INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

Curso Académico 2009/2010

Proyecto de Fin de Carrera

Estudio de una implementación para renderizado en paralelo con Yafaray

Autor: Noelia González Méndez
Tutores:
Óscar David Robles Sanchez
Jorge Gascón Pérez

Agradecimientos

Agradezco a mis tutores Óscar Robles, Pablo Toharia y Jorge Gascón toda la ayuda y atención prestada.
Por haber sabido guiarme y ayudarme en la elaboración de este proyecto.

Gracias a Jorge por haberme ayudado en todo lo relacionado con blender y yafaray.
Gracias a Óscar y a Pablo por haberme ido guiando poco a poco para conseguir las pequeñas metas que finalmente han constituido mi PFC.
También debo hacer mención al profesor José Luis Bosque y a la universidad de Cantabría, gracias a su ayuda he podido realizar pruebas para mi proyecto.

También agradezco a Óscar la confianza puesta en mí desde el primer momento.

Por último agradecer a mi familia todo el apoyo que me han proporcionado.

Resumen

Hoy día el uso de gráficos 3D es una tecnología muy demandada, esto es debido a la multitud de áreas en las que están presentes.
Es por ello que el diseño de gráficos 3D por computador ha obtenido un papel de gran importancia dentro del mundo de la computación.
Entre sus muchos usos se pueden destacar algunos como su utilización para la creación de videojuegos, de películas y simulación de distintos tipos de entornos.

La creación de gráficos 3D por computador está compuesta por una serie de fases, una de las más importantes es la de renderizado, concretamente esta fase necesita de una gran capacidad de cómputo.
En general crear gráficos 3D es una tarea costosa que requiere de una gran cantidad de tiempo y de una gran capacidad de cómputo, es por ello que se están buscando formas de poder reducir el tiempo invertido en realizar esta tarea utilizando al máximo todos los recursos disponibles del computador.

De esta idea surge la motivación de este proyecto el cual intenta abordar una reducción en el tiempo empleado en la obtención de gráficos, utilizando varios procesadores que trabajen de forma colaborativa.

Para ello se ha implementado un algoritmo que se encarga de paralelizar la tarea de renderizado.
Su ejecución ha sido probada sobre diferentes arquitecturas, como un multiprocesador de memoria compartida y un *cluster*, obteniendo unos resultados que verifican el cumplimiento de los objetivos propuestos.

Índice general

Índice de Figuras

Índice de Tablas

1. Introducción

El presente proyecto de fin de carrera se ha realizado en colaboración con el Departamento de Arquitectura y Tecnología de Computadores y Ciencia de la Computación e Inteligencia Artificial de la Escuela Técnica Superior de Ingeniería Informática de la Universidad Rey Juan Carlos.

Los gráficos en 3D se forman mediante un proceso de cálculos matemáticos sobre entidades geométricas tridimensionales, se puede decir que la finalidad de un gráfico en 3D es conseguir una proyección visual en dos dimensiones para ser mostrada posteriormente en una pantalla o sobre papel.

En la computación se utilizan los gráficos en 3D para crear animaciones, gráficos, películas, juegos, realidad virtual, diseño, etc.

Para poder crear gráficos en 3D hay que pasar por una serie de fases que se citan y explican a continuación [1,2]:

- Fase de Modelado: La etapa de modelado consiste en crear una imagen bidimensional de una escena o un objeto. Es decir, ir modelando cada uno de los objetos que posteriormente formarán la imagen.

  Existen diversos tipos de modelado entre los que se pueden mencionar el modelado poligonal, CSG, o las técnicas de subdivisión espacial o representación de objetos con funciones implícitas.

- Fase de Composición de la escena: esta etapa engloba las actividades de iluminación, distribución de objetos, cámaras y otro tipo de elementos que juntos crean un gráfico estático o dinámico. La iluminación del gráfico implica la creación de luces de diversos tipos, como pueden ser las puntuales, direccionales en área o volumen con distinto color, o potencia, etc.

- Fase de Renderizado: Se llama rénder al proceso final de obtener una imagen bidimensional o animación a partir de la escena creada, es decir es la

transformación de un modelo en 3D hacia una imagen. Para la realización de esta fase se han desarrollado diversas técnicas, algunas más complejas que otras. Algunas de las técnicas son rénder de alambre (*wireframe rendering*), rénder basado en redes de polígonos, renderización de una descripción CSG, *raytracing* (trazado de rayos) y de funciones implícitas, entre otras muchas técnicas.

El proceso de rénder necesita una gran capacidad de cálculo. La capacidad de cálculo se ha incrementado rápidamente a través de los años permitiendo un grado superior de realismo en los rénders.

Una de las técnicas más importantes de renderizado es el *raytracing*, concretamente el proceso de rénder denominado *Raytrace* se encarga de simular efectos como las reflexiones, refracciones y el sombreado de luces en las superficies.

Cada una de estas fases anteriores es esencial para obtener un gráfico en 3D.

Como se cita anteriormente hay diversas herramientas que se encargan de realizar gráficos en 3D, en particular en este proyecto las herramientas utilizadas han sido blender y yafaray, cuyas principales características se exponen a continuación.

Blender es un programa libre que permite la realización de modelado, animación, iluminación y renderizado.[3]

Permite importar y exportar distintos formatos de imagen 2D y modelos y escenas 3D.

En definitiva es un programa dedicado especialmente al modelado y a la creación de gráficos tridimensionales.

Posee una interfaz gráfica poco intuitiva, ya que está enfocada para usuarios expertos, para los cuales ésta posee algunas ventajas muy importantes con respecto a otras aplicaciones, como por ejemplo la configuración personalizada de la distribución de los menús o las vistas de cámara.

Algunas de las características esenciales de esta aplicación son: que es multiplataforma, libre y gratuito, aparte posee una gran variedad de primitivas geométricas, permite la simulación de pelo, fluidos, *softbodies*, permite guardar todas las escenas en ficheros independientes, permite editar formatos muy variados de audio y video, además de que tiene posibilidad de renderizado e integración externa de potentes herramientas como por ejemplo Yafaray

La otra aplicación que se complementa con Blender y que también ha sido necesaria para la elaboración del presente proyecto ha sido yafaray (motor de render).

Yafaray (*Yet Another Free Ray tracer*) es un *raytracer* libre. *Raytracing* es una técnica de interpretación para poder generar imágenes reales. Esto se consigue trazando el camino que sigue la luz a través de una escena de 3D. Yafaray es un motor de renderizado que puede utilizarse como plug-in de Blender.[4]

Utiliza un lenguaje de descripción de escenas en XML y está publicado bajo la licencia LGPL 2.1.

Algunas de las principales características son:

- Iluminación global.
- Iluminación de fondo, este tipo de iluminación es la proveniente de un cielo.
- Efectos Caústicos: esta característica se encarga de la concentración de luz producida por objetos reflectantes o por refracción.
- Materiales: Yafaray posee cuatro tipos de sombreado(*ShinyDiffuse*, Brillante, *CoatedGlossy* **y** Vidrio) que permiten simular cualquier tipo de material
- Texturas: Posee una mezcla de diferentes texturas.
- Cámaras: Yafaray posee implementaciones de cuatro tipos de cámaras distintas para producir diferentes efectos.
- Volumétrica: Para hacer volumetría este programa se basa en la física permitiendo crear niebla, humo, nubes entre otros muchos efectos.

Para poder realizar gráficos en 3D se necesitan herramientas y computadoras potentes, ya que durante la creación de gráficos de este tipo se consume tiempo y recursos de la computadora.

Gracias a las progresivas mejoras tecnológicas desarrolladas durante las décadas pasadas, la capacidad de procesamiento de los ordenadores ha aumentado cada vez más.

De todas formas, estas mejoras tecnológicas tienen un límite físico, por lo cual el aumento de capacidad de procesamiento no es ilimitado.

Es por eso que cuando se plantea el resolver un problema con un computador, según la capacidad de procesamiento que se necesite, una de las cuestiones a decidir es si se resolverá con una sola máquina, es decir de forma secuencial o con varias, programación en paralelo.

Para que un ordenador adquiera una mayor capacidad de procesamiento se puede aumentar el número de procesadores, de esta forma se paralelizaría la tarea a realizar, aunque una de las cuestiones que hay que tener en cuenta es que no todo problema puede ser resuelto de forma paralela.

La programación en paralelo se apoya en:

-Paralelismo en los datos (*data parallelism*). Paralelismo en los datos se basa en realizar operaciones en paralelo sobre un conjunto de datos.

-Paralelismo en el flujo de control del programa (*control parallelism*). El paralelismo en el control del programa se basa en tener varios flujos de control ejecutándose en paralelo.

A continuación se explicará la clasificación de arquitecturas según la taxonomía propuesta por Michael J. Flynn.

La clasificación definida por M.J. Flynn se basa en el número de instrucciones que se van ejecutando y los datos que utilizan dichas instrucciones [5]:

- SISD: Arquitectura con un flujo de datos y otro de instrucciones.
- SIMD: Arquitectura con varios flujos de datos y un solo flujo de instrucciones.
- MISD: Arquitectura con varios flujos de instrucciones y un solo flujo de datos. Esta arquitectura nos es muy común ya que los múltiples flujos de instrucciones normalmente requieren múltiples flujos de datos.
- MIMD: Varios procesadores que ejecutan varios flujos de instrucciones y varios flujos de datos.

Las unidades centrales de proceso en ordenadores paralelos funcionan bajo el control centralizado de una sola unidad de control o trabajan independientemente, por separado, es decir aparecen con configuraciones SIMD o MIMD: [6,7]

- SIMD: Una única Unidad de Control. La misma instrucción se ejecuta síncronamente por todas las unidades de procesamiento. Requiere menos hardware porque sólo necesita una unidad de control global y menos memoria porque tiene una sola copia del programa. Este tipo de computadores son más adecuados para ser programados en un estilo de paralelismo de datos.

- MIMD: Cada procesador posee una unidad de tratamiento y una unidad de control y en cada uno de ellos se almacena el programa y el Sistema Operativo.

  Al contrario que el diseño SIMD, los computadores que dispongan de un diseño MIMD son más adecuados para ser programados en un estilo de paralelismo con varios flujos de control.

  En el caso de máquinas con diseño MIMD se utilizan 2 tipos de programación: uno es el *fork-join* y el otro el *SPMD* (*Single Program Multiple Data*).

  En el tipo *fork-join* un proceso se divide en varios subprocesos.

En el *SPMD* se realizan varias instancias del mismo programa, es decir todos los procesos ejecutan el mismo programa, pudiéndose diferenciar algunas partes del código dependiendo del identificador que tenga el proceso.

La variable que indica el identificador de cada proceso es *process*, de esta forma el proceso 0 tendrá la variable *process* con valor 0 y el proceso 1 su variable *process* será 1 y así con todos los procesos que ejecuten el programa.

Por lo tanto tan sólo existirá una copia del código en la memoria, aunque cada proceso tenga sus propias variables, de esta forma se reduce el espacio ocupado en memoria.

