos que requiera para ejecutar.

En cuanto al algoritmo empleado consideramos aprovechar comunicaciones colectivas para la parte de MPI ya que están optimizadas y son mucho más eficientes para transmisión de información que hacer lo mismo por mensajes individuales (punto a punto). Se tomó como base el algoritmo OpenMP de la entrega 2 y el algoritmo MPI de esta entrega.

Script.sh (16 procesos)

```
#!/bin/bash
#SBATCH -N 2
#SBATCH --exclusive
#SBATCH --tasks-per-node=1
#SBATCH -o /nethome/spusuariol4/Entrega3/Parte2/HIBRIDO/outputH.txt
#SBATCH -e /nethome/spusuariol4/Entrega3/Parte2/HIBRIDO/errorsH.txt
export OMP_NUM_THREADS= 8
mpirun --bind-to none salida_hibridoMODIFICADO2 $1 $2
```

```
spusuariol4@frontend:~/Entrega3/Parte2/HIBRIDO$ sbatch ./scriptHibrido.sh 512 16
Submitted batch job 159246
spusuariol4@frontend:~/Entrega3/Parte2/HIBRIDO$ cat outputH.txt
N=512 Tiempo total=0.115176 Tiempo comunicacion=0.094527
```

```
spusuariol4@frontend:~/Entrega3/Parte2/HIBRIDO$ sbatch ./scriptHibrido.sh 1024 16
Submitted batch job 159251
spusuariol4@frontend:~/Entrega3/Parte2/HIBRIDO$ squeue
      JOBID PARTITION    NAME    USER  ST       TIME  NODES NODELIST(REASON)
      159201      Blade scriptHi spusuari  R        48:14      2 nodo[1-2]
      159245      Blade runOMP.s spusuari  R         3:00      1 nodol2
      159249      Blade run16Nuc spusuari  R         1:30      2 nodo[13-14]
spusuariol4@frontend:~/Entrega3/Parte2/HIBRIDO$ cat outputH.txt
N=1024 Tiempo total=0.535975 Tiempo comunicacion=0.350898
```

```

N=1024 Tiempo total=0.1373023 Tiempo comunicacion=0.132438
spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ sbatch ./scriptHibrido.sh 512 16
Submitted batch job 159485
spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ cat outputHibrido.txt
N=512 Tiempo total=0.097724 Tiempo comunicacion=0.030656
spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ █

```

```

spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ sbatch ./scriptHibrido.sh 1024 16
Submitted batch job 159484
spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ cat outputHibrido.txt
N=1024 Tiempo total=0.573023 Tiempo comunicacion=0.132438
spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ █

```

```

spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ sbatch ./scriptHibrido.sh 2048 16
Submitted batch job 159486
spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ cat outputHibrido.txt
N=2048 Tiempo total=3.465727 Tiempo comunicacion=0.480968
spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ █

```

```

N=2048 Tiempo total=3.465727 Tiempo comunicacion=0.480968
spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ sbatch ./scriptHibrido.sh 4096 16
Submitted batch job 159488
spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ cat outputHibrido.txt
spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ cat outputHibrido.txt
N=4096 Tiempo total=21.493750 Tiempo comunicacion=1.829034
spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ █

```

Script.sh (32 procesos)

```

#!/bin/bash
#SBATCH -N 4
#SBATCH --exclusive
#SBATCH --tasks-per-node=1
#SBATCH -o /nethome/spusuariol4/Entrega3/Parte2/HIBRIDO/outputH.txt
#SBATCH -e /nethome/spusuariol4/Entrega3/Parte2/HIBRIDO/errorsH.txt
export OMP_NUM_THREADS=8
mpirun --bind-to none salida_hibridoMODIFICADO2 $1 $2

```

```

spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ sbatch ./scriptHibrido.sh 512 32
Submitted batch job 159489
spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ cat outputHibrido.txt
N=512 Tiempo total=0.108342 Tiempo comunicacion=0.050826
spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ █

```

```

N=512 Tiempo total=0.108342 Tiempo comunicacion=0.050826
spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ sbatch ./scriptHibrido.sh 1024 32
Submitted batch job 159490
spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ cat outputHibrido.txt
N=1024 Tiempo total=0.536492 Tiempo comunicacion=0.191111
spusuariol4@frontend:~/Entrega3/Parte2/hibrido_entrega$ █

```

```
spusuario14@frontend:~/Entrega3/Parte2/hibrido_entrega$ sbatch ./scriptHibrido.sh 2048 32
Submitted batch job 159491
spusuario14@frontend:~/Entrega3/Parte2/hibrido_entrega$ cat outputHibrido.txt
N=2048 Tiempo total=2.491953 Tiempo comunicacion=0.705682
spusuario14@frontend:~/Entrega3/Parte2/hibrido_entrega$
```

```
spusuario14@frontend:~/Entrega3/Parte2/hibrido_entrega$ sbatch ./scriptHibrido.sh 4096 32
Submitted batch job 159492
spusuario14@frontend:~/Entrega3/Parte2/hibrido_entrega$ cat outputHibrido.txt
N=4096 Tiempo total=16.161110 Tiempo comunicacion=8.215382
spusuario14@frontend:~/Entrega3/Parte2/hibrido_entrega$
```

Conclusión final:

Las operaciones no bloqueantes como MPI_Isend y MPI_Irecv son más rápidas porque permiten solapar comunicación y cómputo. Sin embargo, si no se sincronizan adecuadamente con MPI_Wait(), pueden dar resultados incorrectos, ya que la comunicación podría no haberse completado antes de usar los datos. Además, al usar múltiples nodos, aparece un overhead extra por las comunicaciones entre nodos.

En cuanto al rendimiento, con pocos procesos ($P=8$, en un solo nodo), OpenMP puro es más eficiente que MPI porque aprovecha la memoria compartida y tiene menos overhead. Cuando el número de procesos aumenta y se distribuyen entre nodos ($P=16$, $P=32$), el modelo híbrido (MPI+OpenMP) supera a MPI puro. Esto se debe a que combina OpenMP para paralelismo dentro del nodo y reduce los mensajes MPI entre nodos, mejorando el tiempo de ejecución, speedup y eficiencia.

Por último, la eficiencia disminuye al aumentar P con un N fijo, ya que crece el overhead de coordinación. Sin embargo, mejora al aumentar N con un P fijo porque el cómputo útil tiene más peso frente a la comunicación.