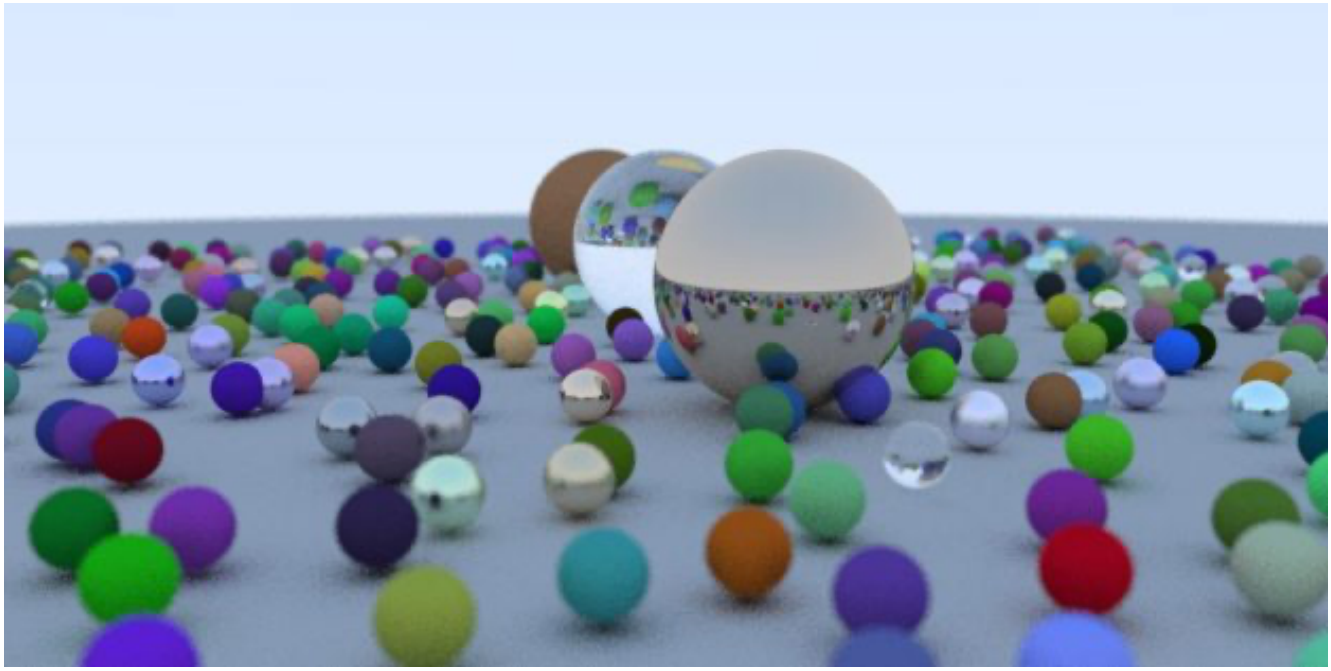
Universidad Rey Juan Carlos

Escuela Técnica Superior de Ingeniería Informática

GRADO EN INGENIERIA DE COMPUTADORES

Curso académico 2024/2025

RENDERIZADO DE PELÍCULAS CON RAYTRACING UTILIZANDO MPI, OPENMP Y CUDA



Santiago Nicolás Díaz Tituaña
Grupo W – Práctica 2

Índice

1. Introducción

El objetivo de esta práctica es realizar un análisis detallado del rendimiento en el renderizado de imágenes mediante ray tracing (trazado de rayos), utilizando diferentes estrategias de paralelización con MPI, OpenMP y CUDA. El objetivo es aplicar estas tecnologías para mejorar los tiempos de ejecución del motor de ray tracing base (originalmente secuencial), y comparar tanto dentro de una misma tecnología como entre tecnologías distintas, incluyendo posibles implementaciones híbridas.

En la fase de implementación de la paralelización, se abordarán distintas estrategias dentro de cada tecnología. Estas incluyen la paralelización de distintos fotogramas (especialmente útil para animaciones) y, dentro de un único fotograma, las siguientes estrategias:

- Procesamiento independiente de filas,
- Procesamiento independiente de columnas,
- Procesamiento independiente de porciones rectangulares(chunks)

En el caso de CUDA, se explorarán además diferentes configuraciones de bloques para analizar su impacto en el rendimiento.

Para la experimentación y recogida de datos, se variarán parámetros clave como el tamaño de imagen, el número de muestras por píxel (samples) y la cantidad de procesos, hilos o bloques utilizados, según la tecnología. Se registrarán los tiempos de ejecución para cada configuración y se realizarán comparativas tanto entre estrategias dentro de la misma tecnología como entre tecnologías diferentes, incluyendo aquellas combinadas.

Finalmente, se hará un análisis de los resultados para ver cómo escala cada implementación al aumentar la resolución, el número de muestras por píxel o el número de hilos/procesos. También se estudiarán los factores que más influyen en el rendimiento y se extraerán conclusiones claras sobre qué tecnología (o combinación de ellas) se adapta mejor a cada tipo de paralelización. Además, se tendrá en cuenta el equilibrio entre la dificultad de implementación de dicha estrategia y su impacto sobre otros más sencillos.

Es importante señalar que la implementación detallada del código no se incluye en este documento, ya que la memoria está enfocada en la recogida de datos, el análisis experimental y las conclusiones. El código fuente se adjunta como anexo.

2. Implementación de la paralelización

Las estrategias de paralelización utilizadas en esta práctica parten de conceptos comunes, pero su implementación varía notablemente según la tecnología empleada. En esta sección se describe cómo se ha aplicado cada una en OpenMP, MPI y CUDA, sin entrar en detalles específicos del código, pero destacando las particularidades relevantes de cada enfoque.

2.1 Estrategia de paralelización con OpenMP

2.1.1 Paralelización para varios frames

En esta estrategia, se generan un conjunto de imágenes independientes(frames) para procesarse en paralelo. Cada hilo lanzado mediante OpenMP se encarga de su renderizado, incluyendo la escritura final de la imagen. La limitación de la implementación viene dada por el número de hilos disponibles para lanzar pues concuerda con el número de imágenes renderizadas.

2.1.2 Paralelización por filas

En esta implementación, cada hilo se encarga de procesar un conjunto de filas completas. Cada hilo recorre las columnas de una o varias filas , es decir la paralelización se aplica en el bucle externo del recorrido de la imagen

2.1.4 Paralelización por columnas

En este caso, cada hilo se encarga de procesar columnas de una imagen , a diferencia de la paralización por filas , aquí el bucle interno -el de columnas- es el que se paraleliza, distribuyéndolas entre los hilos de ejecución.

2.1.4 Paralelización por bloques (chunks)

En esta estrategia, la imagen se divide en bloques rectangulares asignados a cada hilo. Se implementaron dos variantes:

- **Estática**: cada hilo recorre una región fija de la imagen —en esta implementación, de 128x128 píxeles— que se mantiene durante todo el proceso. Se aplica el principio de granularidad gruesa, explicado en los siguientes apartados tras el análisis.
- **Dinámica**: se asignan bloques de trabajo conforme los hilos van terminando(granularidad fina). Esto permite un mejor balanceo de carga y evita hilos inactivos, aunque introduce una sobrecarga de coordinación, sobre todo cuando la imagen es muy sencilla, pues dicha sincronización no es despreciable para cálculos simples y puede llevar a tener peores tiempos que estrategias secuenciales.

2.2 Estrategia de paralelización con MPI

A diferencia de OpenMP, MPI no utiliza memoria compartida, sino paso de mensajes, por lo que se adaptó el esquema para minimizar las comunicaciones entre el proceso que escribe la imagen y los que renderizan (incluido el propio proceso 0). Se probó un esquema maestro-esclavo, donde el proceso 0 solo coordinaba y escribía, pero al eliminarlo del cálculo, los tiempos empeoraban. Por eso, se optó por un enfoque en el que todos los procesos, incluido el 0, renderizan parte de la imagen. Aunque esto implica que el ensamblado final se retrasa hasta que el proceso 0 termina su parte, el resultado fue mejor en rendimiento que con el esquema anterior.