Como conclusión se puede decir que las máquinas con diseño SIMD requieren menos sowftware que otras que tengan diseño MIMD porque ellas tienen sólo una unidad de control global. Además, las máquinas que tengan diseño SIMD requieren menos memoria porque sólo se necesita una copia del programa, al contrario que el diseño MIMD que almacena el programa y el sistema operativo en cada procesador.[7]
Actualmente el diseño SIMD genera una gran impopularidad, esto es debido a que el poder diseñarlos requiere un gran esfuerzo que normalmente supone un largo plazo de tiempo, esto no sucede con las arquitecturas MIMD cuyo diseño requiere poco esfuerzo y componentes relativamente baratos.

Otro aspecto a tener en cuenta es que hay dos formas de intercambio de datos entre tareas paralelas accediendo a un espacio de datos compartido y utilizando mensajes.
Cuando el computador tiene un espacio de dirección compartido, esto quiere decir que posee un espacio de datos común al que pueden acceder todos los procesadores, en este tipo de plataformas la memoria puede ser local o global.
Según el tipo de memoria se puede hacer la siguiente clasificación:

- Memoria distribuida (Memoria local)

Este tipo de arquitecturas tienen para cada procesador su propia memoria local.

Y se utiliza paso de mensajes para el intercambio de datos.

- Memoria Compartida (memoria global)

  Las arquitecturas que sean de este tipo tendrán una única memoria para todos los procesadores. Todos los procesos tienen acceso a la memoria a través de una red de conexión.

En la actualidad para la resolución de problemas complejos se está investigando intensamente en el procesamiento paralelo. Esta es la principal línea de investigación para mejorar la potencia de los ordenadores ya que nos proporciona una mayor velocidad de ejecución y mucha más precisión en los cálculos, además de tener una muy buena relación en cuanto al coste y las prestaciones.
Esta línea de investigación ha dado lugar a los multiprocesadores y los multicomputadores.

Un multicomputador es una máquina de memoria distribuida. Está compuesta por una serie de computadores completos, los cuales reciben el nombre de nodos, la comunicación entre ellos se realiza a través de una red utilizando paso de mensajes. [5]

Un multiprocesador es una máquina que dispone de varias unidades de proceso y una memoria común. Según el esquema de memoria que posea pertenecerá a uno de estos tres modelos: [5]

- UMA (*Uniform Memory Access*): en este modelo todos los procesadores comparten la memoria principal y tienen la misma facilidad para acceder a cualquier parte de ella.
- NUMA (*Non Uniform Memory Access*): al igual que en el modelo anterior todos los procesadores comparten la memoria principal, la diferencia es que no todos pueden acceder con la misma facilidad a ciertas zonas de la memoria.
- COMA (*Cache Only Memory Architecture*): en este modelo los procesadores acceden a la memoria a través de sus caches.

Cabe mencionar el modelo CC-NUMA (*Cache Coherent Non Uniform Memory Access*), caso particular del modelo COMA, ya que una de las arquitecturas en las que se ha probado el programa está diseñada según este modelo. Al igual que el modelo

COMA posee una memoria distribuida, lo que le diferencia del modelo NUMA. Este modelo utiliza una caché coherente

2. Objetivos y estructura del documento

2.1. Objetivos

Renderizar, es decir obtener la imagen correspondiente a un modelo 3D, es una tarea que requiere una gran cantidad de cálculos matemáticos, por ello se define como una tarea costosa que tarda bastante tiempo en realizarse.

La tarea de renderizar un gran número de gráficos 3D de gran tamaño en un procesador es de larga duración, esto sería posible en una sola máquina pero emplearía demasiado tiempo de ejecución, esto dependería de los recursos que nos proporcionara la computadora y de su capacidad de cómputo.

Para poder reducir este tiempo de ejecución se podría paralelizar esta tarea, de tal forma que cada uno de los procesadores renderizara una serie de gráficos 3D, así la tarea pesada de renderizar estaría repartida entre los distintos procesadores y el tiempo de ejecución se reduciría notablemente.

El objetivo principal del presente proyecto es la creación de un algoritmo que se encargue de paralelizar la tarea de renderizado, teniendo en cuenta los distintos procesadores que pueda utilizar en cada caso reduciendo el tiempo de ejecución lo máximo posible.

Para poder probar el algoritmo ejecutándose en paralelo, hay que utilizar diferentes arquitecturas que nos lo permitan, en este caso se han empleado para realizar las pruebas un *cluster* y un multiprocesador de memoria compartida.

Este objetivo principal a su vez conlleva otros pequeños objetivos:

- Estudio del programa que se va a utilizar para la renderización. Crear a partir de una imagen el conjunto de ficheros .xml que se necesitarán para probar el

programa. La obtención de estos ficheros se hace posible mediante la utilización de la herramienta blender con una extensión de yafaray.

- Estudio de las herramientas y máquinas que se utilizarán para la implementación paralela.

- Creación del programa que se encarga de realizar el renderizado paralelo.

- Realizar diversas pruebas, para poder comparar el tiempo de ejecución, dependiendo de los procesadores utilizados en cada caso y de la carga de envío.

Una vez cumplidos todos estos objetivos este proyecto valdrá para realizar estudios comparativos con otros proyectos que tienen el mismo objetivo pero que utilizan otras herramientas.

2.2. Estructura del documento

La estructura del presente documento dividida por capítulos es la siguiente:

- Capítulo 1: Introducción
  Este capítulo realiza una breve introducción, de lo que posteriormente se tratará en el documento.

- Capítulo 2: Objetivos y estructura del documento
  En este capítulo se citan los objetivos del proyecto explicando de forma detallada cada uno de ellos, y la estructura del documento.

- Capítulo 3: Descripción Informática
  Este capítulo explica tanto el código fuente del maestro como del esclavo y qué es lo que realiza cada uno de ellos.
  También proporciona una descripción de la librería utilizada, MPI.

- Capítulo 4: Resultados Experimentales

  Dentro de este capítulo hay una descripción de las máquinas en las que se ha probado el programa creado, también se proporciona una explicación acerca de cómo se ejecuta en cada una de ellas.

  En otro apartado de este capítulo aparecen una serie de resultados obtenidos tras las pruebas en las diferentes arquitecturas, mostrados en tablas y gráficas.

- Capítulo 5: Conclusiones

  En este capítulo se presentan las conclusiones obtenidas a partir de los resultados del apartado anterior.

  Y una serie de ideas con las cuales podría continuar el trabajo del proyecto.

3. Descripción informática

A continuación se describirá cada una de las etapas que se han seguido para poder desarrollar este proyecto.
Se podría dividir inicialmente en dos grande etapas:

-La primera de ellas engloba la creación del programa encargado de realizar el renderizado en paralelo.

-La segunda sería toda la parte de experimentación y pruebas, todas ellas necesarias para medir los datos de interés (como tiempo de ejecución y de comunicación) y poder tener unas conclusiones con las que más tarde se realizarán estudios comparativos con otros programas.

Una vez mencionadas las etapas, a continuación se mostrará una explicación más detallada y exhaustiva de cada una de ellas.

3.1. Obtención de los parámetros de entrada del programa

A su vez esta etapa se podría dividir en varias partes, cada una de ellas concuerda con uno de los diferentes objetivos que se han ido planteando en la realización del proyecto.

-Uno de los primeros objetivos fue la familiarización con las herramientas de trabajo que se iban a utilizar, así como la puesta a punto del software necesario.
(blender, yafaray…)

-Otro objetivo fue aprender acerca de blender ya que con esta herramienta es con la que se obtendrían las primeras imágenes con las que posteriormente se probaría el programa.

La imagen utilizada para realizar las pruebas debía cumplir una serie de requisitos, tenía que ser un gráfico 3D que no contuviera demasiados elementos, de esta forma la fase de renderizado no sería de excesiva duración.
Pero tampoco podía ser una imagen demasiado simple, ya que si no al realizar las pruebas no se percibirán bien los tiempos y las mediciones que se desean observar.
Finalmente la imagen utilizada para realizar todas las pruebas, se obtuvo de la página de yafaray, estaba compuesta por dos bombillas, pero para adaptarla a las condiciones requeridas, tuvo que ser modificada.
Las modificaciones que se llevaron a cabo fueron la extracción de una de las bombillas y dotar de movimiento a la bombilla restante en diferentes fotogramas.
Gracias a la cámara situada y a la luz (ambos objetos pertenecen a blender) podemos ver el fotograma que se muestra.
Este gráfico posee movimiento a lo largo de 1000 fotogramas que es lo que se ha indicado en el menú de configuración.

Tras tener esta imagen en 3D en blender, modificando algunas propiedades del programa se obtuvieron las especificaciones xml de las imágenes, que es lo que se necesitaba ya que son los datos de entrada del programa.
Una de las características importantes de los parámetros de entrada es que son ficheros xml independientes es decir, por cada fotograma se obtiene una especificación xml distinta, que posteriormente tendrá que ser proporcionada al programa para que se encargue de la renderización.
Para poder obtener el fichero xml, es necesario instalar el plugin de yafaray en blender.
Un fichero xml contiene DTD (*Document Type Definition*) esto permite definir la estructura y los elementos que forman la escena 3D que contiene el fichero.
En la especificacón xml de cada uno de los fotogramas, están contenidas las descripciones detalladas de los elementos que forman el gráfico 3D, la forma, el color, la textura, la posición de cada objeto, los materiales empleados y como afectan cada uno de ellos a la escena, la colocación de las cámaras, las luces, etc.

La imagen utilizada es la siguiente:
Primero se muestra el gráfico 3D y a continuación las imágenes renderizadas de ese gráfico.

