

UNIVERSIDAD NACIONAL DE SAN JUAN

FACULTAD DE CIENCIAS EXACTAS FÍSICAS Y NATURALES DEPARTAMENTO DE INFORMÁTICA

**Sistemas Distribuidos y Paralelos**

Exposición

Matriz por Vector

OpenMP - MPI

**Alumnos:**

* Coleme Ignacio
* Elizondo Ramiro

AÑO 2025

## Introducción

La multiplicación de una matriz por un vector es una operación fundamental en la computación científica y en múltiples disciplinas como álgebra lineal, gráficos por computadora, inteligencia artificial, y simulaciones físicas. El objetivo de este trabajo es comparar dos enfoques de paralelismo OpenMP y MPI para resolver esta operación de forma eficiente, evaluando el rendimiento y escalabilidad en sistemas multiprocesador.

## Marco teórico

### 2.1. Multiplicación Matriz-Vector

La operación consiste en multiplicar una matriz binaria A de dimensión por un vector binario de dimensión , obteniendo como resultado un vector también de dimensión , tal que:

Dado que tanto los elementos de la matriz A cómo del vector son binarios, la multiplicación equivale a la operación lógica AND.

### 2.2. Paralelismo con OpenMP

**OpenMP** es una API basada en directivas que permite paralelizar aplicaciones en memoria compartida de manera sencilla. Permite dividir bucles entre múltiples hilos ejecutados en paralelo. Ideal para computadoras con múltiples núcleos.

Ventajas:

* Facilidad de implementación.
* Escasa sobrecarga de comunicación.

##### 2.3. Paralelismo con MPI

**MPI (Message Passing Interface)** es un estándar para la programación paralela en memoria distribuida. Permite que múltiples procesos (posiblemente en distintas máquinas) cooperen intercambiando mensajes.

Ventajas:

* Escalabilidad en clusters.
* Flexibilidad en la comunicación entre nodos.

## Descripción del Código

### 3.1. Prototipos de Funciones

Los siguientes son los prototipos de las funciones que utilizamos:

### 3.2. Versión con OpenMP

El enfoque con OpenMP aprovecha la arquitectura de memoria compartida para dividir el trabajo entre múltiples hilos dentro de un mismo proceso. Cada hilo se encarga de calcular una o más filas del resultado de forma concurrente, accediendo a la misma matriz y vector de entrada.

Pasos principales:

#pragma omp parallel for divide automáticamente el bucle externo entre los hilos disponibles.

Cada hilo calcula los elementos correspondientes del vector resultado, trabajando de forma independiente sobre filas distintas.

No es necesario combinar los resultados, ya que todos los hilos escriben directamente en posiciones distintas del vector resultado, evitando condiciones de carrera.

Métricas y evaluación OPENMP

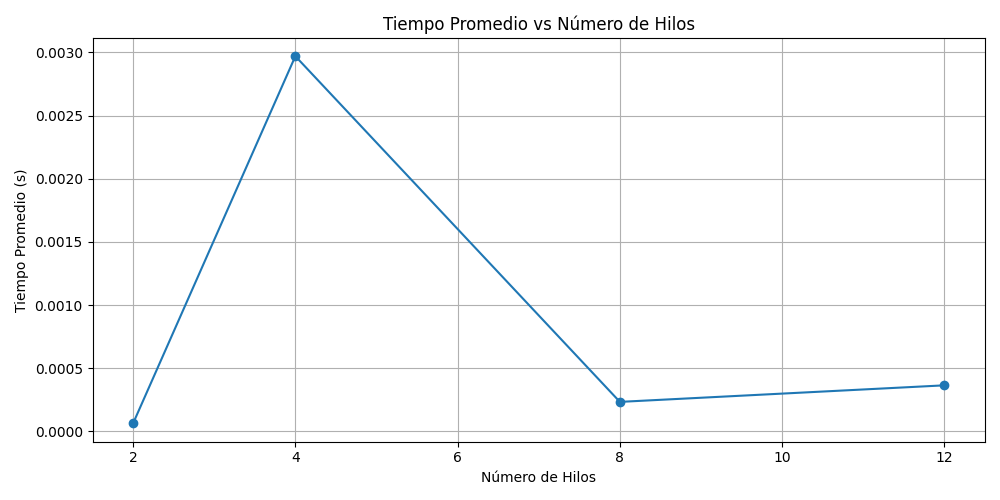
Para evaluar el rendimiento del algoritmo de multiplicación de matriz por vector usando OpenMP, se realizaron 5 mediciones para distintos niveles de paralelismo (2, 4, 8 y 12 hilos). Luego descartamos aquella medida que era mínima y aquella que era máxima, con las 3 restantes calculamos un promedio. Obtenemos un tiempo y lo utilizamos para calcular las métricas. También se realizaron 4 escenarios distintos, donde varía el tamaño de la matriz, usamos matrices de 10x10, 100x100, 1000x1000 y 10000x10000

