Universidad Autónoma Metropolitana
Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Proyecto Terminal en Ingeniería en Computación

Codificación paralela de video de formato VCD(MPEG-1) a MPEG-4

Proyecto que presentan:

Araceli Morales González

Luis Adrián Ortiz Olivares

para obtener el titulo de:

Ingeniero en Computación

Asesor de Proyecto:

M. en C. Oscar Alvarado Nava

México, D.F. **Julio de 2010**

Resumen

Actualmente el tema de la codificación juega un papel muy importante en el campo de la multimedia. Esto se debe, en mayor medida al avance de la tecnología, pues cada día se demanda mayor capacidad de compresión de archivos de video con una buena calidad y así poder ser utilizados en dispositivos electrónicos de alto rendimiento.

El desarrollo de este proyecto tiene como objetivo fundamental, programar una aplicación paralelizada, que permita al usuario reducir el tiempo de codificación de un video al convertir formato MPG a formato MP4.

Para lograr el objetivo planteado, la metodología empleada está basada en las diversas técnicas de codificación del estándar MPEG, concretamente el MPEG-1 y MPEG-4, para el formato MPG y MP4 respectivamente. Estos estándares son determinantes en lo que se refiere al planteamiento y adaptación del algoritmo de codificación que se diseñó para la realización del presente proyecto.

Una vez que el algoritmo se escribió en un lenguaje de programación, el siguiente paso fue probar la aplicación para comprobar las hipótesis planteadas al inicio, para ello se utilizaron la biblioteca ffmpeg, encargada de la codificación de formato MPG a MP4 y la biblioteca de paso de mensajes OpenMPI, la cuál es un interfaz multiplataforma de alto desempeño implementada en código abierto, que de manera muy específica es utilizada en cómputo paralelo o distribuido, razón por la cual permite trabajar en un ambiente paralelizado dentro de un clúster de computadoras en cual se llevaron a cabo las pruebas.

Los resultados que se lograron fueron una optimización de tiempo, es decir se redujo el tiempo de codificación de video, comparado con pruebas realizadas en un sólo equipo de cómputo y además una optimización en el tamaño puesto que los archivos resultantes en formato MP4 son menores a los de formato MPG, todos estos resultados están representados en gráficas obtenidas al comparar y promediar los tiempos obtenidos durante las pruebas realizadas.

Finalmente, se muestran las conclusiones obtenidas al desarrollar el presente proyecto y se hacen sugerencias para su adaptación ó mejora total en proyectos futuros, al igual que se indica el uso de una interfaz que permita al usuario la ejecución de esta aplicación de manera fácil y amigable.

Agradecimientos

- A la División de Ciencias Básicas e Ingeniería de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco.

- Al Departamento de Electrónica y de Sistemas

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Motivaciones

La codificación de video es de primordial importancia para la mayoría de aplicaciones multimedia, en especial las que manejan flujos de video de alta calidad, como son los teléfonos móviles de última generación, la difusión de vídeo en Internet y la televisión de alta definición.

El video digital ofrece innumerables ventajas respecto al analógico; resolución y calidad de imagen muy superior, mayor capacidad de edición, prácticamente inmune al ruido, mayor durabilidad, etc. Sin embargo, todas estas ventajas superarían en la mayoría de los casos las posibilidades de nuestros equipos de cómputo, principalmente en lo que ha capacidad de almacenamiento se refiere.

1.2. Objetivos

El objetivo de realizar este proyecto, es el de procesar en paralelo un video para reducir el tiempo de codificación, pasando de un formato VCD (MPEG-1 ó MPG) a MPEG - 4 ó MP4.

A continuación se enlistan las tareas desarrolladas para la realización del proyecto.

1. Elegir y analizar el algoritmo de codificación de video.
2. Identificar el grado de paralelización del algoritmo de codificación de video.
3. Programación del algoritmo paralelizado con OpenMPI[1] de codificación de video, para pasar de formato VCD (MPEG-1 ó MPG) a MPEG-4 ó MP4.
4. Diseñar e implementar una interfaz gráfica para la interacción con el programa que implemente el algoritmo resultante.