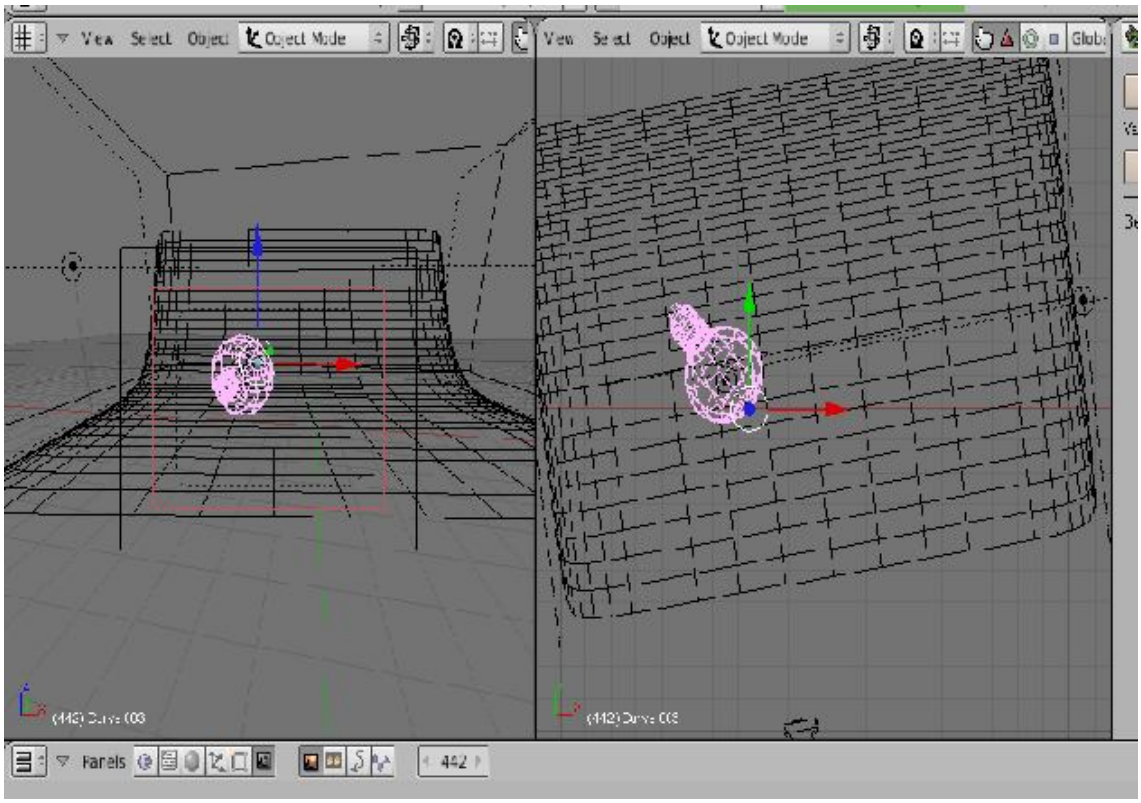

Figura 1: Gráfico creado con blender

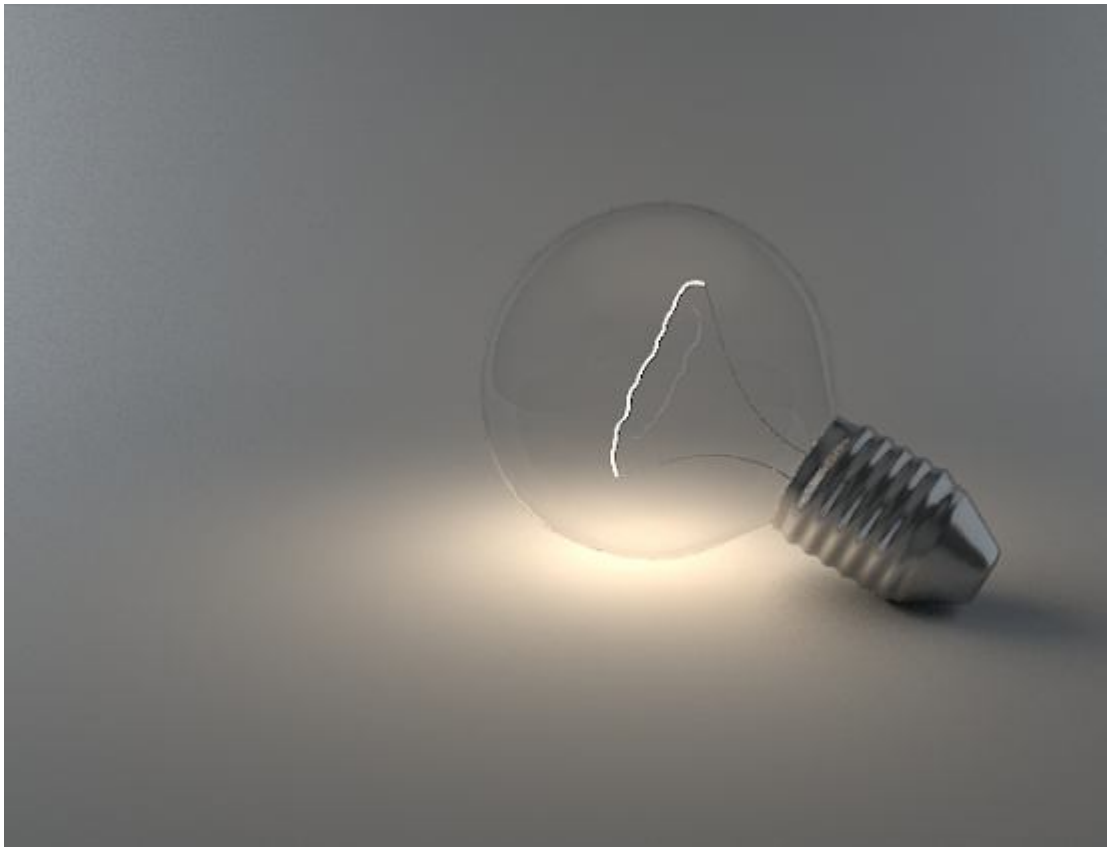

Figura 2: Imagen renderizada del gráfico 3D anterior

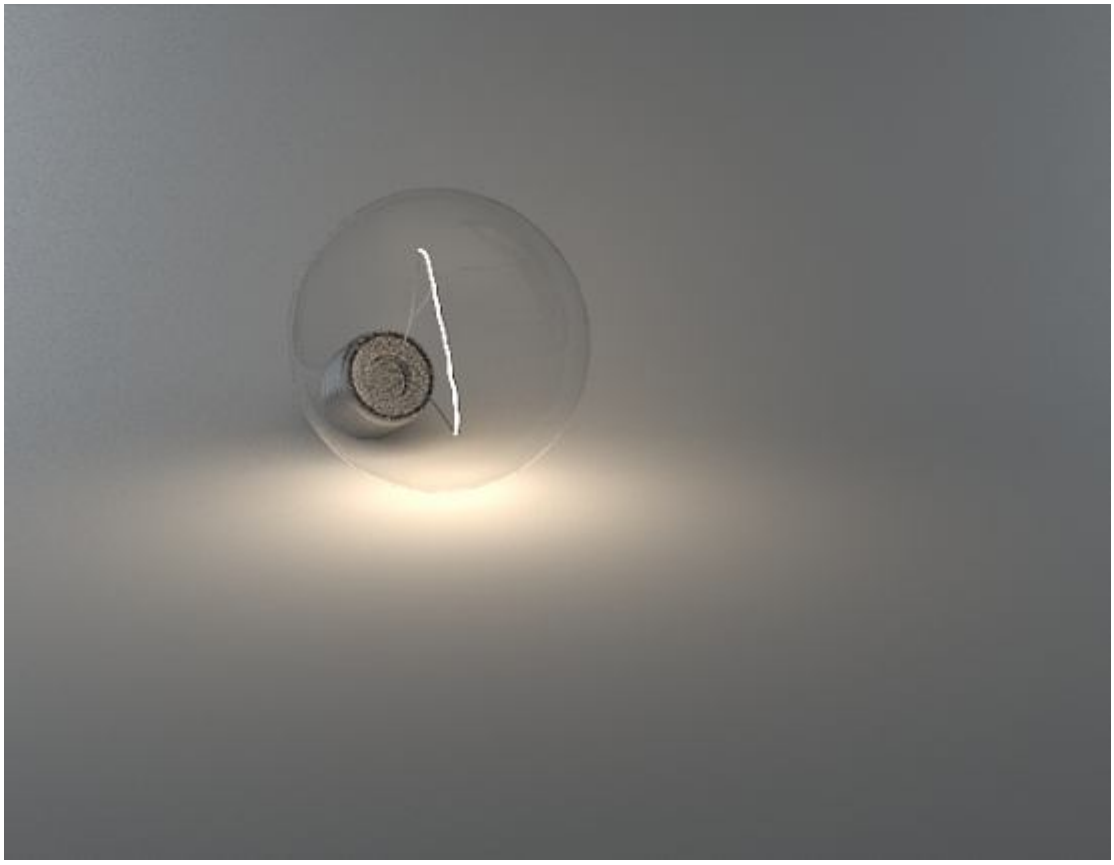

Figura 3: Imagen renderizada de un fotograma distinto al anterior

Las dos imágenes anteriores muestran la imagen inicial y la final de los fotogramas utilizados para las pruebas, el resto de fotogramas intermedios muestran el movimiento de rotación de la bombilla.

3.2. Implementación del código

Para la implementación del programa el lenguaje utilizado ha sido C esto ha sido por diferentes motivos: uno de ellos es por la eficiencia y portabilidad que nos aporta, otro motivo es que la librería utilizada, MPI, está preparada para C, y otro de los motivos es que al estar GNU/Linux programado en C, se obtiene una mayor flexibilidad y facilidad para acceder a llamadas al sistema.

A continuación se muestra una descripción de la librería utilizada para poder gestionar el paso de mensajes.
MPI (*Message Passing Interface*) es una especificación de biblioteca propuesta como un estándar del cual existen varias implementaciones que permite el paso de mensajes, facilitando así la comunicación entre procesos. [8,9]
MPI ha sido desarrollado por el MPI Forum, este grupo está constituido por investigadores de universidades, laboratorios y empresas todos ellos con un mismo fin la computación de altas prestaciones.

Este estándar proporciona funciones que nos permiten implementar lo citado en el párrafo anterior para C, C++ o Fortran.
Otra de las características útiles de estas librerías es que nos permiten crear programas que posteriormente podrán ser utilizados en diferentes computadores paralelos.

Como se ha mencionado anteriormente MPI es un estándar del cual existen varias implementaciones, concretamente se van a mencionar dos de ellas LAM/MPI y MPICH2 ya que son las utilizadas en las arquitecturas en las que se han realizado las pruebas.

LAM-MPI es una implementación libre de MPI. Constituye un poderoso entorno para ejecutar y monitorizar aplicaciones MPI sobre redes de ordenadores. [10]

La utilización es la siguiente: el programa se escribirá como un proceso secuencial, del cual se lanzarán varias instancias que trabajarán juntas en la ejecución.

El fichero que hay que ejecutar, es un fichero en el que indicamos por cada línea una instancia, la ruta en la que se encuentra el ejecutable y los parámetros que necesita nuestro programa.

MPICH2 es una implementación portable de MPI disponible libremente. El fichero que hay que ejecutar se especificará en la parte de la arquitectura que utiliza esta librería ya que según el la política de trabajo de esta arquitectura necesita ciertos datos.

Algunas de las funciones principales son:[11]

- **int MPI_Init(int *argc, char **argv)**
  Los programas MPI tienen que ser inicializados con esta función.

- **int MPI_Finalize(void)**
  Para finalizar los programas, cuando termina la ejecución en MPI.

- **int MPI_Comm_rank (MPI_Comm comm, int *rank);**
  Esta función devuelve el PID del identificador del proceso.

- **int MPI_Comm_size (MPI_Comm comm, int *size);**
  Devuelve el número de procesos que están participando en la ejecución.

La comunicación se realiza punto a punto, es decir el emisor se espera a que la comunicación se produzca.

- **int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm);**
  Con esta función se envía un buffer de datos a un destino determinado, dentro de los procesos que se están ejecutando.

- **int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status);**
  Con esta función se recibe un buffer de datos, de un origen dentro de los procesos que se están ejecutando.

- **int MPI_Pack(void *data, int count, MPI_Datatype datatype, void *buff, int size, int count, MPI_Comm comm.);**
  Esta función empaqueta datos en un buffer continuo en memoria para ser enviados como un mensaje único.

- **int MPI_Unpack(void *buff, int size, int index, void *dataOut, int count, MPI_Datatype datatype, MPI_Comm comm.);**
  Esta función desempaqueta los datos recibidos en un mensaje.

Otras funciones utilizadas en la implementación del programa son:

- **MPI Wtime()**

Con esta función se puede calcular el tiempo de ejecución y de comunicación del programa.
Para calcular el tiempo de ejecución se ejecuta esta instrucción al inicio y al final del programa, una vez esté este finalizado se realiza la resta del tiempo final menos el tiempo inicial y se obtiene el tiempo de ejecución.

Podría calcularse el tiempo de comunicación de la siguiente forma:

Cuando se realizara un envío se calcularía el tiempo inicial (tini), y cuando se realizara la recepción de este envío se calcularía el tiempo final (tfin).
Cada vez que se realizara un envío habría que calcular su correspondiente tiempo inicial y tiempo final.
El tiempo final y tiempo inicial se calculará uno por cada nodo que tengamos trabajando como esclavo.
En la implementación del programa para calcular los tiempos iniciales y finales de cada nodo se pueden ir acumulando en una variable.
De esta forma al finalizar la ejecución, se restarán los tiempos finales menos los iniciales.
Ej:
tfin1 - tini1
tfin2 - tini2

Posteriormente se sumarán los resultados obtenidos de cada resta, y el resultado de esta suma es lo que constituirá el tiempo de comunicación.

Tiempo de comunicación= (tfin1-tini1)+(tfin2-tini2)+(tfin3-tini3)+…

Para poder llegar al código que se ha utilizado para realizar las pruebas, se han ido realizando diferentes versiones del mismo código cada una de ellas más completa que la anterior, hasta que finalmente se ha logrado conseguir la versión actual.

A continuación se describirá el funcionamiento del programa y las estructuras de datos utilizadas:

El programa realizado consta de dos programas implementados por separado: el código fuente perteneciente al maestro (*master*), y el código fuente perteneciente al esclavo (*slave*).
A continuación se realizará una explicación detallada de cada uno de los códigos fuente facilitando así la comprensión del programa.

Código fuente del maestro:

Este programa que recibe el nombre de *master* recibe como parámetros de entrada:

- El número de fotogramas (con formato xml) comprimidos (.bz2) que en total hay que enviar y posteriormente recibir.
- El número de fotogramas que como máximo deben ir empaquetados en cada envío.
- La ruta en la que se encuentran los fotogramas.

Una de las características importantes que ha influido en la forma de codificación del programa es que, como se ha expuesto con anterioridad, cada fotograma posee su propio fichero .xml, esto nos lo proporciona blender y el plugin de yafaray, que son las herramientas utilizadas para generar las especificaciones xml de cada uno de los fotogramas.

El programa se encarga de lo siguiente:

- Inicializa las variables para poder realizar el paso de mensajes con el esclavo

- Posteriormente se encarga de preparar el buffer que va a enviar al esclavo, esto se realiza dentro de una función que se denomina “prepararenvio”.
  Este subprograma recibe como parámetros de entrada el número total de ficheros xml que hay que enviar para finalizar el programa, el número de xml que debe de enviar en cada paquete, la ruta en la que se encuentran los ficheros .xml y otras variables que se encargan de controlar los envíos realizados.
  El cometido de esta función es tanto calcular la longitud del buffer que se va a enviar al esclavo como prepararlo.
  Por lo tanto lo que esta función devuelve es un array de caracteres que contiene:
  - El número de xml que van en el buffer
  - Un identificador de cada fichero para poder seguir un orden
  - El número que indica el tamaño del fichero xml
  - El contenido del fichero xml

  Los tres últimos puntos se repiten por cada fichero xml.

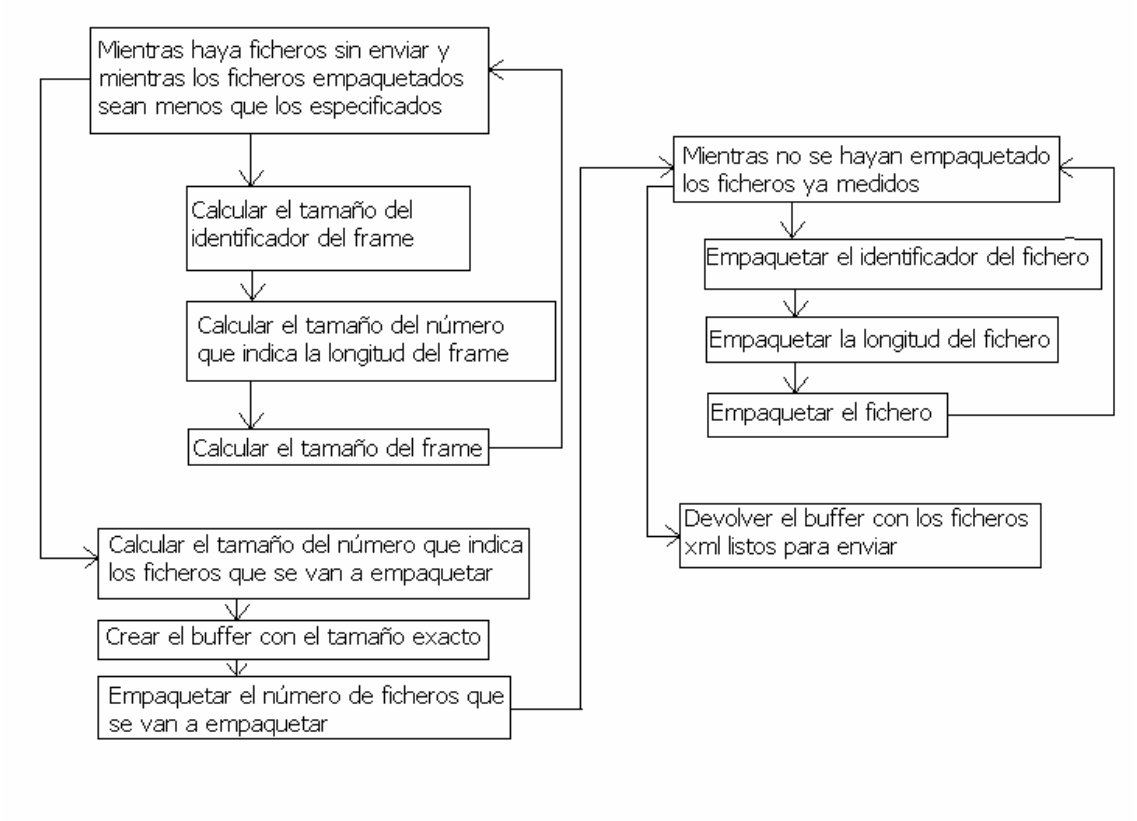

Figura 4: Esquema que explica la función "prepararenvio"

- Una vez realizado el paquete de envío el maestro se encarga de realizar el envío al esclavo correspondiente.

- Tras haber realizado el maestro los envíos iniciales, es decir un envío a cada uno de los esclavos de los que dispongamos, éste entra en una fase de espera, de cual saldrá en cuanto reciba un mensaje de algún esclavo.

- Según va recibiendo mensajes se encarga de sacar del buffer recibido cada uno de los fotogramas comprimidos y depositarlos en una carpeta, la cual se denomina "Resultados".

- Cada vez que el maestro recibe un mensaje de un esclavo, si todavía quedan ficheros xml que no han sido enviados, realiza de nuevo la tarea de "prepararenvio" haciendo uso de la función anteriormente descrita y de nuevo espera a posteriores recepciones.

Si por el contrario ya no dispone de más ficheros xml que enviar se encarga de enviar otro tipo de mensaje con el cual comunica al esclavo el fin de su ejecución.

- La ejecución del maestro acaba en el momento en el que ha recibido tantos fotogramas como ficheros xml disponía en el inicio.

La política de distribución de trabajo que sigue el maestro implementado es bajo demanda, ya que primeramente se encarga de realizar un envío a cada esclavo, pero una vez hecha esta tanda de envíos, sólo prepara los paquetes para enviarlos cuando algún esclavo queda libre.

Código fuente del esclavo:

Este programa que recibe el nombre de *slave* recibe como parámetros de entrada, el número de imágenes que van en cada envío, es decir la carga de trabajo.

Este programa se encarga de realizar lo siguiente:

-Inicializa las variables para poder realizar el paso de mensajes con el *master*.

-No continúa su ejecución hasta que no recibe un mensaje por parte del maestro.

-Puede recibir dos tipos de mensajes:

- Uno de ellos es para indicar el fin de su ejecución
- El otro es el paquete con los fotogramas que debe renderizar.
  En caso de recibir este tipo de mensaje, el esclavo se encarga de desempaquetar uno por uno cada uno de los ficheros xml, descomprimirlos y renderizarlos, esto consiste en este caso en obtener el fotograma correspondiente al fichero xml recibido.
  Posteriormente, se encarga de volver a comprimir y empaquetar de nuevo todos los fotogramas para enviárselos de vuelta al maestro.

La estructura del buffer de envío es la misma que el maestro utiliza para comunicarse con el esclavo, es decir primero empaquetará el número de fotogramas que van en el paquete posteriormente un identificador del fotograma, el tamaño, y el fotograma, estos tres últimos datos se repiten dependiendo de los fotogramas empaquetados.

Después de haber realizado el envío al maestro de nuevo el esclavo se queda esperando para volver a recibir otro mensaje.

Por lo tanto el fin de la ejecución del esclavo se producirá cuando este reciba un mensaje del maestro en el cual se lo indica.

El siguiente pseudocódigo explica el comportamiento del maestro y del esclavo.

El código del maestro realiza lo siguiente:

```
Begin
    Para i=1 hasta  Número_de esclavos
        paquete=prepararenvio(..);
        enviar (paquete);
    fin Para;

    Mientras (no se han recibido todos los fotogramas enviados) hacer
        Esperar la llegada de un mensaje;
        Recepción del mensaje;
        Si(hay fotogramas sin enviar)
            paquete=prepararenvio(..);
            enviar (paquete);
        sino
            enviar(fin de ejecución para el esclavo);
    fin Mientras;
End;
```

El código del esclavo realiza lo siguiente:

```
Begin
        Mientras (el maestro no indique el fin de la ejecución) hecer
            Esperar la llegada de un mensaje;
            Recepción del mensaje;
            Si(contiene ficheros xml)
                Renderizarlos;
                Enviar al maestro los ficheros xml renderizados;
            Sino
                Fin=true;
        Fin Mientras;
    End;
```

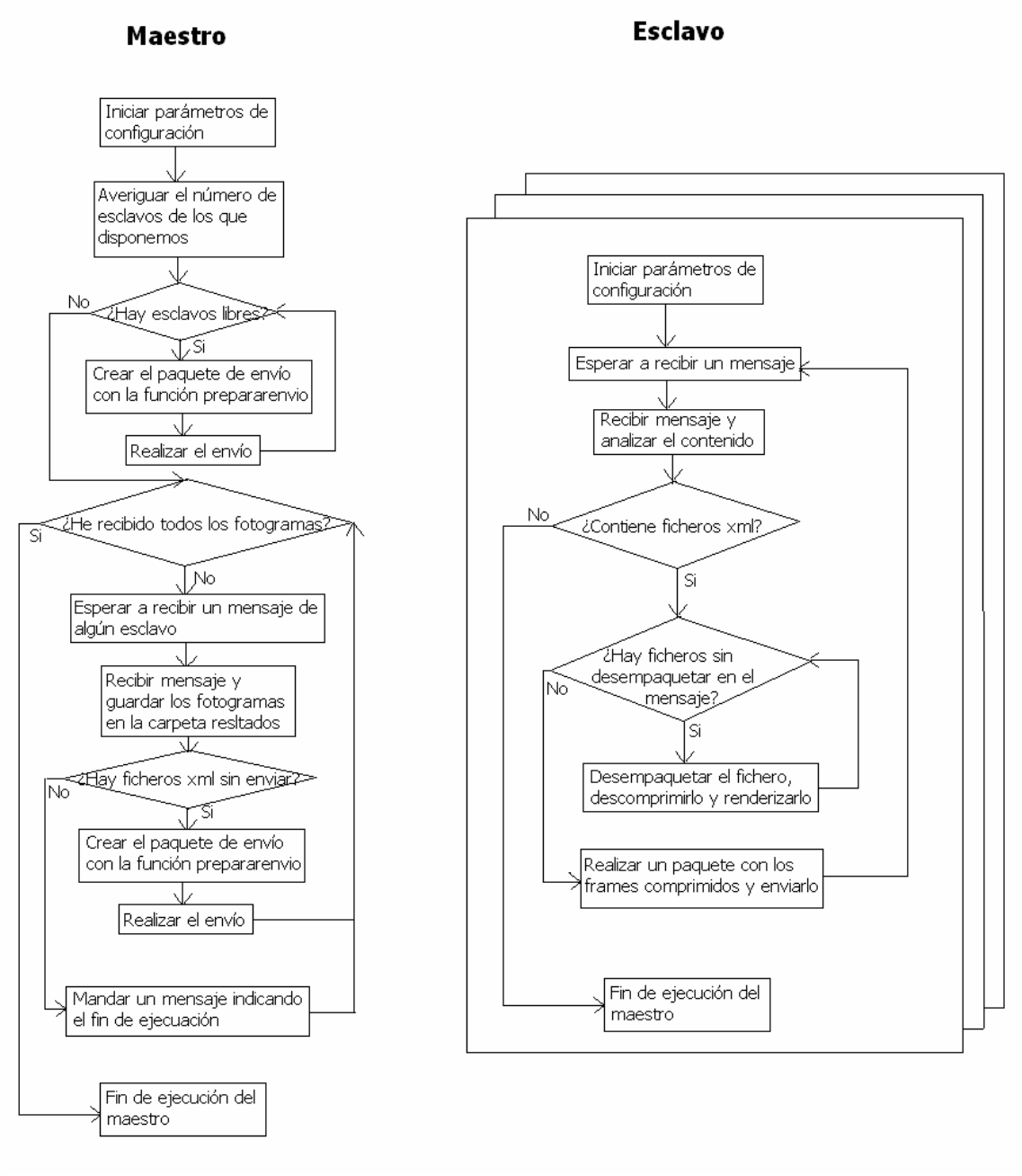

Figura 5: Diagrama de ejecución del programa de reparto equilibrado de trabajo

4. Resultados experimentales

4.1. Entorno de trabajo

Para realizar las pruebas se han utilizado dos arquitecturas diferentes:
Un multiprocesador simétrico de memoria compartida (Nemea) y un *cluster* (Altamira).

4.1.1. Multiprocesador simétrico de memoria compartida (Nemea)

Nemea es un multiprocesador simétrico de memoria compartida, es un SGI PRISM con 16 procesadores INTEL *Itanium*2, 1500 Mhz y 32 GB de RAM.
Es un tipo de arquitectura en la que varios procesadores comparten la misma memoria central.
Todos los procesadores pueden realizar cualquier tarea y acceder a memoria en cualquier momento, ya que no hay ninguna jerarquía entre ellos. [5]
Para poder compilar el código paralelo en Nemea se necesitan las librerías LAM/MPI.

Al realizar las primeras pruebas en esta arquitectura se observa que yafaray dispone de un mecanismo interno de reparto, es decir además del reparto de trabajo del que se encarga el programa implementado, yafaray se encarga de realizar otro reparto de trabajo internamente con los distintos procesadores.
Esto impedía realizar las mediciones fiables sobre la ejecución del programa.
Para impedir que yafaray realice ese reparto internamente, el propio comando que se encarga de renderizar dispone de una opción que deshabilita esta función.
La otra manera de solucionarlo, que es la utilizada en este caso, es modificar el fichero que se ejecuta obligando a yafaray a ejecutarse en el procesador que se le indica, de esta forma el único reparto que se realiza es el gestionado por el programa implementado.
Después de introducir este cambio el código del fichero que se ejecuta queda modificado de la siguiente forma:

n0 taskset –c nºprocesador programa parámetros de entrada

El Nº del procesador es el identificador que tiene el procesador en el cual queremos ejecutar la instancia de nuestro programa.

Por último mencionar que para obtener los datos que se han ido calculando durante la ejecución del programa, se necesita redireccionar la salida estándar, esto se realiza al invocar el comando con el que lanza la ejecución.

4.1.2. *Cluster* (Altamira)

Altamira es un *cluster* situado en la Universidad de Cantabria, perteneciente a la RES (Red Española de Supercomputación), se formó con una serie de nodos pertenecientes al supercomputador MareNostrum.
Este supercomputador posee 256 nodos biprocesadores PC970Fx Altivez Supported a 2200 Mhz y 4 Gbytes de RAM y un rendimiento de 2994,04 GFlops. [12]

Un *cluster* es un conjunto de computadores, unidas por una red, que se comportan como si fuesen una única máquina. Esto nos proporciona una alta potencia computacional.[5]

La implementación de MPI disponible en esta arquitectura es MPICH2.

Para realizar las pruebas ha sido necesario un trabajo de adaptación debido a la política de ejecución de este *cluster*, para efectuar las ejecuciones se necesita crear un programa que se encargue de manejar al maestro y al esclavo desde un mismo código dado que no se permite lanzar dos ejecutables como era el caso del multiprocesador simétrico de memoria compartida.
Este programa se encarga de ejecutar las funciones de inicio y finalización de MPI, a la vez que se encarga de realizar la llamada al programa del maestro o del esclavo según corresponda.

Para ejecutar el programa en el *cluster*, hay que crear un script que contenga entre otros datos como el nombre de los ficheros de salida, el tiempo máximo que este programa

puede tardar en ejecutarse, la ruta en la que se encuentra el código del programa o los datos que mi programa recibe como parámetros de entrada.
En la misma línea en la que se indica la ruta de donde se encuentra el código del programa se redireccionan todas las salidas por pantalla a un fichero determinado.

Cuando se ejecuta el programa, se manda la ejecución a una cola de procesos de la que dispone el *cluster*, esta política de ejecución se debe a la demanda que sufre esta máquina.
Por lo tanto los procesos se irán ejecutando cuando la arquitectura disponga de los recursos que requiera cada proceso y según el orden de preferencia del que dispongan dentro de la cola de espera.

4.2. Resultados

Para poder realizar un estudio acerca de la eficiencia y la escalabilidad del programa implementado, la medida que se ha considerado relevante es el Tiempo de ejecución, con este dato obtenemos el tiempo que tarda en ejecutarse el programa.
Este dato se calcula de forma secuencial, y en paralelo utilizando distinto número de esclavos.

La explicación de cómo calcular este tiempo aparece en el apartado 2.2.
Con este tiempo medido, se pueden realizar otros cálculos de interés, como por ejemplo calcular el *Speedup*, que compara la ejecución de un programa con la ejecución del mismo programa pero habiéndole introducido mejoras y la Eficiencia que mide el uso óptimo de los recursos disponibles.

El *Speedup* se calcula de la siguiente forma: dividiendo el tiempo de ejecución del programa entre el tiempo de ejecución del programa con n procesadores (este número n varía según la prueba), es decir habiéndole introducido una mejora.

$$Speedup = t_1 / t_n$$

La Eficiencia se calcula dividiendo el *Speedup* entre el número de procesadores.

Eficiencia = *Speedup* / n

A continuación se mostrarán una serie de tablas y gráficas que interpretan lo anteriormente citado.
Todas las unidades de tiempo están en segundos.

4.2.1. Experimentos realizados en el multiprocesador

Las pruebas realizadas en esta arquitectura son las siguientes:

- Ejecución del programa de forma secuencial

- Ejecución del programa en paralelo

    - Utilizando 2 nodos
        - Variando la carga de trabajo: 8, 16, 32, 63, 125, 250 fotogramas en cada mensaje.

    - Utilizando 4 nodos
        - Variando la carga de trabajo: 8, 16, 32, 63, 125 fotogramas en cada mensaje.

    - Utilizando 8 nodos
        - Variando la carga de trabajo: 8, 16, 32, 63 fotogramas en cada mensaje.

    - Utilizando 16 nodos
        - Variando la carga de trabajo: 8, 16, 32 fotogramas en cada mensaje.

Las pruebas se han realizado con distinto número de nodos y variando la carga de trabajo en cada caso. Se han utilizado 500 fotogramas para realizar las pruebas en esta arquitectura.