2.2.1 Paralelización para varios frames

Para la paralelización por frames en MPI, cada uno de los procesos renderiza una imagen de manera individual, limitado como máximo a N imágenes si existen N procesos. Se pensó en fases iniciales que fuera el proceso 0 quien escribiera todas las imágenes, pero no resultó eficiente, ya que el proceso 0 tendría que reservar una gran cantidad de memoria para recoger todos los resultados y luego escribirlos. De todas formas, en este esquema no existe comunicación directa entre los procesos, ya que no la necesitan: cada uno sabe qué imagen debe renderizar y guardarla.

2.2.2 Paralelización por filas

Para la paralelización por filas, se optó por que cada proceso renderizara varias filas de forma local en lugar de enviar una por una, ya que esto resultaba ineficiente. Al terminar, los resultados se envían en bloque al proceso 0, que los recopila y escribe la imagen. Así se mantiene la estrategia por filas, pero con menos comunicaciones y mejor rendimiento.

2.2.3 Paralelización por columnas

En el caso de la paralelización por columnas, se siguió un enfoque similar al de las filas: cada proceso renderiza un bloque de columnas de forma local, y luego envía el resultado al proceso 0, que se encarga de ensamblar la imagen. Al igual que con las filas, se agrupan varias columnas por proceso para reducir el número de comunicaciones y mejorar la eficiencia.

2.2.4 Paralelización por bloques (chunks)

Para la paralelización por bloques se probaron dos estrategias distintas: una estática y otra dinámica. En la versión estática, el grid de la imagen se divide desde el principio entre todos los procesos, repartiéndose bloques grandes de forma fija (granularidad gruesa). En la versión dinámica, a cada proceso se le asignan bloques de tamaño fijo, siguiendo un orden tipo round robin, sin necesidad de que los procesos los soliciten activamente. Ambas versiones se incluirán en las comparativas.

2.3 Estrategia de paralelización con CUDA

2.3.1 Paralelización para varios frames

Para la estrategia de paralelización por fotogramas se utilizó un grid tridimensional, donde cada capa del eje Z representa un frame distinto de la animación. De esta forma, se pueden procesar varios fotogramas en paralelo, y cada hilo se encarga de un píxel específico dentro de su respectivo frame. Esta organización permite lanzar múltiples imágenes (una animación)

2.3.2 Paralelización por filas

En la estrategia por filas se dividió cada fila en varios bloques para adaptarse a las limitaciones del número máximo de hilos por bloque en CUDA. Se configuró un grid 2D donde cada fila corresponde a un valor del eje Y, y las columnas se reparten entre bloques en el eje X. De este modo, cada hilo procesa un píxel concreto dentro de su fila, permitiendo renderizar imágenes grandes y manteniendo la lógica de paralelización por filas con un buen acceso a memoria global.

2.3.3 Paralelización por columnas

En la estrategia por columnas se siguió un enfoque similar al de filas, pero invirtiendo el reparto del trabajo: cada bloque se asigna a una columna, y los hilos dentro del bloque procesan píxeles de esa columna, uno por cada fila. Así se mantiene la lógica de recorrer la imagen verticalmente, distribuyendo el trabajo de forma equilibrada y adaptándolo a las restricciones del modelo de ejecución de CUDA.

2.3.4 Paralelización por bloques (chunks)

La estrategia de paralelización por bloques no fue implementada manualmente, ya que venía incluida en el código base de la práctica. Esta versión utiliza una configuración de bloques de 16×16 hilos, dividiendo la imagen en cuadrículas pequeñas donde cada hilo procesa un píxel. Esta forma de paralelización corresponde a un esquema bidimensional.

2.3 Estrategia de paralelización híbrida

2.3.1 MPI + OpenMP (distribución de frames)

En esta estrategia híbrida se ha combinado MPI y OpenMP con el objetivo de poder aprovechar al máximo los recursos tanto a nivel de hilos como de procesos. En concreto, se ha utilizado MPI para distribuir la carga de trabajo entre los procesos. De la misma forma que en la implementación normal, cada proceso se encarga de la generación completa de un frame distinto. A su vez, cada uno de los procesos emplea OpenMP para poder paralelizar internamente la creación del frame, dividiendo el trabajo entre los hilos asignados a ese proceso.

3. Experimentación y recogida de datos

3.1 Entorno de pruebas

El desarrollo se hizo principalmente en un ordenador de torre sin tarjeta gráfica, así que se trabajó sobre Linux. Pero para poder ejecutar también la parte de CUDA, se usó un portátil ASUS TUF Gaming con una NVIDIA RTX 3050, cuya compute capability es 8.6, algo importante ya que afecta al lanzamiento de bloques y a la distribución de hilos en los kernels.

Para facilitar la compilación y no tener que adaptar el entorno, se utilizó **WSL** en el portátil. Así se pudo seguir trabajando con Linux y probar todas las tecnologías en la misma máquina, lo cual viene bien para comparar resultados sin que el hardware o el sistema operativo afecten a las pruebas.

3.2 Experimentos

Para realizar los experimentos se utilizará la función randomScene() dentro de la lógica de procesamiento de escena. A diferencia de la escena inicial, que solo genera 4 esferas, esta crea muchas más, lo que incrementa notablemente la complejidad del procesamiento y exige más a nivel computacional o gráfico. En el apartado de “Escalabilidad” se analizará cómo este cambio, de una escena simple a una más compleja, afecta al rendimiento de cada estrategia, sobre todo en el caso de OpenMP.

3.2.1 Comparativa de estrategias para una tecnología

Para las comparativas siguientes se utilizará una resolución fija de 1200x800 y 50 muestras por píxel (ns = 50). En OpenMP se emplearán 6 hilos, en MPI se utilizarán 6 procesos, y en el caso de CUDA no se configura explícitamente el número de hilos como tal, sino que se define a través de la configuración de lanzamiento del kernel, según se detalló en el apartado de implementación.

3.2.1.1 OpenMP

Estrategia	Tiempo 1 (s)	Tiempo 2(s)	Tiempo 3(s)	Tiempo 4 (s)	Tiempo medio
Filas	123,741	122,514	122,883	124,880	123.504
Columnas	136,008	134,143	133,655	134,170	134.494
Bloques(estáticos)	124,928	123,945	125,129	123,611	124.403
Bloques(dinámicos)	123,509	123,500	123,493	123,403	123.476

3.2.1.2 MPI

Estrategia	Tiempo 1 (s)	Tiempo 2(s)	Tiempo 3(s)	Tiempo 4 (s)	Tiempo medio
Filas	60.286	60.638	61.153	61.272	60.837
Columnas	54.515	55.086	55.822	55.999	55.355
Bloques(estáticos)	61.717	62.231	62.318	62.312	62.144
Bloques(dinámicos)	52.235	52,345	52.500	52.690	52.442

3.2.1.3 CUDA

Estrategia	Tiempo 1 (s)	Tiempo 2(s)	Tiempo 3(s)	Tiempo 4 (s)	Tiempo medio
Filas	10.998	11.072	10,977	11,046	11.023
Columnas	13.505	13.459	13.459	13.549	13.493
Bloques	10.007	9.938	9.944	9.993	9.970

3.2.2 Comparativa de tecnologías para una misma estrategia

Aunque previamente se han comparado para cada tecnología las distintas estrategias, en este apartado no se reutilizarán esos datos. Se opta por realizar nuevas pruebas, ya que aumentar el número de mediciones facilita un análisis más general, permite descartar posibles valores atípicos y favorece la visualización de los resultados en una única tabla. De esta forma, es más sencillo identificar qué tecnología resulta más eficiente bajo una misma estrategia y analizarlo en detalle más adelante

3.2.2.1 Paralelización de frames

Para comparar la generación de frames (animación), se limitará el experimento a seis fotogramas, ya que es el máximo número de procesos que se puede lanzar con MPI. De este modo, se asegura una comparación justa

entre tecnologías. La resolución de las imágenes se mantendrá constante, aunque se reducirá el número de muestras por píxel (ns = 10).