5. Comparar los resultados obtenidos al ejecutar las aplicaciones secuencial y paralela.

1.3. Codificación de video

La codificación de video digital, también conocida como compresión, consiste básicamente en la reducción del número de parámetros requeridos para representar la señal, manteniendo una buena calidad de imagen. Al descomprimir, se recupera el número máximo de parámetros de la señal original en función del subconjunto de parámetros.

El proceso de codificación de video consta básicamente de tres pasos: el pre procesado de la señal de entrada, en el que básicamente se elimina el ruido. Enseguida, se realiza la conversión de la señal de video a un formato común intermedio, conocido como CIF, y finalmente se realiza la codificación en sí.

Existen diversos estándares para la codificación de video, todos ellos dirigidos a diversas aplicaciones con diferentes requerimientos de velocidad. Entre los más importantes cabe mencionar el estándar MPEG (*Motion Pictures Expert Group*) que especifica la representación codificada de video para dispositivos de almacenamiento digital. Esta representación soporta distintas formas de reproducción, como pueden ser: reproducción rápida, aleatoria, hacia atrás y congelación de imagen.

MPEG utiliza un sistema de codificación con pérdida para comprimir el flujo de datos del archivo de video, es decir, el resultado de un archivo MPEG es un archivo menor que el original y de menor calidad de imagen y de sonido, aunque con las posteriores versiones del formato, MPEG-4 por ejemplo, esta pérdida se convierte en algo casi imperceptible para el ojo y oído humanos.

Este estándar, es totalmente compatible con cualquier monitor de una computadora personal y con cualquier formato de televisión; de igual manera no especifica cómo debe hacerse la codificación anteriormente mencionada, lo que permite a los fabricantes desarrollar nuevos algoritmos de compresión, aunque todos deben ser compatibles entre sí. Un archivo MPEG se compone de tres capas: audio, video y una capa de sistema que especifica la calidad de la grabación, la sincronización entre capas y el tiempo de duración, entre otros datos.

Los tipos de MPEG disponibles en la actualidad son[2]:

- **MPEG-1**. La primera versión de este estándar, se hizo para CD-ROM audio/-video (VCD).
- **MPEG-2**. Estándar de transmisión de archivos de video digital de alta calidad a pantalla completa.

- **MPEG-3**. Desarrollado para televisión digital de alta resolución, pero que fue reemplazado por MPEG-2.
- **MPEG-4**. Es un estándar desarrollado para Internet y videoconferencia, así como para nuevas tecnologías de video y transmisiones a través de entornos GSM (dispositivos móviles). Su calidad está muy cercana al DVD, pero utiliza un factor de compresión mayor, por lo que sus archivos están más comprimidos que en el formato DVD.
- **MPEG-7**. Este estándar aprobado recientemente por ISO, en un futuro muy próximo será el formato más utilizado para Internet y nuevas tecnologías del tipo televisión interactiva. No sólo incluye codificación de imagen y sonido, sino también datos en formato XML.

Capítulo 2

Algoritmo de Codificación

2.1. Elección del algoritmo

El objetivo de las técnicas de codificación es la reducción de la tasa binaria, analizando tipos de redundancia y codificando la mínima información necesaria. Los resultados de la codificación dependen de la cantidad de redundancia contenida en la imagen y de la técnica de codificación aplicada.

Para llevar a cabo la codificación de video de un formato a otro, como se ha venido mencionando, nos apoyaremos de la biblioteca **ffmpeg**[3] la cual es una colección de software libre que permite grabar, convertir y hacer un *streaming* de audio y video. Esta biblioteca de software se basa en el estándar MPEG, por lo que nos permitirá trabajar con cualquiera de los formatos que lo componen.