Primer Escenario a analizar: Matriz 10x10

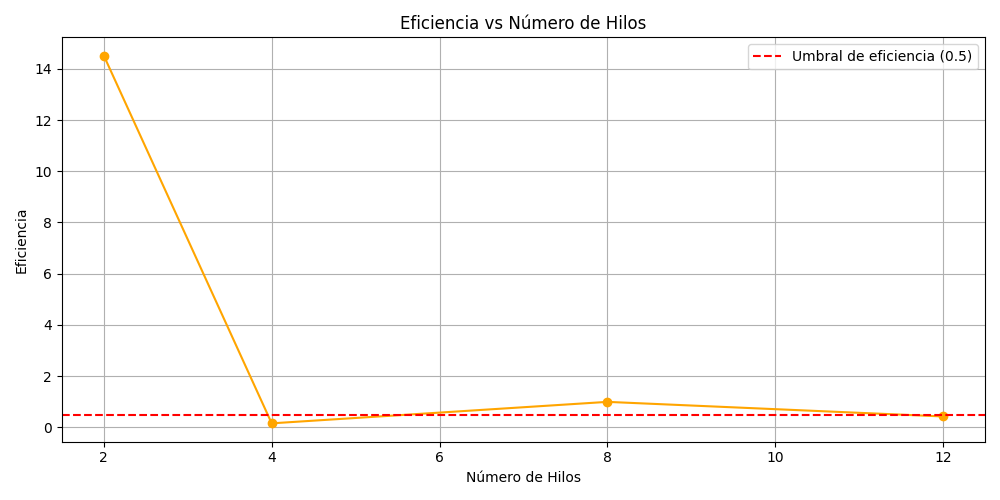
Tiempo Secuencial: 0.00186543

| Procesadores | 2 | 4 | 8 | 12 |  |
| --- | --- | --- | --- | --- | --- |
| Tiempo | 0.0000642807 | 0.002969600 | 0.000234275 | 0.000365194 |  |
| Speed up | 29.0201304695 | 0.628176564 | 7.962568260 | 5.108061286 |  |
| Eficiencia | 14.5100652348 | 0.157044141 | 0.995321033 | 0.425671774 |  |

La configuración con 8 hilos presenta una eficiencia cercana a 1, lo que indica un excelente aprovechamiento de los recursos disponibles. Por otro lado, la eficiencia observada con 2 hilos es bastante alta, lo cual podría estar influenciado por el bajo tamaño del problema y la menor sobrecarga de sincronización, factores que en ciertos contextos pueden favorecer configuraciones con menor paralelismo. Sin embargo, considerando la escalabilidad se identifica a 8 hilos como la configuración más adecuada.