El tiempo de ejecución calculado de forma secuencial es el siguiente:

Tiempo de ejecución de forma secuencial = 125262,7793 segundos

La siguiente tabla muestra la relación entre el número de nodos y el tamaño de paquete utilizado, mostrando en cada caso el tiempo de ejecución:

Tamaño paquete/nodos	2	4	8	16
250	63979,7447			
125	62624,2215	32285,4881		
63	63120,6804	31589,7631	16293,7399	
32	64044,0576	32057,5615	16044,9726	8390,5628
16	64023,0741	32045,8151	16045,3225	8141,7712
8	63028,5555	32031,1206	16021,1372	8051,7123

Tabla 1: Tiempo de ejecución

A continuación se muestra la gráfica correspondiente a la tabla anterior:

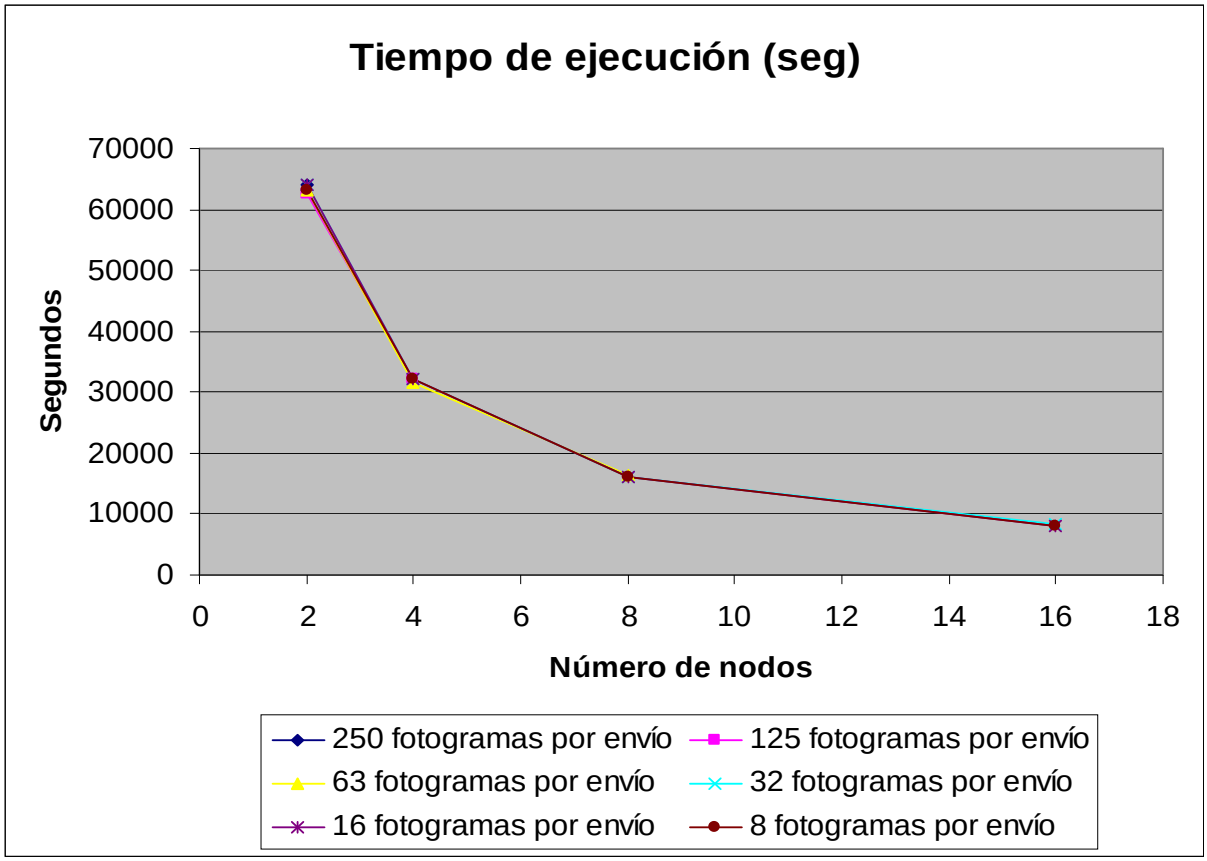

Figura 6: Tiempos de ejecución

Observando esta gráfica una de las conclusiones que se pueden obtener, es que si no se varía el número de nodos en la ejecución del programa, el tiempo de ejecución disminuye cuanto menor sea el tamaño de paquete.
Y en cuanto al número de nodos contra más nodos haya trabajando, menor es el tiempo de ejecución.

La siguiente tabla muestra la relación entre el número de nodos y el tamaño de paquete utilizado, mostrando en cada caso el *Speedup* calculado.

Tamaño paquete/nodos	2	4	8	16
250	1,9579			
125	2,0002	3,8798		
63	1,9845	3,9653	7,6878	
32	1,9559	3,9074	7,8070	14,9290
16	1,9565	3,9089	7,8068	15,3852
8	1,9874	3,9107	7,8186	15,5573

Tabla 2: *Speedup*

A continuación se muestra la gráfica correspondiente a la tabla anterior:

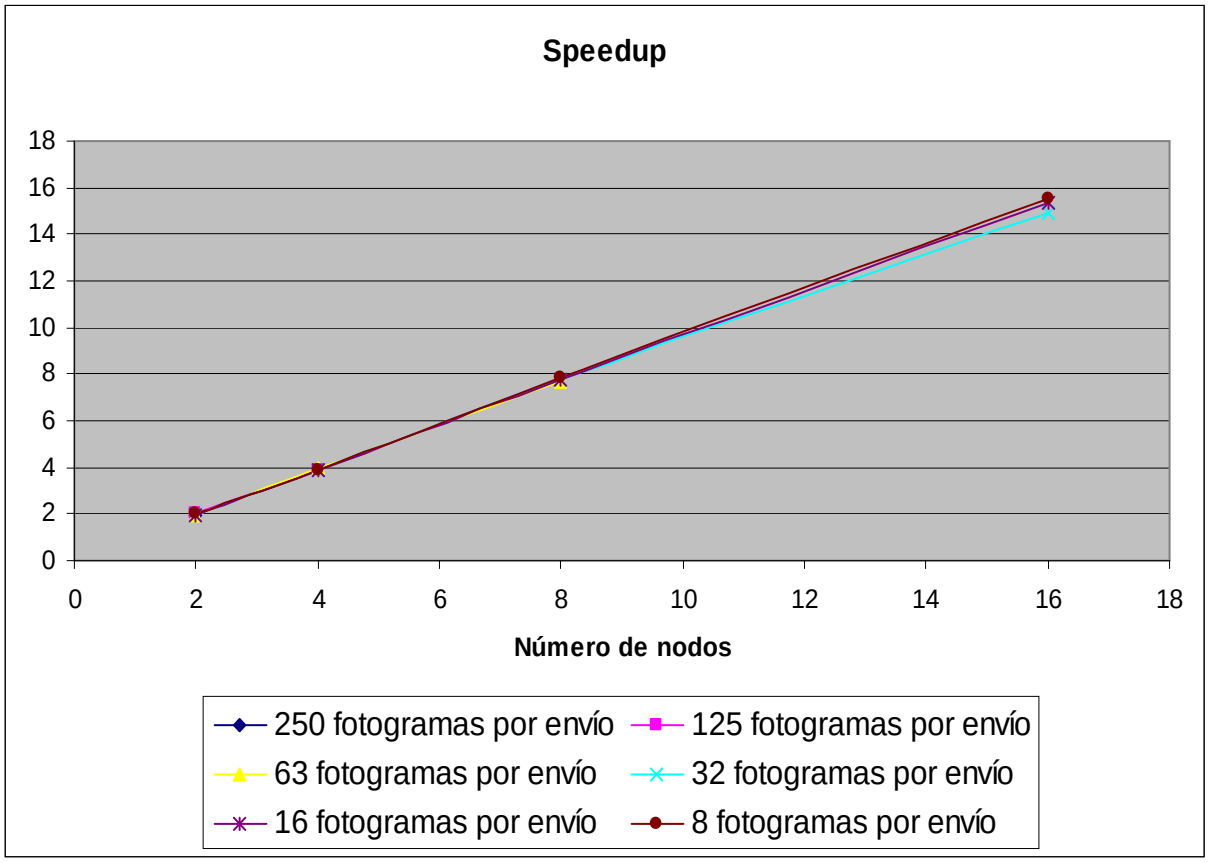

Figura 7: *Speedup*

La gráfica anterior muestra un comportamiento prácticamente igual que el ideal, este sería una gráfica que represente un crecimiento lineal.

La siguiente tabla muestra la relación entre el número de nodos y el tamaño de paquete utilizado, mostrando en cada caso la Eficiencia calculada según la fórmula citada anteriormente.

Tamaño paquete/nodos	2	4	8	16
250	0,9789			
125	1,0001	0,9700		
63	0,9922	0,9913	0,9610	
32	0,9779	0,9769	0,9759	0,9331
16	0,9783	0,9772	0,9759	0,9616
8	0,9937	0,9777	0,9773	0,9723

Tabla 3: Eficiencia

A continuación se muestra la gráfica correspondiente a la tabla anterior:

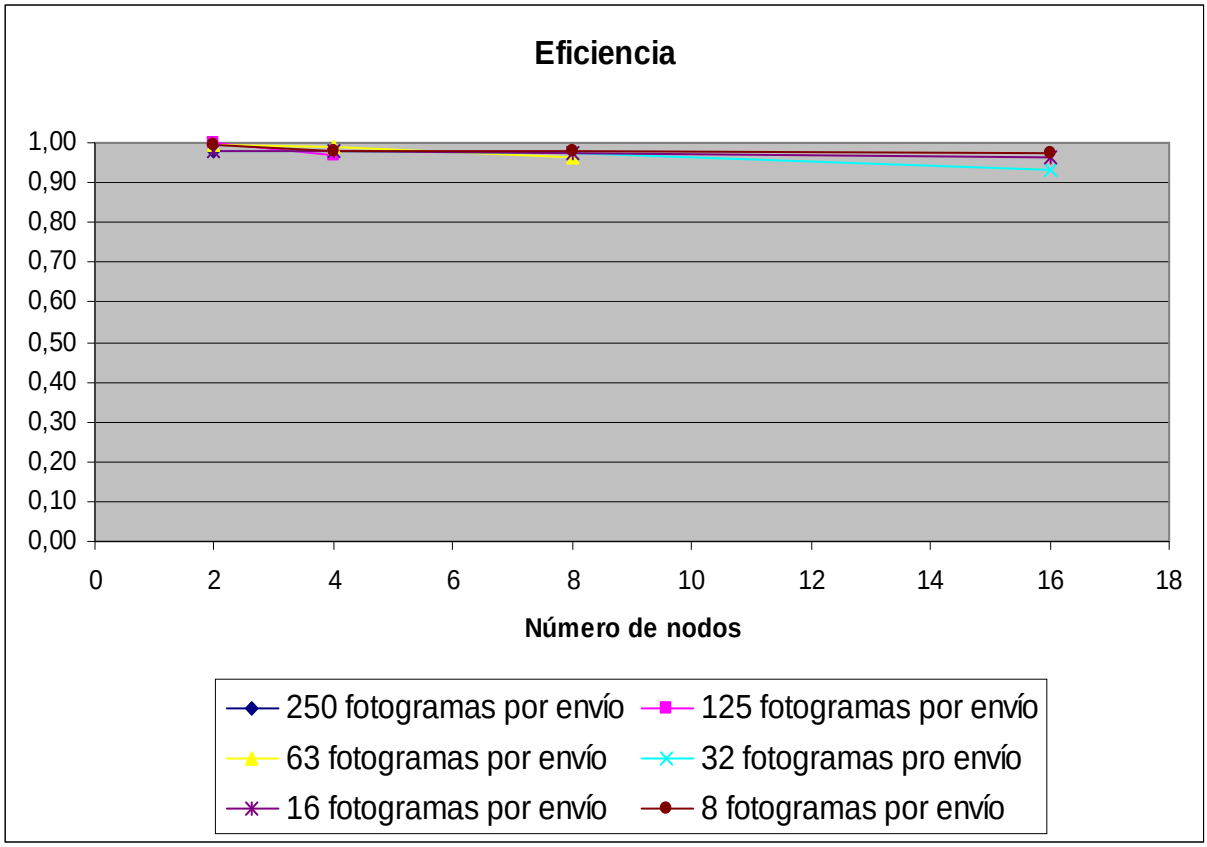

Figura 8: Eficiencia

Observando la gráfica anterior podemos decir que el algoritmo implementado es bastante eficiente, ya que los resultados obtenidos nos proporcionan una eficiencia en la mayoría de los casos muy cercana a uno.

4.2.2. Experimentos realizados en el *cluster*

Las pruebas realizadas en esta arquitectura son las siguientes:

- Ejecución del programa de forma secuencial

- Ejecución del programa en paralelo

    - Utilizando 2 nodos
        - La carga de trabajo utilizada en este caso es: 1000 fotogramas en cada mensaje.

    - Utilizando 4 nodos
        - La carga de trabajo utilizada en este caso es: 500 fotogramas en cada mensaje.

    - Utilizando 8 nodos
        - La carga de trabajo utilizada en este caso es: 250 fotogramas en cada mensaje.

    - Utilizando 16 nodos
        - La carga de trabajo utilizada en este caso es: 125 fotogramas en cada mensaje.

    - Utilizando 32 nodos
        - La carga de trabajo utilizada en este caso es: 63 fotogramas en cada mensaje.

    - Utilizando 64 nodos
        - La carga de trabajo utilizada en este caso es: 32 fotogramas en cada mensaje.

    - Utilizando 128 nodos
        - La carga de trabajo utilizada en este caso es: 16 fotogramas en cada mensaje.

    - Utilizando 256 nodos
        - La carga de trabajo utilizada en este caso es: 8 fotogramas en cada mensaje.

En este caso sólo ha realizado por cada número de nodo, una prueba utilizando la máxima carga de trabajo posible para cada configuración. Se han utilizado 2000 fotogramas para realizar las pruebas en esta arquitectura.

El tiempo de ejecución calculado de forma secuencial es el siguiente:

Tiempo de ejecución de forma secuencial = 353600 segundos

La siguiente tabla muestra la relación entre el número de nodos y el tamaño de paquete utilizado, mostrando en cada caso el tiempo de ejecución:

Nodos	4	8	16	32	64	128
	90982,3490	45865,3114	23060,2525	11777,2315	6004,8684	3034,8025

Tabla 4: Tiempo de ejecución

A continuación se muestra la gráfica correspondiente a la tabla anterior:

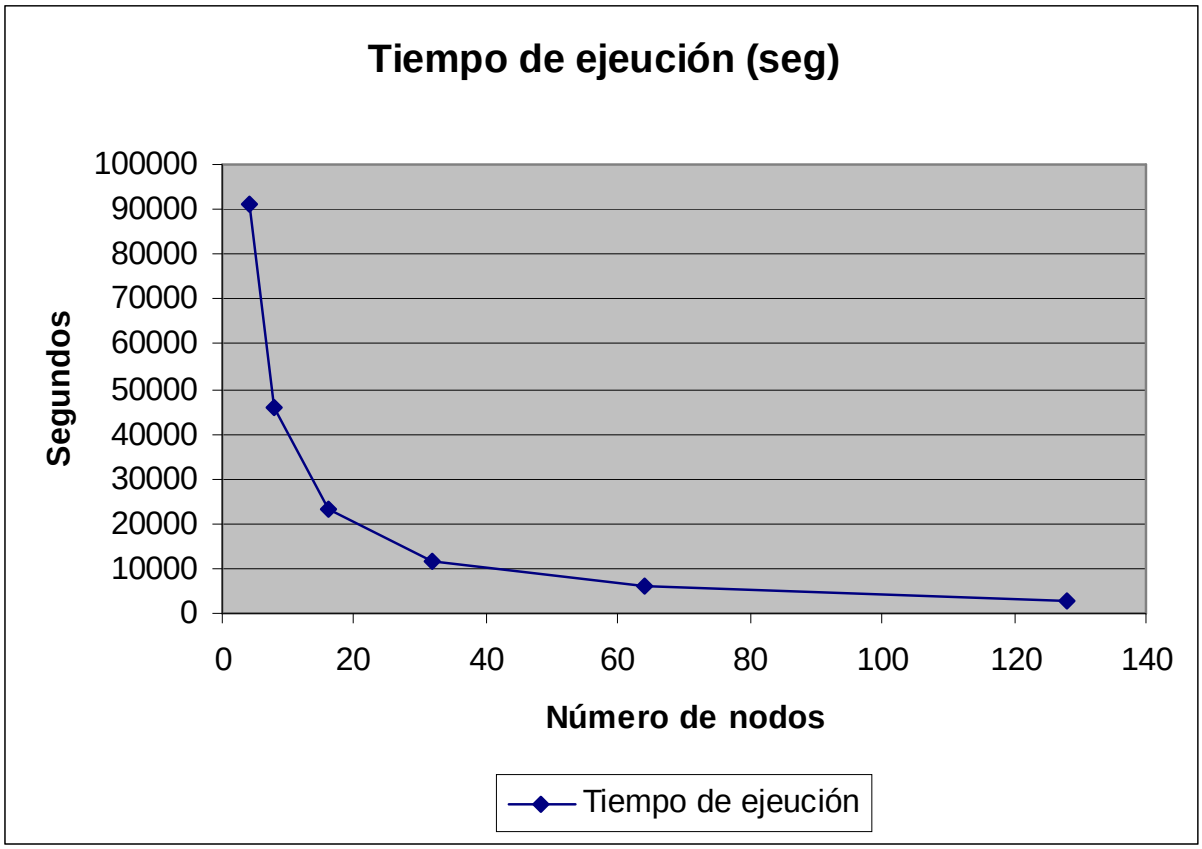

Figura 9: Tiempo de ejecución

Observando la gráfica anterior se comprueba que contra más nodos se utilicen en la ejecución, menor será el tiempo empleado.

La siguiente tabla muestra la relación entre el número de nodos y el tamaño de paquete utilizado, mostrando en cada caso el *Speedup* calculado según la fórmula citada anteriormente.

Nodos	4	8	16	32	64	128
	3,8865	7,7095	15,3337	30,0240	58,8856	116,5150

Tabla 5: *Speedup*

A continuación se muestra la gráfica correspondiente a la tabla anterior:

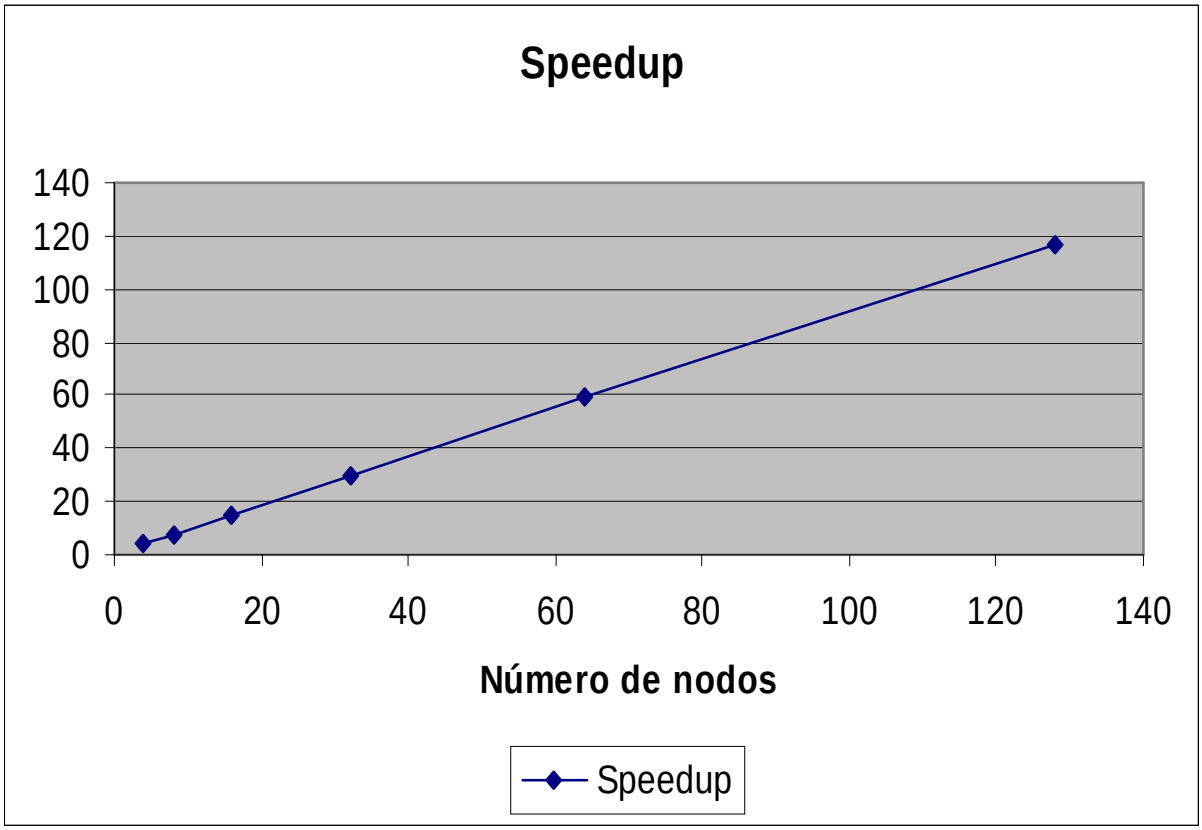

Figura 10: *Speedup*

Al igual que sucede en las pruebas realizadas en el multipocesador, la gráfica que representa el *speedup*, muestra un comportamiento prácticamente igual que el ideal, este sería una gráfica que represente un crecimiento lineal.

La siguiente tabla muestra la relación entre el número de nodos y el tamaño de paquete utilizado, mostrando en cada caso la Eficiencia calculada según la fórmula citada anteriormente.

Nodos	4	8	16	32	64	128
	0,9716	0,9637	0,9584	0,9383	0,9201	0,9103

Tabla 6: Eficiencia

A continuación se muestra la gráfica correspondiente a la tabla anterior:

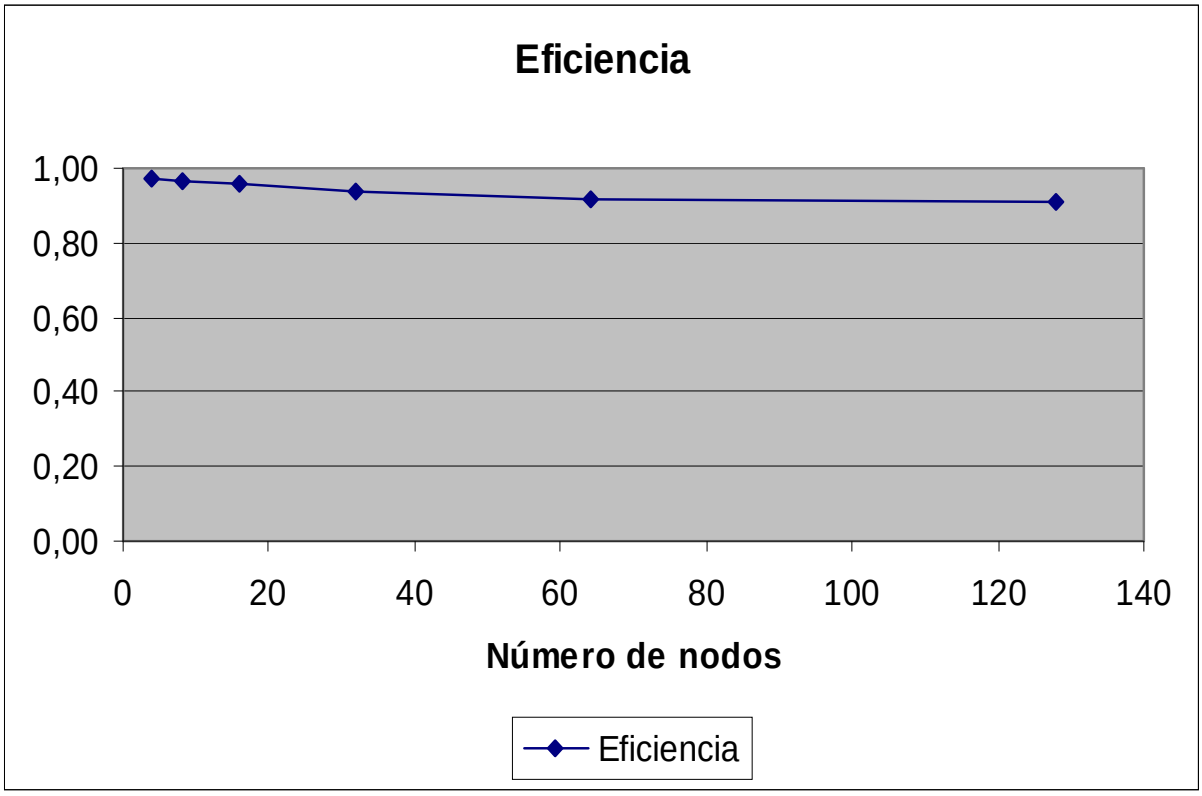

Figura 11: Eficiencia

Con los resultados de la gráfica anterior, se verifica la eficiencia del algoritmo al ser ejecutado en un cluster. Ya que en todos los casos la eficiencia es superior al 90%.

5. Conclusiones y trabajos futuros

5.1. Conclusiones

El objetivo principal de este proyecto, el cual consistía en implementar un algoritmo que fuera capaz de paralelilar la taea de renderizado y redujera el tiempo de ejecución se ha conseguido de forma satisfactoria, esto lo demuestran los resultados obtenidos.

Para poder superar el objetivo principal de este proyecto, previamente se han ido consiguiendo de forma individual las siguientes metas, formando en su conjunto la totalidad del proyecto:

- Aprendizaje acerca de blender y yafaray las herramientas utilizadas para poder conseguir los ficheros xml de cada uno de los fotogramas empleados en la realización de las pruebas y también las encargadas del proceso de renderizado.

- Se ha implementado un algoritmo que se encarga de paralelizar la tarea de renderizado de una serie de fotogramas. Este programa podría ser ejecutado en cualquier máquina paralela sin importar el número de nodos que tuviera a su disposición.

- Se han obtenido una serie de datos a partir de pruebas realizadas en diferentes arquitecturas, con el fin de tener unos datos objetivos con los que poder valorar la eficiencia del programa.

También hay que mencionar que para realizar las pruebas se han utilizado dos librerías diferentes cada una de ellas utilizada en una arquitectura, LAM/MPI y MPICH2.
Por lo tanto la elaboración de este PFC también incluye el aprendizaje acerca de estas dos librerías.

Se puede afirmar que los resultados obtenidos son favorables, que gracias a la implementación de un algoritmo que utiliza las características que le ofrecen las arquitecturas que permiten trabajar en paralelo, se ha conseguido reducir el tiempo de ejecución respecto a la ejecución secuencial.

Uno de los aspectos que se puede concluir es la escalabilidad del algoritmo en base a los resultados obtenidos.

Otra conclusión que se ha obtenido a partir de los resultados que se proporcionan en el capítulo 4, concretamente en las pruebas realizadas en el multiprocesador de memoria compartida es la siguiente: observando los tiempos de ejecución se aprecia que disminuyen a medida que disminuye el tamaño de paquete (carga de trabajo en cada mensaje).

A primera vista esto sería un dato erróneo, ya que si el tamaño de paquete es menor, se envían más mensajes y esto implica un número mayor de envíos y recepciones es decir un número mayor de comunicaciones entre el maestro y el esclavo.

La razón de que esto no suceda así y de que el tiempo de ejecución sea menor cuanto menor sea el tamaño de paquete es porque el maestro emplea bastante tiempo en preparar el paquete de envío, este tiempo aumenta dependiendo del tamaño de paquete; por otro lado, cuando aumenta su carga de trabajo, también lo hará el tiempo de preparación de dicho paquete de envío.

El problema es que mientras el maestro prepara los paquetes de envío deja bloqueados, en espera, a los esclavos, que si ya han acabado su tarea, no pueden seguir trabajando hasta que el maestro les atienda.

Es por eso por lo que cuanto mayor sea el paquete de trabajo más tiempo puede que estén bloqueados los esclavos, sin embargo si el tamaño del paquete de trabajo es pequeño, el maestro tarda poco tiempo en prepararlo y por lo tanto atiende antes a los esclavos, apenas tienen tiempo de quedarse bloqueados.

Observando las gráficas (Figuras 8 y 11) que representan la eficiencia en las arquitecturas empleadas en la realización de las pruebas, vemos que los resultados obtenidos son bastante buenos ya que prácticamente todas las eficiencias están muy cercanas a 1, concretamente en todos los casos la eficiencia supera el 90%, este es otro indicio de que el programa ha cumplido satisfactoriamente las expectativas que se tenían al inicio del proyecto.

Como conclusiones finales se puede decir:

Una de las definiciones más sencillas de escalabilidad es que el funcionamiento de un ordenador aumente linealmente con respecto al número de procesadores usados para una aplicación dada.
En el caso ideal del funcionamiento de un ordenador debe ser linealmente escalable con el aumento del número de procesadores empleados en la ejecución del algoritmo.

Y tras estudiar los resultados obtenidos podemos decir que esta condición se verifica al comprobar cómo van aumentando los resultados de forma lineal dependiendo del número de procesadores, por lo tanto, el programa implementado posee una alta escalabilidad.

La eficiencia es una indicación del grado real de la velocidad de funcionamiento alcanzada de un ordenador comparado con el valor máximo.

Esta otra definición confirma la eficiencia del algoritmo implementado ya que si tomamos el valor obtenido en el tiempo de ejecución de forma secuencial y lo comparamos con cualquier otro valor obtenido de la ejecución con varios procesadores, se puede observar la diferencia al disminuir los tiempos.

5.2. Líneas Futuras

La implementación del programa ha servido para facilitar el trabajo a la hora de renderizar imágenes. Es una implementación útil ya que permite la ejecución en paralelo del programa permitiendo reducir el tiempo de ejecución intentando usar de la mejor forma posible todos los recursos de los que se dispone, pero como todos los programas podría tener algunas mejoras.
A continuación se citan algunas ideas por las que podría continuar la investigación acerca de este proyecto:

- Explotar las capacidades de paralelismo interno de los procesadores: una de las consecuencias que actualmente presenta el programa es que el maestro invierte demasiado tiempo en preparar los paquetes de envío si envía una gran cantidad de fotogramas (*frames*) en él.
  Esta consecuencia repercute en que mientras el maestro está preparando el paquete de envío no puede atender a posibles esclavos que ya hayan acabado con su tarea, por lo tanto deja a los esclavos bloqueados.
  Este hecho nos impide aprovechar al máximo todos los recursos de los que disponemos, ya que no aprovechamos al máximo todos los esclavos, dado que una parte del tiempo de su ejecución están bloqueados esperando la atención del maestro.
  Este hecho se incrementa cuanto más fotogramas queremos empaquetar en cada envío ya que aumenta el tiempo de ejecución, sin embargo con un número pequeño de fotogramas este hecho apenas repercute en el tiempo de ejecución.

  Tras comprobar esta consecuencia, una de las posibles mejoras del programa sería hacer uso de las técnicas de paralelismo interno, esto significa que se pudieran lanzar varios hilos de ejecución del maestro.
  De esta forma uno de los hilos de ejecución podría encargarse de enviar los paquetes con los fotogramas, y el otro hilo de ejecución podría recibir los

mensajes de los esclavos, de esta forma nunca tendríamos a los esclavos esperando, y reduciríamos notablemente el tiempo de ejecución ya que el trabajo del maestro estaría siendo ejecutado por varios *threads*.

- Estudiar el comportamiento de este programa en un sistema heterogéneo, para lo que podría ser necesario utilizar algorítmica de equilibrado de carga, bien estático o bien dinámico, de forma que se aprovechen al máximo las capacidades de aquellos procesadores más potentes.

- Otra idea podría ser la de tratar de reducir el tiempo empleado en las operaciones de entrada y salida. En las pruebas realizadas se ha observado que una gran cantidad de tiempo es invertido en este tipo de operaciones, por lo que puede resultar interesante utilizar técnicas que permitan emplear menos tiempo en esta fase; por ejemplo, buscando una reducción en el número de ficheros utilizados.

Bibliografía

- [1] Alan Watt, “3D Computer Graphics”, Third Edition. Editorial Addison-Wesley, 2000.

- [2] Enciclopedia libre Wikipedia. Gráficos 3D por computadora
http://es.wikipedia.org/wiki/Gr%C3%A1ficos_3D_por_computadora

- [3] Blender
http://www.blender.org/

- [4] Yafaray
http://www.yafaray.org/

- [5] Pedro de Miguel Anasagasti, “Fundamentos de los computadores”, Séptima Edición. Editorial Paraninfo, 1999.

- [6] Kai Hwang,“Advanced Architect: Parallelism, Scalability, Programmability”. McGraw-Hill Series in Computer Science, 1993.

- [7] Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar, “Introduction to Parallel Computing”, Second Edition. Editorial Addison-Wesley, 2003.

- [8] Introducción a MPI
http://informatica.uv.es/iiguia/ALP/materiales2005/2_2_introMPI.htm#lam-mpi

- [9] Paso de mensajes
http://www.sc.ehu.es/acwmialj/edumat/mpi.pdf

- [10] LAM/MPI
http://informatica.uv.es/iiguia/ALP/materiales2005/2_2_introMPI.htm

- [11] Funciones MPI
  http://telematica.cicese.mx/computo/super/cicese2000/mpi/mpi.html
- [12] Altamira
  https://unicornio.ifca.es/news/5/

- Marcos Novalbos Mendiguchía, "Paralelización de un algoritmo de segmentación de vídeo ", 2006.

- Raúl Torquemada Charle, "Generación y Visualización de Vídeos estéreo para entornos de realidad virtual", 2006.