A grandes rasgos, el formato MPEG tiene una filosofía muy básica, en un video digital, cada imagen captada por la cámara se llama fotograma. Los primeros formatos M-JPEG, se aprovechaban de esta tecnología, pero comprimiendo la imagen de cada fotograma mediante el algoritmo de compresión para imágenes fijas JPEG.

Sin embargo, el formato MPEG es mucho mejor, ya que toma fotograma por fotograma y encuentra un fotograma de referencia que será almacenado completo, comparando los fotogramas anteriores y posteriores, y sólo almacenando las diferencias que existan entre estos fotogramas y el de referencia, así no se tiene la necesidad de guardar todos los fotogramas de video que se deben visualizar; la mayor parte de la información perdida no es perceptible para el ojo humano. El algoritmo de compresión y descompresión de los distintos estándares de MPEG, necesitan un códec (codificador/decodificador) hardware para MPEG, para que en conjunto con el software no produzca cortes y saltos al archivo final.

Las técnicas de codificación MPEG son de naturaleza estadística. Las secuencias

de vídeo contienen normalmente redundancia estadística en las dimensiones espacial y temporal. La propiedad estadística en la que se basa la compresión MPEG es la correlación entre *pixeles*. Se asume que la magnitud de un *pixel* determinado puede ser prevista mediante *pixeles* cercanos del mismo cuadro (correlación espacial), o los *pixeles* de cuadros cercanos (correlación temporal). Intuitivamente se puede apreciar que en los cambios abruptos de escena, la correlación entre cuadros adyacentes es pequeña o casi nula, en ese caso es mejor usar técnicas de compresión basadas en la correlación espacial en el mismo cuadro.

Los algoritmos de compresión MPEG usan técnicas de codificación DCT (transformada discreta del coseno) sobre bloques de 8*8 *pixeles* para explotar la correlación espacial. Sin embargo, cuando la correlación temporal es alta, en imágenes sucesivas de similar contenido, es preferible usar técnicas de predicción temporal (DPCM: codificación por modulación diferencial de pulsos). En la codificación MPEG se usa una combinación de ambas técnicas para conseguir una alta compresión de los datos.

2.2. Técnica de compresión MPEG-1

La técnica básica de compresión está basada en una estructura macro bloque[1], compensada en movimiento, y el relleno condicional de dichos macro bloques. El primer fotograma (cuadro) de una secuencia de video se codifica de modo intracuadro, sin ninguna referencia respecto a anteriores o futuros fotogramas. Es lo que se llama *I-picture*, son los fotogramas normales o de imagen fija, proporcionando una compresión moderada, en JPEG. Los siguientes fotogramas o cuadros se codifican usando predicción intercuadro (*P-picture*, son imágenes predichas a partir de la imagen anterior). La predicción se basa en los datos del cuadro codificado inmediatamente anterior, ya sea *I-picture* o *P-picture*.

Cada cuadro es dividido en macro bloques no solapados, y cada macro bloque contiene bloques de datos de luminancia y crominancia (cuatro bloques de luminancia, Y1, Y2, Y3, Y4, y dos de crominancia, U y V), cada uno de tamaño 8*8 *pixeles*.

En el codificador se aplica la DCT a cada uno de los bloques de crominancia y luminancia. Se obtienen a la salida los bloques de 64 coeficientes DCT, que son cuantificados uniformemente. El escalón de cuantificación empleado es transmitido al receptor. Después de la cuantificación, el coeficiente DCT más pequeño (coeficiente DC) se procesa de forma diferente a los restantes (coeficientes AC).

- El coeficiente DC representa la intensidad media del bloque en cuestión y se codifica usando una técnica de predicción diferencial (debido a la fuerte correla-

[1]Macro bloque. El vector de movimiento es una estimación del desplazamiento horizontal y vertical de cada región de una cierta imagen con respecto a uno o varios fotogramas (frames) de la misma secuencia.

ción entre coeficientes DC de bloques adyacentes, se codifica la diferencia entre el del bloque anterior y el actual).