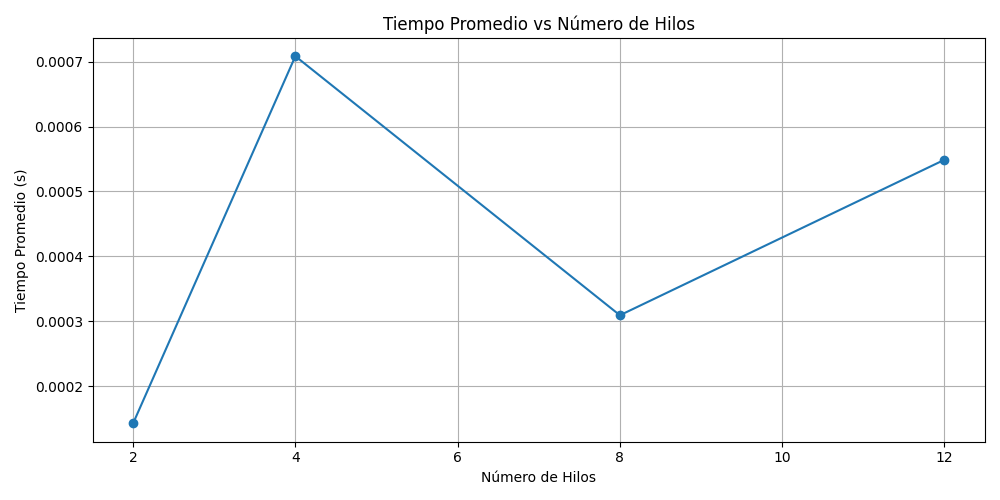


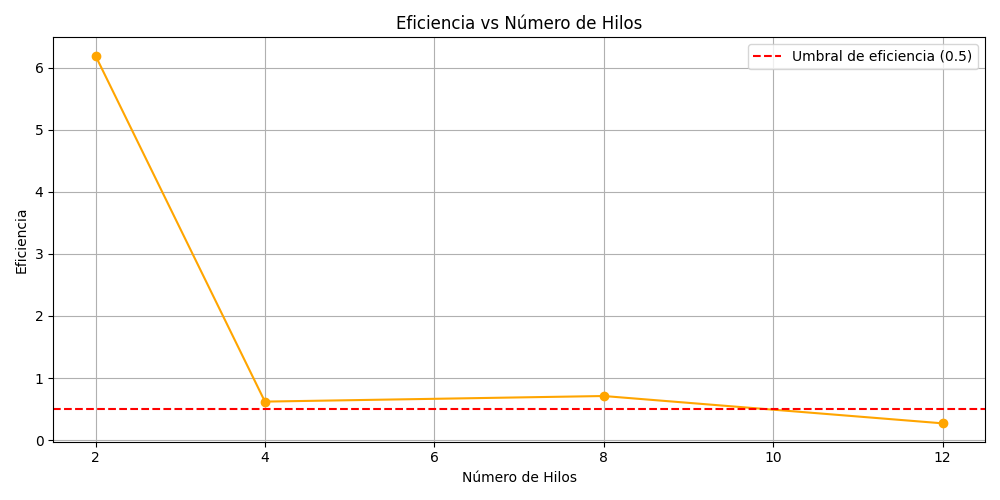
Segundo Escenario a analizar: Matrix 100x100

Tiempo Secuencial: 0.00176853

| Procesadores | 2 | 4 | 8 | 12 |
| --- | --- | --- | --- | --- |
| Tiempo | 0.0001427450 | 0.000708319 | 0.000309257 | 0.000548762 |
| Speed up | 12.3894590587 | 2.496804640 | 5.718658719 | 3.222767355 |
| Eficiencia | 6.1947295293 | 0.624201160 | 0.714832340 | 0.268563946 |

Tanto 4 como 8 hilos superan el umbral de 0.5, siendo opciones válidas. Sin embargo, 4 hilos es la opción más cercana a 0.5 y por ende más equilibrada en cuanto a uso de recursos y rendimiento.



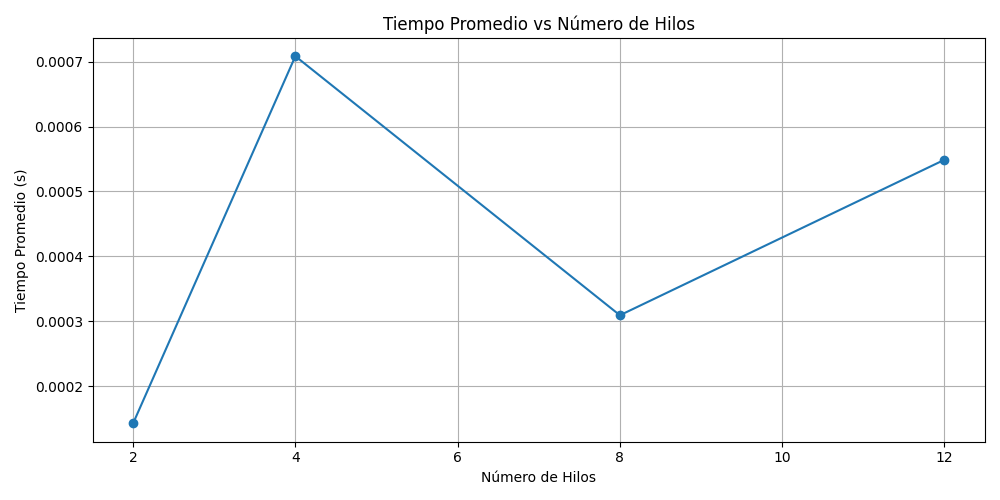


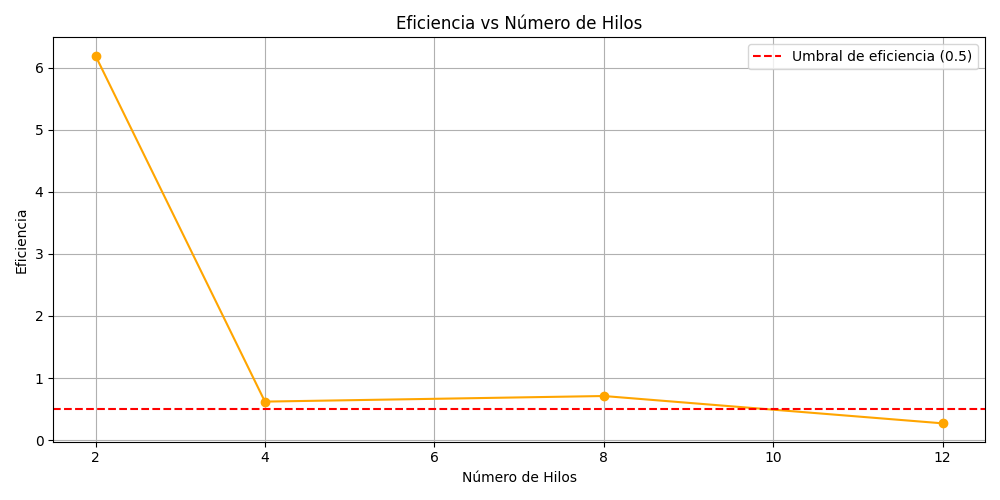
Tercer Escenario a Analizar: Matriz 1000x1000

Tiempo Secuencial: 0.00180633

| Procesadores | 2 | 4 | 8 | 12 |  |
| --- | --- | --- | --- | --- | --- |
| Tiempo | 0.0030745403 | 0.004195787 | 0.003610479 | 0.003850382 |  |
| Speed up | 0.5875132987 | 0.430511175 | 0.500302915 | 0.469130901 |  |
| Eficiencia | 0.2937566494 | 0.107627794 | 0.062537864 | 0.039094242 |  |

En este escenario ninguna configuración supera el umbral de eficiencia 0.5, lo que indica una falta de escalabilidad efectiva con OpenMP. Esto puede deberse a cuellos de botella o sobrecarga de sincronización por la complejidad del problema.





Cuarto Escenario a Analizar: Matriz 10000x10000

En este caso no se pudo obtener matriz debido a que al momento de ejecutar el código nos devolvió “Segmentation fault (core dumped)”, esto ocurre porque se intenta acceder a una zona de memoria a la que no tiene permiso o que no está asignada. Lo que lleva a desbordamientos de pila, errores de lectura o escritura en memoria, o el uso incorrecto de punteros. Para probar este escenario necesitamos un equipo con más recursos.