Tecnología	Tiempo 1 (s)	Tiempo 2 (s)	Tiempo 3 (s)	Tiempo 4 (s)	Tiempo medio
OpenMP	148.233	149.343	148.586	151.996	149.539
MPI	57.143	57.705	58.029	57.454	57.582
CUDA	12.068	12.098	12,094	12,137	12.099

3.2.2.2 Paralelización por filas

Tecnología	Tiempo 1 (s)	Tiempo 2 (s)	Tiempo 3 (s)	Tiempo 4 (s)	Tiempo medio
OpenMP	123.913	121.839	123.976	122.496	123.056
MPI	60.711	61.263	60.609	60.489	60.768
CUDA	11.037	10.936	11.006	10.954	10.983

3.2.2.3 Paralelización por columnas

Tecnología	Tiempo 1 (s)	Tiempo 2 (s)	Tiempo 3 (s)	Tiempo 4 (s)	Tiempo medio
OpenMP	132.678	134.110	133.452	136.187	134.106
MPI	55.728	55.261	55.263	55.330	55.395
CUDA	13.646	13.470	13,670	13.6293	13.603

3.2.2.4 Paralelización por bloques

Para esta comparativa, la **ausencia de CUDA** se debe a que, en OpenMP y MPI, la división por bloques responde a una cuestión de reparto del trabajo: cada hilo o proceso se encarga de procesar bloques de la imagen, ya sea de forma estática o dinámica. En cambio, en CUDA los bloques se organizan como parte del modelo de ejecución: cada bloque contiene un conjunto de hilos, y cada hilo suele encargarse de un único píxel. Es decir, su propósito está más relacionado con el acceso eficiente a memoria (coalescencia) y con limitaciones del hardware, no con la distribución del trabajo como tal.

1)Bloques dinámicos

Tecnología	Tiempo 1 (s)	Tiempo 2 (s)	Tiempo 3 (s)	Tiempo 4 (s)	Tiempo medio
OpenMP	123.503	123.456	123.446	123.441	123.4615
MPI	52.503	52.381	52.409	52.108	52.350

2)Bloques estáticos

Tecnología	Tiempo 1 (s)	Tiempo 2 (s)	Tiempo 3 (s)	Tiempo 4 (s)	Tiempo medio
OpenMP	123.512	123.560	125.713	123.416	124.050
MPI	62.374	62.529	62.318	61.957	62.294

3.2.3 Comparativa de implementaciones hibridas

Para comparar **MPI + OpenMP**, se hará con implementaciones por frames frente a otras tecnologías, y se evaluará cuánto mejora respecto a la versión individual de MPI u OpenMP. También se incluirá CUDA como

referencia, aunque sus tiempos son menores y se basan en un enfoque distinto, lo cual permite obtener una visión global más completa.

Se realizarán pruebas con escenas de alta complejidad (pruebas costosas) y también con escenas más sencillas, para comprobar si la mejora de la hibridación se mantiene en distintos contextos. En todos los casos, estas comparaciones se centrarán en el tiempo de ejecución total.

Dado que las implementaciones híbridas generan una cantidad fija de 3 frames, se intentó ampliar a 6, pero el **hyperthreading** podría afectar negativamente a los resultados. Para la carga pesada se harán con imágenes de 1920 x 1080 y 50 ns (T1) , mientras que para la más ligera con 1200x800 y 10ns (T2)

Tecnología	Configuración	T1 total(s)	T2 total (s)	Tiempo por frame (T1)	Tiempo por frame (T2)
OpenMP	3 hilos	743.890	70.004	247.963	23.335
MPI	3 procesos	573.237	53.925	191.079	17.975
CUDA	-	59.2276	6.127	19.742	2.042
MPI+OpenMP	3 procesos x 2 hilos	379.623	35.245	126.541	11.748

3.2.4 Comparativa de escalabilidad

Para analizar la escalabilidad, se utilizó una escena compleja con más esferas, ya que en escenas simples y resoluciones bajas (sobre todo en OpenMP), las estrategias paralelas resultaban más lentas que la versión secuencial. Esto se debe al overhead de creación y sincronización de hilos, que supera el beneficio del paralelismo. En cambio, al aumentar la carga computacional, ese coste se vuelve despreciable y la paralelización comienza a compensar. Se seleccionó una única estrategia representativa por tecnología: división por bloques para CUDA , división dinámica para OpenMP y MPI.

Para que las comparativas fueran coherentes, se experimentó con distintas resoluciones y número de muestras por píxel, variando a la vez el número de hilos (en OpenMP), procesos (en MPI) o configuraciones de bloques (en CUDA).
En el caso de CUDA, se partió de la configuración por defecto del código base (bloques de 16×16 hilos), y se añadieron dos variantes adicionales (8×8 y 32×8) con el objetivo de evaluar cuál de ellas favorece más la **coalescencia de memoria**, es decir, un acceso más eficiente a memoria global por parte de los hilos que conforman un mismo warp.

Escalabilidad en OpenMP: variación de hilos y resolución

Resolución/Samples	2 hilos	4 hilos	6 hilos
640x480 (ns = 25)	28.843	20.120	19.912
640x480 (ns = 50)	58.608	39.735	39.592
1200x800 (ns = 25)	94.440	63.958	61.778
1200x800 (ns = 50)	190.843	124.971	123.469
1920x1080 (ns = 50)	384.811	262.789	261.664

Escalabilidad en MPI: impacto del número de procesos y resolución

Resolución/Samples	2 procesos	4 procesos	6 procesos
640x480 (ns = 25)	22.961	12.286	8.868
640x480 (ns = 50)	44.688	23.222	17.271
1200x800 (ns = 25)	68.752	35.270	26.491
1200x800 (ns = 50)	135.817	69.644	52.624
1920x1080 (ns = 50)	291.454	147.957	108.374

Escalabilidad en CUDA: configuración de bloques frente a resolución

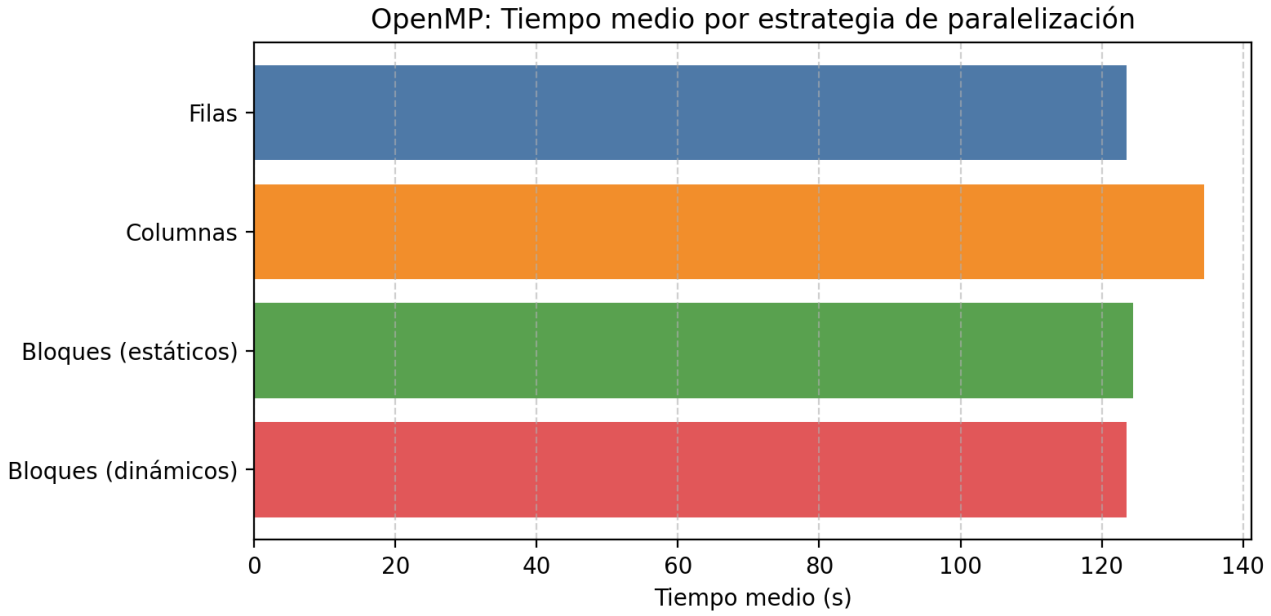
Resolución/Samples	Bloque de 8x8	Bloque de 16x16	Bloque de 32x8
640x480 (ns = 25)	1.717	1.940	1.966
640x480 (ns = 50)	3.298	3.683	3.819
1200x800 (ns = 25)	4.406	5.070	5.239
1200x800 (ns = 50)	8.606	9.935	10.255
1920x1080 (ns = 50)	17.089	19.963	20.422
1920x1080 (ns = 100)	34.109	39.511	40.756

4. Análisis de los resultados

Una vez recogidos todos los datos experimentales, se procede a representarlos gráficamente a partir de los tiempos medios. Esta visualización permite una comparación más clara y directa entre estrategias y tecnologías, facilitando el análisis y la extracción de conclusiones sin depender únicamente de los valores numéricos.

4.1 Comparativa de estrategias para una tecnología

4.1.1 OpenMP



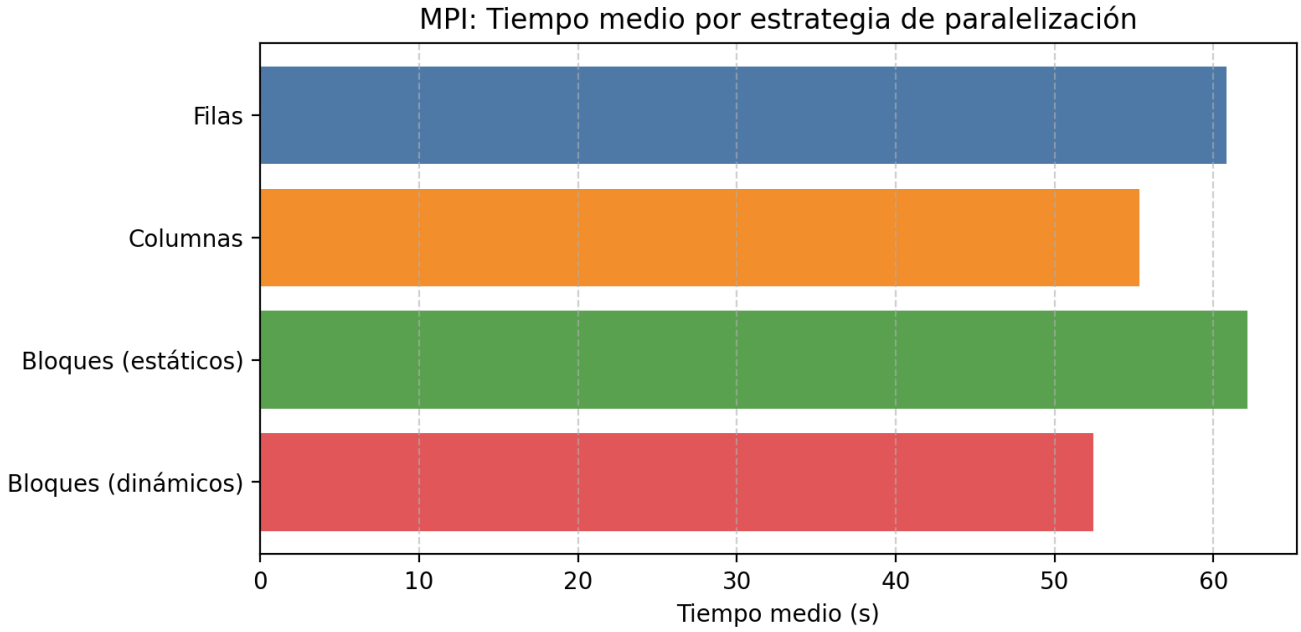
En el gráfico se observa que la estrategia de bloques dinámicos con OpenMP es la más eficiente, ya que presenta el menor tiempo medio de ejecución. Esto se debe a su capacidad para balancear mejor la carga de trabajo entre los hilos, asignando nuevas tareas a medida que estos finalizan, lo que da lugar a un reparto más equitativo y dinámico, especialmente en nuestro caso, cuando hay regiones costosas de renderizar.

Por otro lado, la estrategia menos eficiente fue la paralelización por columnas, probablemente porque el acceso a memoria no es óptimo, pues los hilos no procesan datos contiguos, lo que penaliza el rendimiento.

Las estrategias por filas y por bloques estáticos, en las que cada hilo recibe una carga fija de trabajo desde el inicio, tienen un rendimiento intermedio. En el caso de bloques estáticos, es probable que se aproveche de mejor forma la granularidad fina (dinámico) que la gruesa (estático).

En conclusión, los bloques dinámicos resultan ser la alternativa más adecuada entre todas las estrategias evaluadas, al adaptarse a la distribución de trabajo desigual en la imagen

4.1.2 MPI



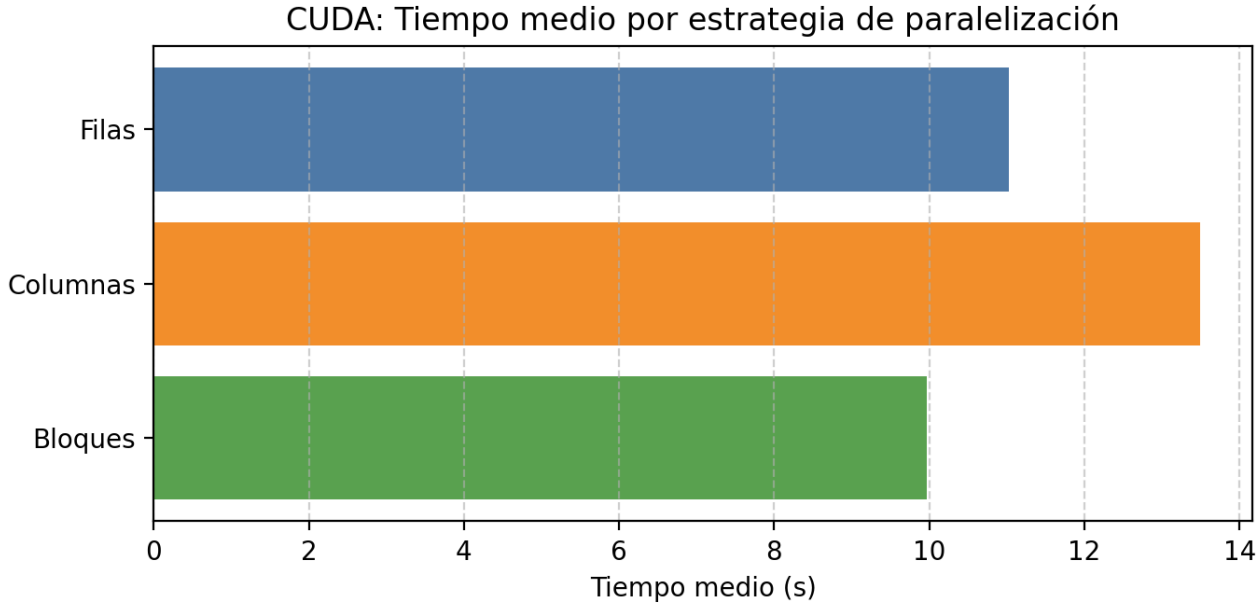
En el caso de la paralelización con MPI, la estrategia con mejor rendimiento —al igual que en OpenMP— fue la dinámica por bloques, principalmente, como vimos anteriormente, por el equilibrio de carga y el mejor aprovechamiento de los recursos en un entorno distribuido.

En contraste, la peor estrategia fue la de bloques estáticos. La granularidad gruesa no ayuda mucho cuando se escala a una mayor resolución, independientemente de la cantidad de procesos que estemos usando, pues la carga de trabajo, si no es equitativa (como en nuestro caso), puede provocar procesos ociosos que terminen tareas antes.

La comparativa intermedia —y sorprendente al analizarla— es la superioridad de columnas en entornos distribuidos. Lo más probable es que, en nuestro caso, el renderizado horizontal sea más homogéneo que hacerlo por filas, pues en pruebas se vio que hay filas que tardan mucho más en renderizarse que otras. De esta forma, se tendría que esperar a que terminen para unir, provocando largas esperas.

En conclusión, al igual que en OpenMP, los bloques dinámicos parecen ser la estrategia más adecuada para reducir el tiempo de ejecución con carga heterogénea.

4.1.3 CUDA



En la ejecución con CUDA, la estrategia de paralelización basada en bloques (de ejecución, no confundir con bloques de trabajo como en OpenMP) —aunque puedan parecer equivalentes, ya que en ambos casos se reparte el trabajo en bloques que se asignan a un hilo o proceso o se ejecutan dinámicamente—, en este caso los bloques contienen hilos que representan la disposición de cómo se va a procesar cada uno de los píxeles de la imagen.

Esta estrategia ofrece un tiempo de ejecución inferior al de las demás. A continuación, se sitúa la paralelización por filas y, en último lugar, la estrategia por columnas, que resultó ser la más lenta.

Esto se debe principalmente al acceso de los hilos a la memoria global. Los hilos se agrupan en unidades de 32 hilos (warp), y su rendimiento depende de que accedan a posiciones de memoria consecutivas y de cuánto se aprovechen las transacciones de lectura/escritura. En estrategias como filas y bloques, los hilos acceden a datos de memoria contigua, lo que permite la coalescencia, reduciendo así el número de operaciones necesarias.

Por otro lado, el acceso por columnas genera posiciones de memoria separadas (stride), lo que se nota mucho más a medida que aumenta la resolución, ya que el stride será mayor. Esto rompe la coalescencia y obliga a realizar más transacciones para llevar a cabo la misma tarea.

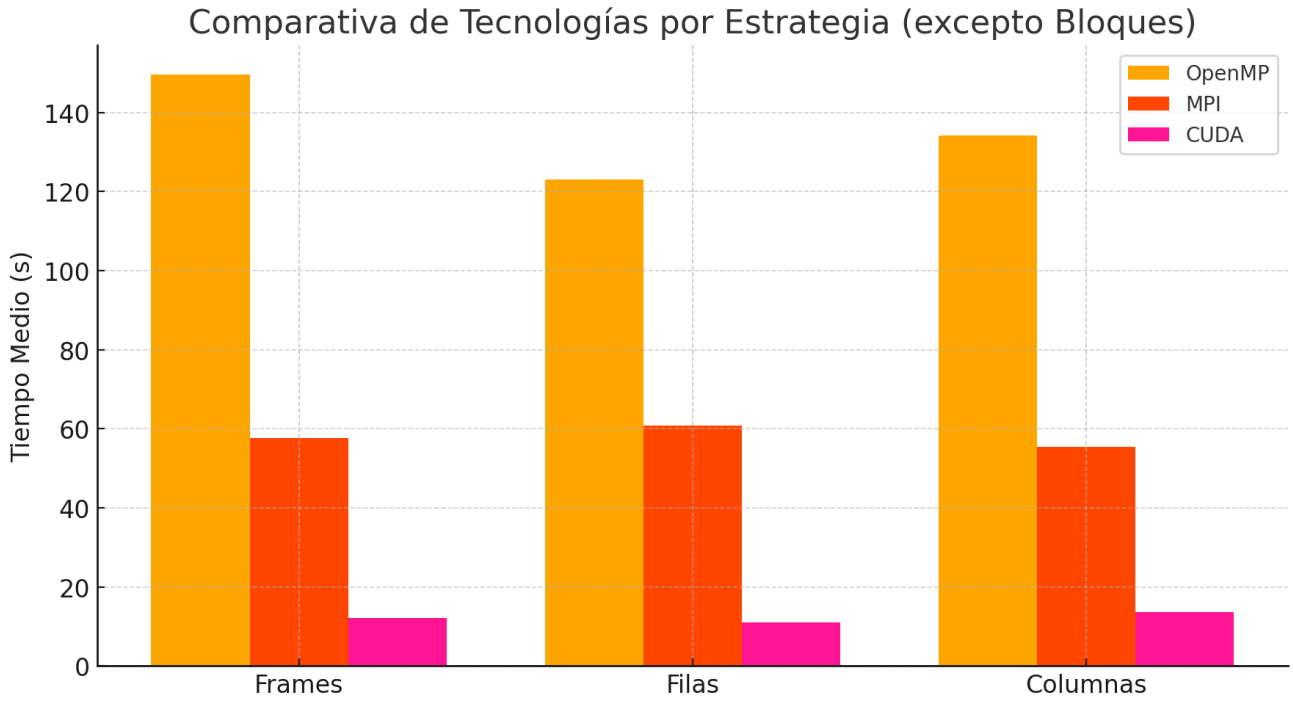
La estrategia por bloques es superior a la de filas debido a cómo se distribuyen los hilos dentro de la GPU, ya que permite repartir el trabajo de forma más equilibrada entre los SM (streaming multiprocessors).

4.2 Comparativa de tecnologías para una misma estrategia

Para comparar tecnologías para una misma estrategia de paralelización, se ha dividido en dos partes. En primer lugar, se analizarán las estrategias que son aplicables para todas las tecnologías.

En segundo lugar, se compararán las estrategias entre bloques solo entre OpenMP y MPI, excluyendo a CUDA, pues, como se explicó previamente, el concepto de bloques está relacionado al modelo de ejecución y no a la distribución de trabajo como en OpenMP y MPI. No sería una comparación justa ni equitativa.

Pero en ambos casos, nos permite saber cuál estrategia se adapta mejor para cada tecnología.



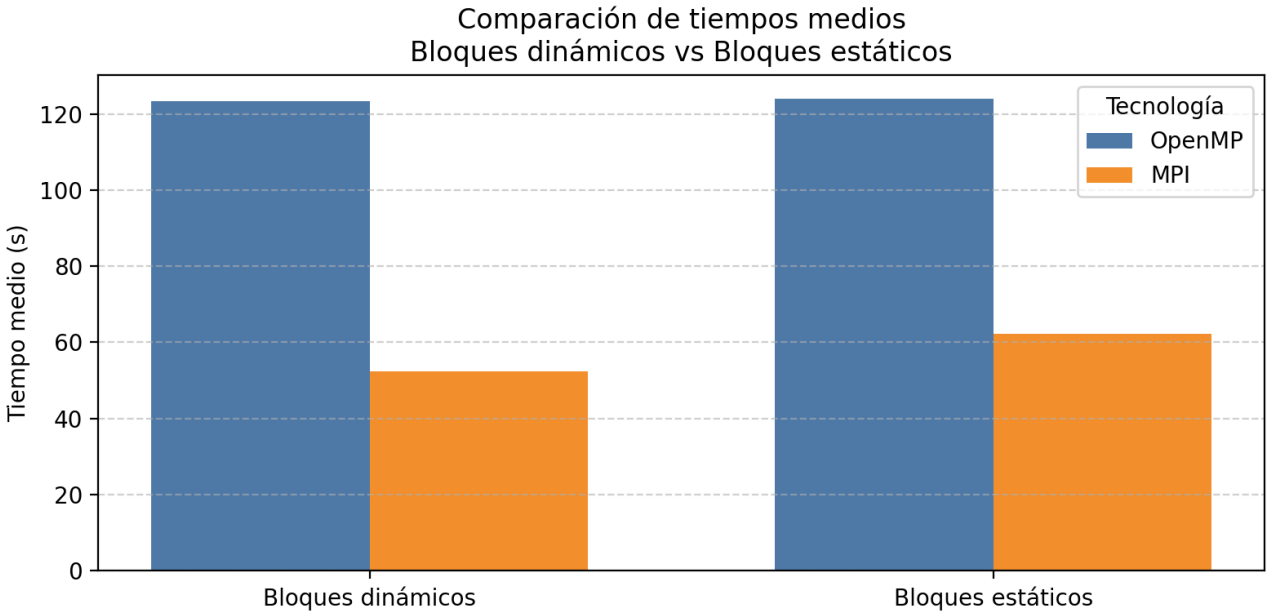
En esta gráfica, a partir de la comparación entre tecnologías para una misma estrategia de paralelización (frames, filas, columnas), se concluye que CUDA es la tecnología que ofrece mejor rendimiento en todos los casos (siempre que sean evaluables).

Se debe principalmente a la capacidad de la GPU para manejar miles de hilos en paralelo, con menor latencia en las operaciones, ya que casos como OpenMP dependen de sincronización y memoria compartida, y MPI de comunicación entre procesos, y obviamente están limitados por la CPU, por el número de hilos o procesos.

MPI, por su parte, es el que refleja un rendimiento intermedio, que supera notablemente a OpenMP por la capacidad que tiene para manejar múltiples procesos en varios nodos en entornos distribuidos, sin necesidad de sincronización entre nodos, más que para comunicación. Cabe recalcar que sigue siendo muy útil para tareas que requieren escalabilidad (explicado a continuación en la sección de escalabilidad).

Por otra parte, OpenMP tiene tiempos muy altos comparado a los dos anteriores, principalmente por la arquitectura de memoria compartida, el overhead y la sincronización de los hilos en un único nodo.

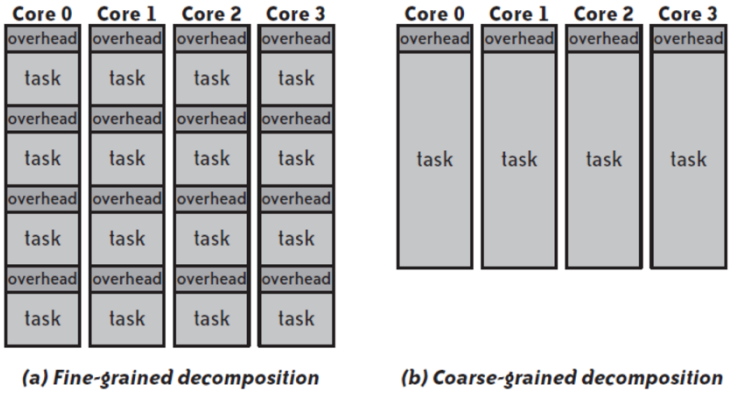
En resumen, CUDA destaca como la tecnología más eficiente, por su alta capacidad para procesar tareas; aunque sus hilos sean más sencillos que los de una CPU, son suficientes para manejar mejor la paralelización.



En cuanto a la comparativa por bloques, se ve claramente, como ya se explicó anteriormente, que MPI es superior al tener tiempos de ejecución menores que OpenMP, independientemente de la interpretación sobre la distribución de bloques.

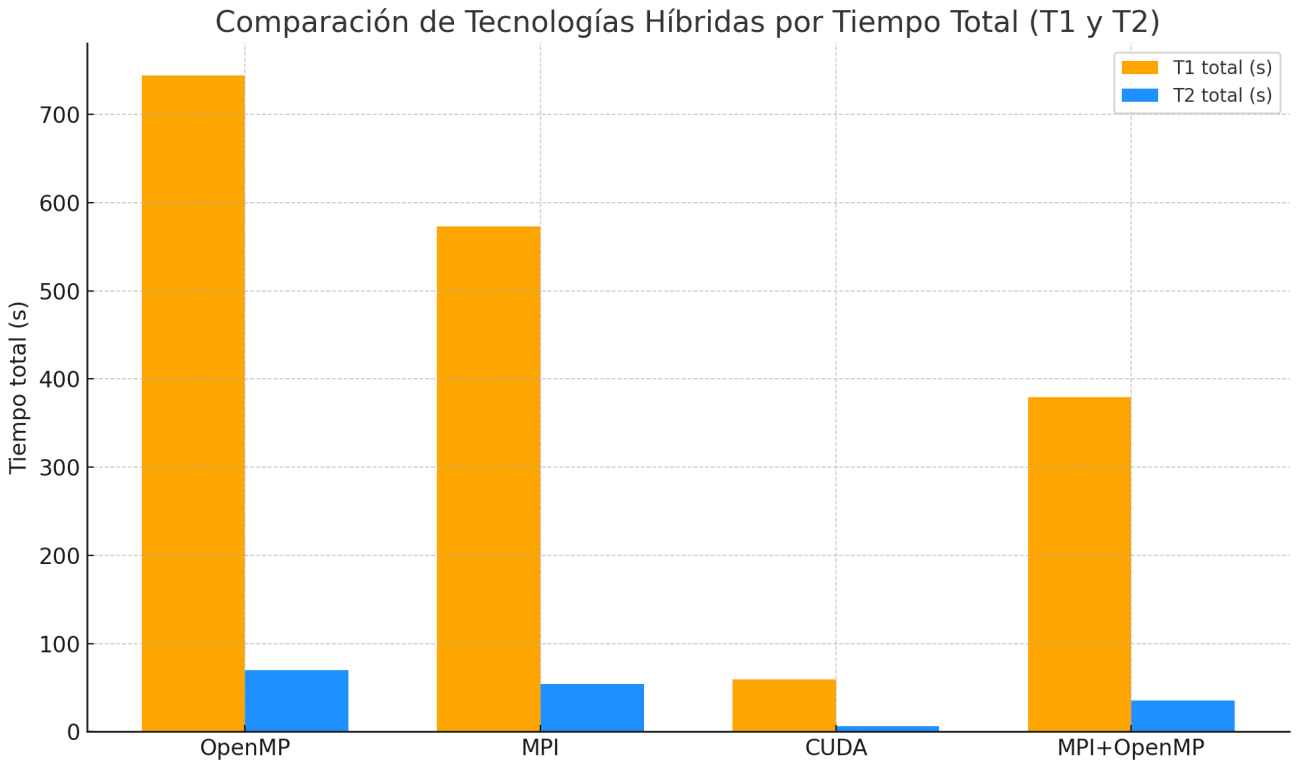
Lo más destacable es que los bloques dinámicos ofrecen menor tiempo de ejecución sin importar la tecnología utilizada. Esto se debe principalmente al enfoque adoptado: en el caso dinámico se aplica el principio de granularidad fina, es decir, el reparto de bloques es fijo, y mediante un algoritmo de asignación, cuando un hilo o proceso termina su bloque, se le asigna otro.

Esto favorece un reparto más homogéneo de tareas y evita que haya hilos o procesos ociosos, algo que sí ocurre en la asignación estática, donde cada hilo toma únicamente una tarea (bloque), pero de gran tamaño.



La imagen muestra esa diferencia: en la granularidad fina (izquierda), aunque hay más overhead, se aprovechan mejor los núcleos al repartir más tareas pequeñas; mientras que en la granularidad gruesa (derecha), aunque hay menos overhead, los núcleos pueden quedarse esperando si las tareas no están equilibradas como es nuestro caso , hay secciones más difíciles de renderizar que otras.

4.3 Comparativa para tecnologías hibridas



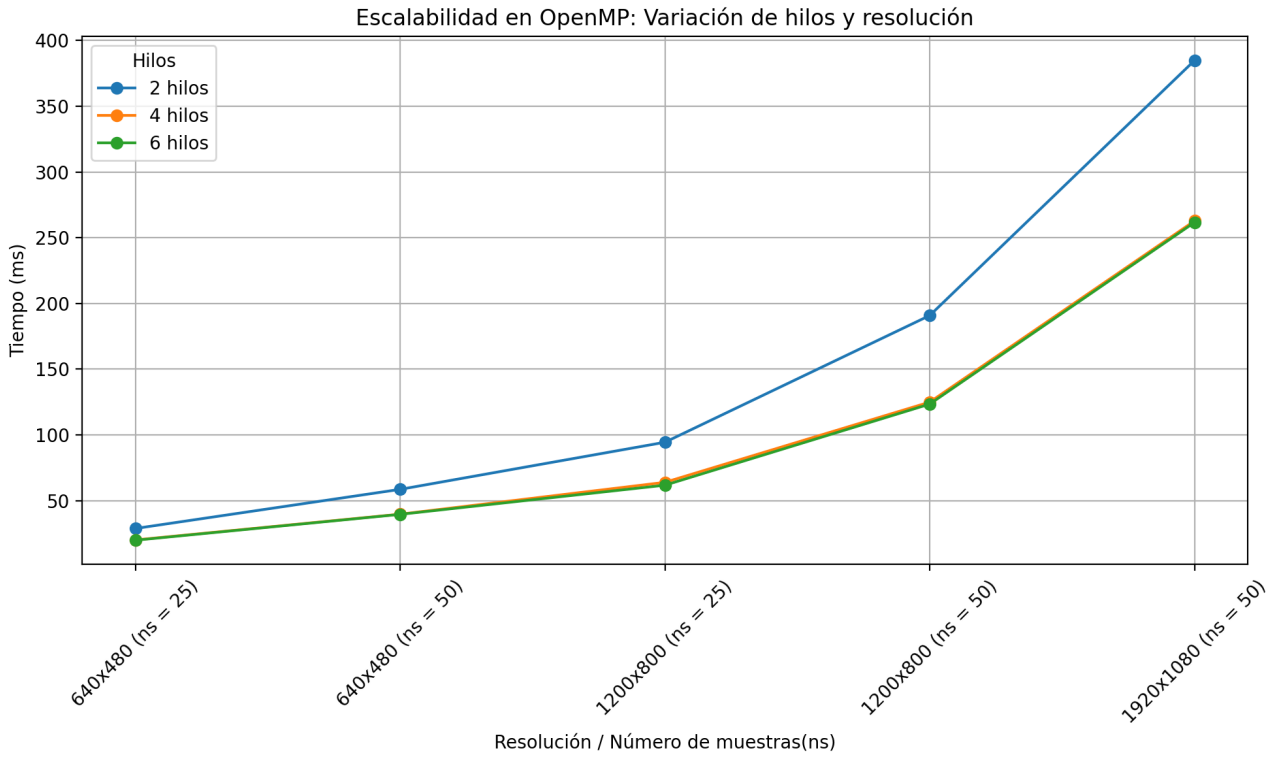
El propósito de esta comparativa es analizar en qué medida una estrategia híbrida mejora respecto a las implementaciones individuales, como son OpenMP y MPI por separado, tanto en condiciones de carga alta como ligera. Además, se incluye CUDA como referencia, al ser la opción más rápida para procesar este experimento.

Los resultados muestran que, tanto en carga alta como ligera, la combinación de OpenMP y MPI proporciona una mejora significativa respecto a la mejor versión de las implementaciones individuales (MPI). Supone una mejora de aproximadamente un 36 % respecto a MPI y casi del 50 % respecto a OpenMP, siendo esta la más lenta bajo carga alta. Esto demuestra que la implementación híbrida permite escalar correctamente, ya que al combinar hilos y procesos se asegura la utilización máxima de los recursos disponibles, reduciendo los tiempos de forma notable. Aun así, CUDA sigue destacando con diferencia por su alta capacidad de paralelización.

Para concluir, la tecnología híbrida entre MPI y OpenMP reduce significativamente los tiempos frente a sus versiones individuales, aunque sigue siendo inferior a CUDA.

4.4 Comparativa de escalabilidad

4.4.1 OpenMP

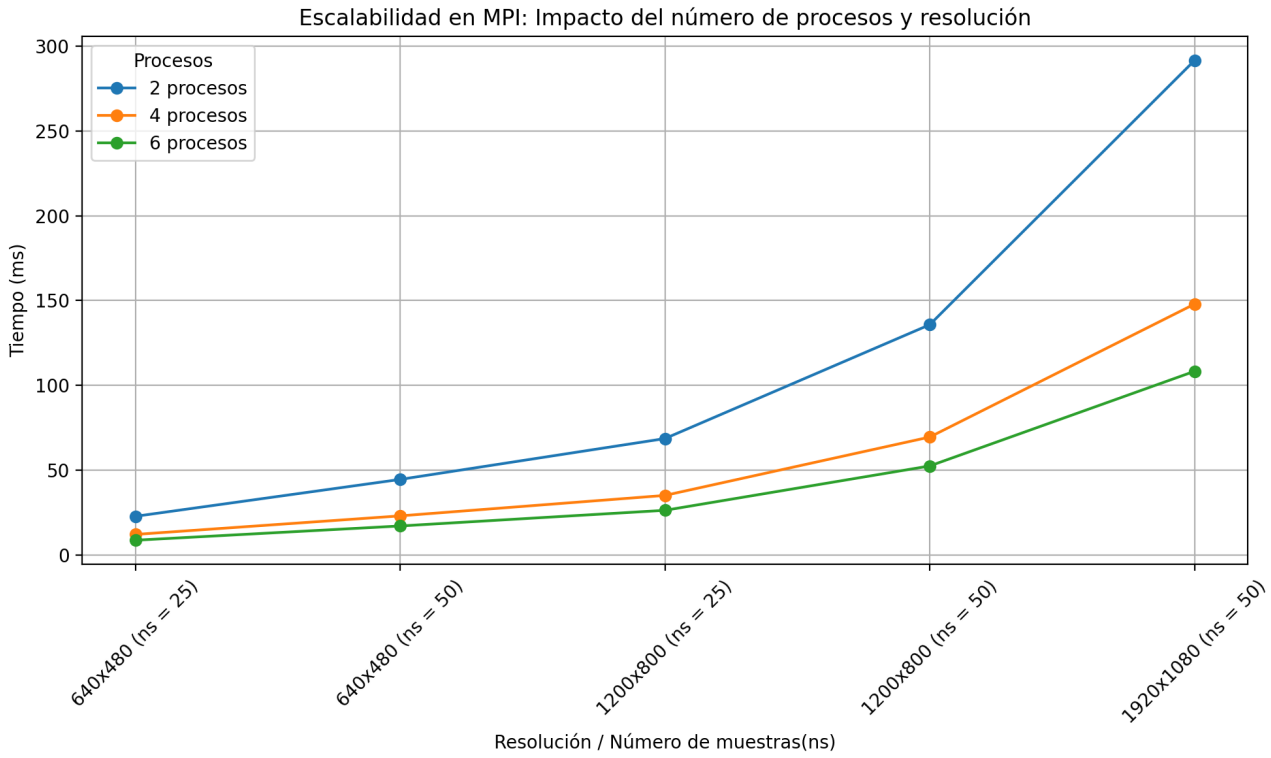


El análisis de escalabilidad muestra que, al aumentar la resolución y el número de samples, el tiempo de ejecución crece de forma progresiva y coherente. En cuanto a los hilos, se observa una mejora significativa al incrementar de 2 a 4 hilos, ya que se reduce la carga individual y se reparte el trabajo entre más hilos.

Sin embargo, al pasar de 4 a 6 hilos, la ganancia es menor; es casi insignificante, incluso para resoluciones mayores con más muestras por píxel. Probablemente, a partir de 4 hilos se alcanza un punto en el que aumentar más hilos no escala de manera significativa ni eficiente. También sugiere que, al aumentar el número de hilos, crece la sobrecarga por sincronización y overhead.

En cuanto a la resolución, se nota que el tiempo crece significativamente al añadir muchos más samples, ya que hay una mayor carga computacional. En definitiva , el escalado en OpenMP da buenos resultados hasta alcanzar cierto umbral de hilos , luego el beneficio disminuye por factores relacionados con la sincronización y creación de hilos.

4.4.2 MPI

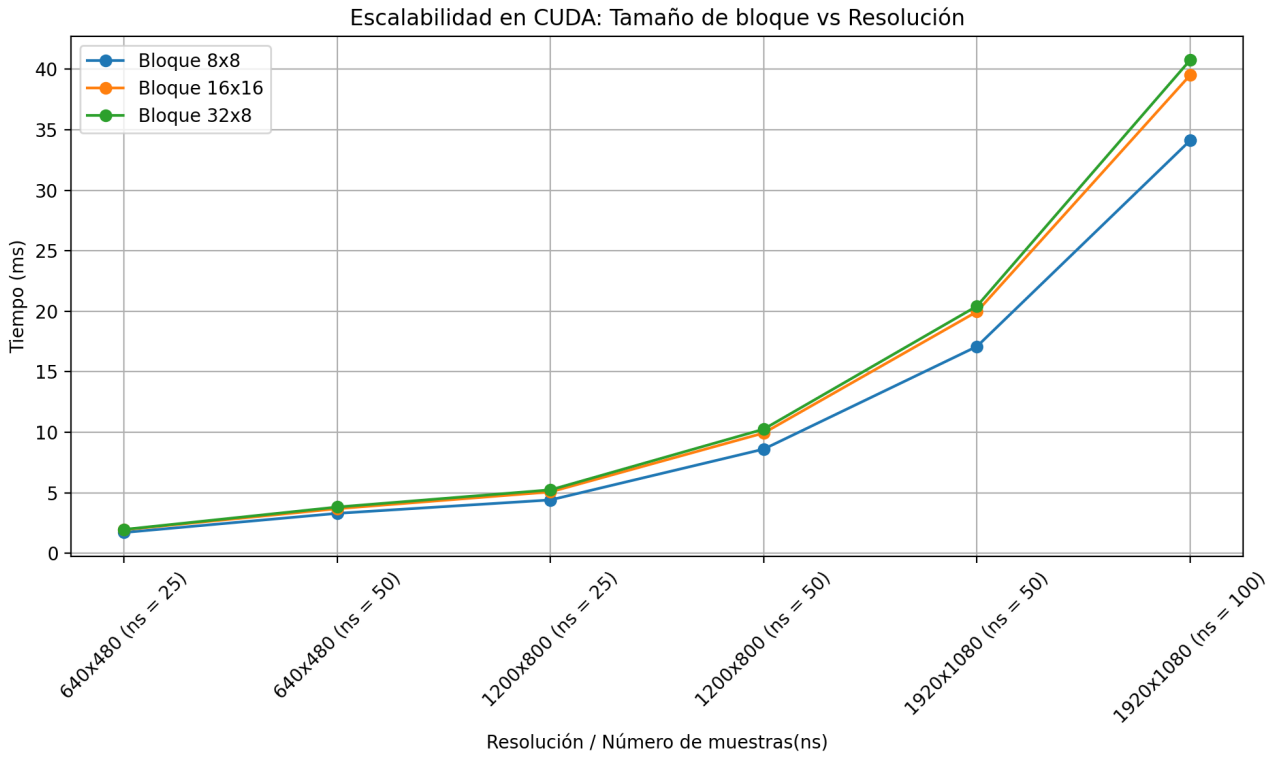


Los resultados de MPI muestran que escala eficientemente al aumentar la carga computacional (resolución y samples). Se observa una clara reducción del tiempo de ejecución al pasar de 2 a 4 procesos, y también al aumentar a 6. Aunque no es tan abrupta como la mejora de 2 a 4, sí es lo suficientemente notable como para mostrar una mejora que, como vemos en la gráfica, tiende a crecer a medida que se va complicando la escena: cada vez la diferencia es mayor.

A diferencia de OpenMP, se sugiere que no existe un punto umbral en el que deje de ser rentable añadir más nodos. Esto se debe a la arquitectura distribuida de MPI, que gestiona bien el paralelismo y la escalabilidad manteniendo la eficiencia.

En conclusión, MPI demuestra ser versátil al incrementar tanto la complejidad computacional como el número de nodos, sin verse limitado por umbrales. Es adecuada para ejecutar tareas de mayor complejidad en entornos distribuidos, siendo más eficiente que los modelos basados en memoria compartida (como OpenMP).

4.4.3 CUDA

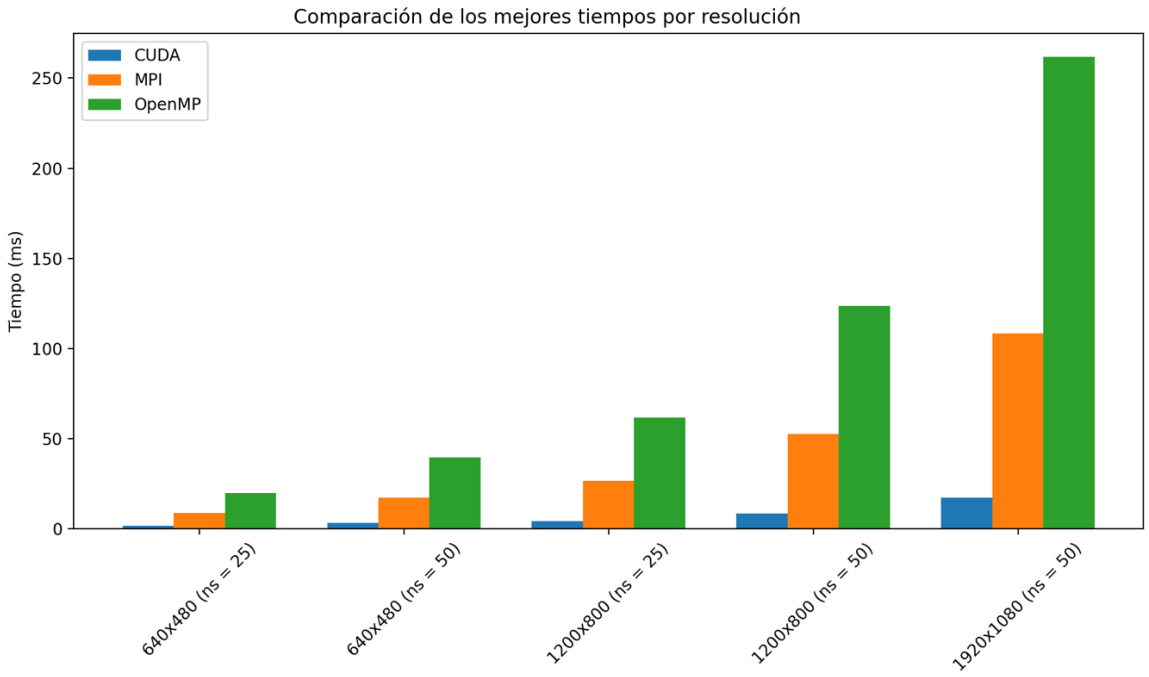


Los resultados obtenidos con CUDA muestran que, como se esperaba, el tiempo de ejecución aumenta al incrementar la resolución y el número de muestras por píxel, pues aumenta la carga de trabajo. Sin embargo, los tiempos se mantienen bajos en comparación con otras tecnologías, reflejando la capacidad de la GPU para manejar gran volumen de datos gracias a su enorme cantidad de hilos.

Al comparar distintas configuraciones de bloques, se observa que el tamaño 8x8 ofrece un mejor rendimiento, sobre todo en resoluciones más altas. En las primeras pruebas con imágenes más simples, la diferencia es mínima, pero a medida que la carga aumenta (hasta llegar a resoluciones altas y 100 muestras), los bloques de 8x8 se mantienen por debajo en tiempo respecto a los demás. Esto sugiere que este tamaño permite una mejor distribución del trabajo entre los warp, además de favorecer accesos más ordenados a memoria.

Aunque todas las configuraciones fueron implementadas para beneficiarse de la coalescencia, aumentar el tamaño no mejora el rendimiento. Podría estar relacionado con el número de registros, ocupación por bloque o capacidad de cada SM. Por eso, aunque se aumente de tamaño, como 16x16 o 32x8, y ofrezca buenos resultados, 8x8 parece ser el que logra aprovechar de mejor forma los recursos.

En resumen, CUDA demuestra una excelente escalabilidad en todas las configuraciones, pero el uso de bloques 8x8 es la configuración más eficiente y la que escala mejor con mayor carga.



Este gráfico resume de forma global los mejores tiempos por resolución obtenidos por el experimento anterior y permite observar claramente la **superioridad de CUDA frente a MPI y OpenMP**, independientemente de la resolución o la carga de trabajo evaluada.

5. Conclusiones

Durante esta práctica se han analizado profundamente las tecnologías de paralelización OpenMP, MPI y CUDA aplicadas al renderizado de imágenes mediante ray tracing. Cada tecnología presenta ventajas específicas según el tipo de paralelización y el entorno de ejecución.

CUDA es, con diferencia, la tecnología más eficiente, por su capacidad de lanzar miles de hilos en paralelo y por su arquitectura optimizada para tareas masivamente paralelas. Su rendimiento destaca especialmente en cargas elevadas y resoluciones altas, siempre que se configuren adecuadamente los bloques para favorecer la coalescencia de memoria.

MPI queda por debajo de CUDA como una solución intermedia, pero favorablemente escalable. En escenarios distribuidos o cuando se busca un alto grado de paralelismo sin necesidad de GPU, MPI ofrece buenos resultados, especialmente con estrategias dinámicas que reparten mejor la carga entre procesos.

OpenMP sigue siendo útil para cargas más ligeras o para introducir el paralelismo con menor complejidad. Sin embargo, su rendimiento se ve penalizado en algunos casos por la sobrecarga de sincronización y el overhead.

Finalmente, las estrategias híbridas (como MPI + OpenMP) permiten aprovechar mejor los recursos disponibles, combinando el control de procesos con los hilos, lo que mejora los tiempos frente a cada tecnología por separado.

En resumen, esta práctica ha permitido identificar fortalezas y limitaciones de cada enfoque, apoyando las conclusiones con datos experimentales sólidos tanto para cargas ligeras como para cargas pesadas, proporcionando así un panorama más amplio.

Como posibles mejoras, se hubiese querido experimentar con más tecnologías híbridas para compararlas con la única que se implementó, pues al tener solo una, no se pudo evaluar si añadir paralelismo a CUDA con MPI es más eficiente o si hay penalización por tener una única GPU gestionando tantas tareas y recursos simultáneamente.

6. Anexos

- Implementación original de la práctica (NVIDIA)

  Proyecto base de ray tracing acelerado con CUDA, proporcionado por NVIDIA.

- Módulo de la asignatura “Programación Concurrente”

  - Consultado especialmente para definiciones sobre paralelización por chunks, granularidad y reparto de tareas. Además, se empleó una imagen extraída del libro The Art of Concurrent Programming para ilustrar visualmente estos conceptos.

- Código fuente y archivo README.md

  Incluyen explicaciones sobre la compilación, ejecución de pruebas y estructura de cada implementación (OpenMP, MPI, CUDA e híbrida).