- El resto de coeficientes son barridos en zig-zag y codificados según un código VLC (*variable lenght code*). Esta técnica transforma la imagen en dos dimensiones en una ristra de bits de una dimensión. En el barrido se detectan los valores de los coeficientes AC no nulos, así como la distancia que separa a dos consecutivos. Este par de valores se codifica con una sola palabra código (VLC).

En el decodificador se realizan las operaciones inversas. Se extraen y decodifican las palabras código, para obtener así la localización y el valor cuantificado de los coeficientes DCT no nulos de cada bloque. Tras la reconstrucción y la aplicación de la transformada inversa, se obtienen los valores de los *pixeles* de ese bloque. Realizando la operación sucesivamente con todos los bloques, se obtiene la reconstrucción de la imagen.

Para la codificación de las *P-pictures*, el fotograma anterior es almacenado (FS, *frame store*) tanto en el codificador como en el decodificador. La técnica de compensación de movimiento se aplica sobre macro bloques, obteniendo los vectores de movimiento que son codificados y transmitidos al receptor.

El error de predicción se calcula para cada *pixel* de los contenidos en el macro bloque. Se calcula posteriormente la DCT de los bloques 8*8 que conforman el macro bloque, se cuantifican los coeficientes obtenidos y se codifican según un código VLC. En éste proceso es necesario el uso de un *buffer* para asegurar una determinada tasa binaria. El decodificador realiza el proceso inverso. Tras decodificar las palabras código, se reconstruyen los valores del error de predicción. Los *pixeles* compensados en movimiento del cuadro anterior, almacenados en el FS, se añaden al error de predicción para obtener el macro bloque del cuadro actual, ver figura 2.1.

La característica de relleno condicional consiste en la posibilidad de transmitir o no al receptor cierta información acerca de los macro bloques, según las necesidades. Existen tres tipos distintos de codificación de los macro bloques (MB):

- **Skipped MB**. No se transmite ni codifica información acerca del MB.

- **Inter MB**. Usa predicción por compensación de movimiento. Se transmite al receptor el tipo, dirección, vector de movimiento, coeficientes DCT, y escalón de cuantificación del macro bloque.

- **Intra MB**. Sólo usa predicción basada en el propio cuadro. No transmite, por tanto, vector de movimiento.

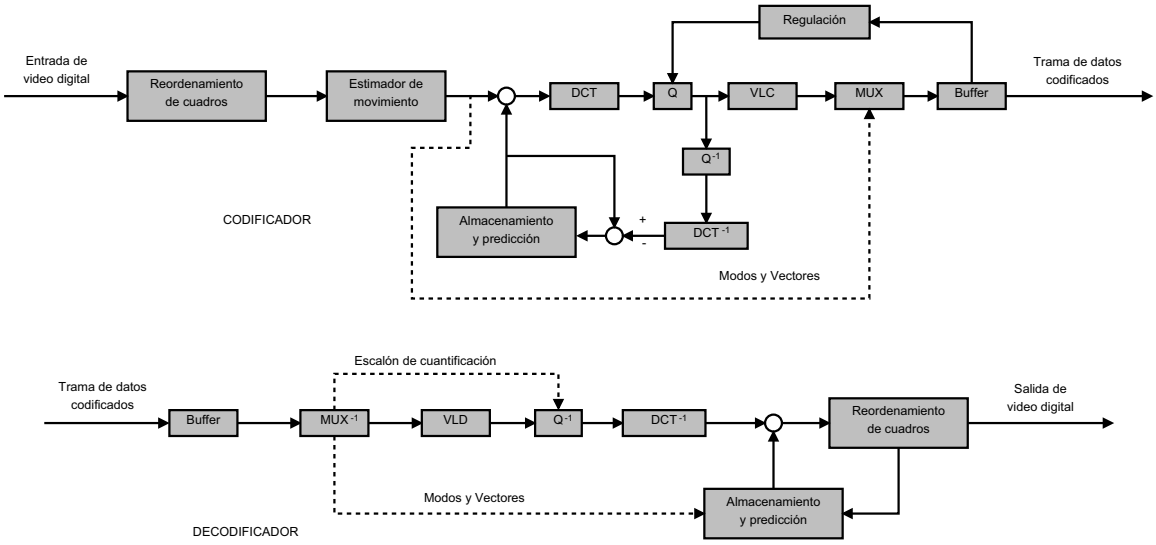

Figura 2.1: Codificación y Decodificación MPEG-1.

2.3. Análisis del algoritmo

Una característica importante de los algoritmos MPEG-1 es la flexibilidad en la tasa binaria, que puede variarse ajustando el escalón de cuantificación (en la cuantificación de los coeficientes DCT), según las exigencias de cada aplicación en particular. Esto permite el almacenamiento o transmisión de video con alto nivel de compresión. Además, el decodificador puede variar el escalón en cada macro bloque adaptándose a las necesidades de cada imagen.

Para el acceso a señales de video almacenadas, MPEG-1 desarrolla funcionalidades como FF, FR, o el acceso aleatorio. Para ello introduce el concepto de *B-pictures*. Las *I-pictures* son codificadas sin referencia a otros cuadros. Permiten de este modo introducir puntos de acceso que faciliten las funcionalidades mencionadas antes. Sin embargo, alcanzan un bajo grado de compresión. Las *P-pictures* no permiten estos puntos de acceso.

Las *B- pictures* (predicción/interpolación bidireccional) requieren anteriores y futuros cuadros o fotogramas para la codificación. Para conseguir mayor compresión se utiliza compensación de movimiento con el cuadro anterior y posterior (de tipo I o P).

Como norma general, una secuencia codificada usando sólo *I-pictures* (IIII...) consigue el mayor grado de accesibilidad pero la compresión más baja. Si se codifica combinando *I-* y *P-pictures* (IPPPPPIPPPP...) se consigue una solución de compromiso entre ambos aspectos, y si se usa la combinación de las tres (IBBPBBPBBI...) se consigue un alto grado de compresión y una razonable accesibilidad, aunque se

aumenta el retardo de codificación, lo que lo hace inviable para aplicaciones de videotelefonía o videoconferencia.

Los conceptos mencionados anteriormente están inmersos en el código de la biblioteca **ffmpeg**, del cual fue posible identificar la función **av_encode**, dicha función es la principal en el proceso de codificación de video, permitiendo realizar conversiones de formato de archivos de video entre estos y otros formatos disponibles en la biblioteca.

La función **av_encode** toma como base, una implementación del algoritmo de codificación MPEG, abarcando cualquiera de los tipos que comprenden a este estándar. Básicamente, identifica el tipo de formato y además las opciones de codificación a utilizar, lo que le permite a esta biblioteca utilizar el CODEC adecuado para cada tarea solicitada por el usuario. Una vez hecho esto, analiza el archivo de entrada para verificar que se encuentra integro y así poder realizar la codificación utilizando una implementación del formato que se desea trabajar, en éste caso MPEG-1. Esta implementación no es mostrada al público ni a los usuarios de esta biblioteca, por razones obvias de política y de seguridad por parte de la organización **FFmpeg**, sin embargo al ser un proyecto desarrollado bajo GNU/Linux es posible obtener el código fuente, y de esa manera conocer cómo trabaja.

El estándar MPEG no especifica una técnica única de compresión, sino un conjunto de las herramientas de compresión que pueden usarse en conformidad con reglas exactas de una sintaxis de compresión, las cuales fueron definidas en apartados anteriores y visualizados en el figura 2.1. Estas reglas exactas facilitan el intercambio de flujo de bits entre aplicaciones diferentes. