Practica 2 Computación de Altas Prestaciones

Renderizado de películas en ray tracing utilizando MPI, OpenMP y CUDA

Autores: Andrés Tena De Tena y Gabriel Morales Dato
Titulación: Ingeniería de Computadores
Curso: 2023/2024

Fecha: 20 de mayo de 2024

Índice

1. Comparación de versiones

1.1. Versión secuencial

En esta versión no se utiliza ningún tipo de mejora. Un único nodo con un hilo de la CPU procesa secuencialmente las imágenes y escribe el bmp.

Parámetros de ejecución:

- Mejoras: Ninguna
- Nodos: 1
- Hilos por nodo: 1
- Frames procesados: 9
- Tamaño de los frames: 512 x 225
- Tiempo de ejecución: 113.522 segundos

Conclusión: esta es la versión más lenta del programa y la práctica consiste en optimizarla para mejorar el tiempo de ejecución.

1.2. Versión MPI

En esta versión se utiliza MPI con una estructura maestro-esclavo. El maestro se encarga de enviar semillas a los esclavos para que estos generen una imagen en base a esa semilla. Después de procesar la imagen, los esclavos la envían de nuevo al maestro, que la guarda en un vector. Al terminar de procesar todas las semillas, el maestro escribe las imágenes secuencialmente, leyéndolas del vector donde se han guardado previamente.

Parámetros de ejecución:

- Mejoras: MPI
- Nodos: 1 maestro y 5 esclavos
- Hilos por nodo: 1
- Frames procesados: 9
- Tamaño de los frames: 512 x 225
- Tiempo de ejecución: 13 segundos por nodo e imagen, 26 segundos en total.

Conclusión: Hay una mejora notable respecto al tiempo secuencial. El tiempo total es proporcional al número de nodos: a mayor número de nodos, menos tardará la ejecución.

1.2.1. Ejecución en el clúster de la universidad

Esta versión también la hemos probado en el clúster de la universidad. En concreto, hemos usado el de 32 hilos. La ejecución es más rápida que en nuestro portátil porque el procesador del clúster es un AMD Epyc dedicado para servidores, y el clúster tiene menos tareas en ejecución.

Parámetros de ejecución:

- Mejoras: MPI
- Nodos: 1 maestro y 31 esclavos
- Hilos por nodo: 1
- Frames procesados: 56
- Tamaño de los frames: 512 x 225
- Tiempo de ejecución: 12 segundos por nodo e imagen, 24 segundos en total.

También hemos intentando ejecutar esta versión en varios clusters, mediante un fichero hosts, pero debido a problemas con el firewall no hemos podido probar esta parte.

Conclusión: El clúster no solo es más rápido sino que también permite procesar muchas más imágenes simultáneamente, mejorando el tiempo total de manera significativa.

1.3. Versión MPI + OMP

En esta versión se utiliza MPI con una estructura maestro-esclavo de igual forma que en la versión anterior. La mejora adicional de esta versión es que utiliza OpenMP en todos los nodos.

Por cada imagen que el maestro envía al esclavo, el esclavo calcula parches de esa imagen con diferentes hilos para luego enviar toda la imagen al maestro. El maestro además, a la hora de escribir las imágenes, lo realiza de manera paralela, utilizando diferentes hilos para escribir las imágenes simultáneamente. Parámetros de ejecución:

- Mejoras: MPI y OMP
- Nodos: 1 maestro y 5 esclavos
- Hilos por nodo: 9
- Frames procesados: 9
- Tamaño de los frames: 512 x 225
- Tiempo de ejecución: 10 segundos por nodo e imagen, 20 segundos en total.

Conclusión: El hecho de utilizar OMP en los esclavos y en el maestro ha mejorado un 25 % la versión anterior. Hemos realizado pruebas con diferente número de hilos por nodo, y hemos llegado a la conclusión de que utilizar muchos hilos empeora el rendimiento considerablemente, así como utilizar muy pocos hilos o ninguno. Esto depende del renidimiento de cada máquina y en nuestro caso hemos encontrado que 9 hilos es una cantidad adecuada.

1.4. Versión CUDA

En esta versión se utiliza CUDA en un único nodo, sin MPI.
Primer caso:

- W: 512
- H: 256
- Frames: 8

En este caso vamos a producir 8 imágenes a través de la GPU y 1 con la CPU. Por cada frame procesado por la GPU, se procesará un parche de 32x512 en la CPU, es decir, cuando acabemos la ejecución se habrán procesado 8 parches de 32x512 (una imagen completa). En este caso se puede apreciar como el tiempo que espera la CPU en cuanto a la sincronización con la GPU para poder guardar las imágenes es prácticamente nulo. Debido a que muy probablemente la GPU sea la que esta esperando a la CPU, esta suposición fue lo que nos motivo a probar con el segundo caso. En este caso se puede observar como el tiempo medio invertido por imagen es "25,569 s".

— GPU —
Rendering a 512x256 image with 10 samples per pixel in 8x8 blocks.
Loading took 0.018 seconds.
Imagen -0
Llamando a CPU...
Tiempo de CPU parada: 0.0408 ms
Imagen GPU creada...
Imagen -1
Llamando a CPU...
Tiempo de CPU parada: 0.0271 ms
Imagen GPU creada...
Imagen -2
Llamando a CPU...
Tiempo de CPU parada: 0.0272 ms
Imagen GPU creada...
Imagen -3
Llamando a CPU...
Tiempo de CPU parada: 0.0255 ms
Imagen GPU creada...
Imagen -4

Llamando a CPU...
Tiempo de CPU parada: 0.0236 ms
Imagen GPU creada...
Imagen -5
Llamando a CPU...
Tiempo de CPU parada: 0.0244 ms
Imagen GPU creada...
Imagen -6
Llamando a CPU...
Tiempo de CPU parada: 4542.55 ms
Imagen GPU creada...
Imagen -7
Llamando a CPU...
Tiempo de CPU parada: 12132.1 ms
Imagen GPU creada...
Imagen CPU creada...
Rendering took 230.088 seconds.
Copy took 0.013 seconds.
Tiempo total procesado: 230123 ms

Tiempo por imagen: 230123/9= 25569,22 ms = 25,569 s

Segundo caso:

- W: 512
- H: 256
- Frames: 16

En este caso vamos a producir 16 imágenes a través de la GPU y 1 con la CPU. Por cada frame procesado por la GPU, se procesará un parche de 16x512 en la CPU, es decir, cuando acabemos la ejecución se habrán procesado 16 parches de 16x512 (una imagen completa). En este caso se puede apreciar como el tiempo que espera la CPU en cuanto a la sincronización con la GPU para poder guardar las imágenes aumenta en algunos casos. Pero sin embargo el tiempo por imagen producida es igual a "22,69 segundos.[el] cual es menor que en el primer caso, por tanto en ese otro caso se puede deducir que la GPU acaba bastante antes que la CPU y por tanto se está perdiendo tiempo de cómputo, ya que la CPU no tiene que esperar nunca a la GPU.

— GPU —
Rendering a 512x256 image with 10 samples per pixel in 8x8 blocks.
Loading took 0.015 seconds.
Imagen -0
Llamando a CPU...
Tiempo de CPU parada: 7974.64 ms
Imagen GPU creada...
Imagen -1

Llamando a CPU...
Tiempo de CPU parada: 10768.1 ms
Imagen GPU creada...
Imagen -2
Llamando a CPU...
Tiempo de CPU parada: 11515.1 ms
Imagen GPU creada...
Imagen -3
Llamando a CPU...
Tiempo de CPU parada: 10579 ms
Imagen GPU creada...
Imagen -4
Llamando a CPU...
Tiempo de CPU parada: 10600.3 ms
Imagen GPU creada...
Imagen -5
Llamando a CPU...
Tiempo de CPU parada: 5135.05 ms
Imagen GPU creada...
Imagen -6
Llamando a CPU...
Tiempo de CPU parada: 7014.42 ms
Imagen GPU creada...
Imagen -7
Llamando a CPU...
Tiempo de CPU parada: 8154.41 ms
Imagen GPU creada...
Imagen -8
Llamando a CPU...
Tiempo de CPU parada: 7772.41 ms
Imagen GPU creada...
Imagen -9
Llamando a CPU...
Tiempo de CPU parada: 9189.13 ms
Imagen GPU creada...
Imagen -10
Llamando a CPU...
Tiempo de CPU parada: 10836.2 ms
Imagen GPU creada...
Imagen -11
Llamando a CPU...
Tiempo de CPU parada: 10866 ms
Imagen GPU creada...
Imagen -12
Llamando a CPU...
Tiempo de CPU parada: 13900.5 ms

Imagen GPU creada...
Imagen -13
Llamando a CPU...
Tiempo de CPU parada: 15593.2 ms
Imagen GPU creada...
Imagen -14
Llamando a CPU...
Tiempo de CPU parada: 16334.5 ms
Imagen GPU creada...
Imagen -15
Llamando a CPU...
Tiempo de CPU parada: 18316.3 ms
Imagen GPU creada...
Imagen CPU creada...
Rendering took 385.706 seconds.
Copy took 0.011 seconds.
Tiempo total procesado: 385737 ms

Tiempo por imagen: 385737/17 = 22690,41 ms = 22,69041 s