### 3.3 Versión con MPI

El enfoque con MPI divide las filas de la matriz entre los procesos, cada uno calcula una porción del resultado y luego se combinan con MPI\_Gather.

Pasos principales:

* MPI\_Scatter reparte filas de la matriz.
* Cada proceso calcula una parte del vector resultado.
* MPI\_Gather reúne los resultados en el proceso maestro.

Matrices y evaluación. MPI

Para evaluar el rendimiento del algoritmo de multiplicación de matriz por vector usando MPI, se realizaron 5 mediciones para distintos niveles de paralelismo (2, 4, 6 y 7 nodos). Luego descartamos aquella medida que era mínima y aquella que era máxima, con las 3 restantes calculamos un promedio. Obtenemos un tiempo y lo utilizamos para calcular las métricas. También se realizaron 4 escenarios distintos, donde varía el tamaño de la matriz, usamos matrices de 10x10, 100x100, 1000x1000 y 10000x10000

Configuración que se llevó a cabo:

En cada slave modificamos el archivo fstab

Y agregamos la linea

Y ejecutamos

Después en el master

Primer Escenario a analizar: Matriz 10x10

Tiempo Secuencial:0.00001333

| Nodos | 2 | 4 | 6 | 7 |
| --- | --- | --- | --- | --- |
| Tiempo | 0.0000993333 | 0.000258333 | 0.000199667 | 0.000187667 |
| Speed up | 0.1342281879 | 0.051612903 | 0.066777963 | 0.071047957 |
| Eficiencia | 0.0671140940 | 0.012903226 | 0.011129661 | 0.010149708 |

Segundo Escenario a analizar: Matriz 100x100

Tiempo Secuencial: 0.00013600

| Nodos | 2 | 4 | 6 | 7 |
| --- | --- | --- | --- | --- |
| Tiempo | 0.0001963333 | 0.000754667 | 0.000221667 | 0.000237667 |
| Speed up | 0.6926994907 | 0.180212014 | 0.613533835 | 0.572230014 |
| Eficiencia | 0.3463497453 | 0.045053004 | 0.102255639 | 0.081747145 |

Tercer Escenario a analizar: Matriz 1000x1000

Tiempo Secuencial: 0.00315467

| Nodos | 2 | 4 | 6 | 7 |
| --- | --- | --- | --- | --- |
| Tiempo | 0.0014820000 | 0.001625667 | 0.001344333 | 0.001215667 |
| Speed up | 2.1286549708 | 1.940537216 | 2.346640218 | 2.595009597 |
| Eficiencia | 1.0643274854 | 0.485134304 | 0.391106703 | 0.370715657 |

Cuarto Escenario a analizar: Matriz 10000x10000

Tiempo Secuencial: 0.28555367

| Nodos | 2 | 4 | 6 | 7 |
| --- | --- | --- | --- | --- |
| Tiempo | 0.1430100000 | 0.077319667 | 0.051032333 | 0.042844000 |
| Speed up | 1.9967391558 | 3.693156980 | 5.595544002 | 6.664962811 |
| Eficiencia | 0.9983695779 | 0.923289245 | 0.932590667 | 0.952137544 |

Conclusiones

Escenario 1: Matriz 10x10

Escenario 2: Matriz 100x100

Escenario 3: Matriz 1000x1000

En este escenario ninguna configuración supera el umbral de eficiencia 0.5, lo que indica una falta de escalabilidad efectiva con OpenMP. Esto puede deberse a cuellos de botella o sobrecarga de sincronización por la complejidad del problema.

Escenario 4: Matriz 10000x10000

En este caso no se pudo obtener matriz debido a que al momento de ejecutar el código nos devolvió “Segmentation fault (core dumped)”, esto ocurre porque se intenta acceder a una zona de memoria a la que no tiene permiso o que no está asignada. Lo que lleva a desbordamientos de pila, errores de lectura o escritura en memoria, o el uso incorrecto de punteros. Para probar este escenario necesitamos un equipo con más recursos.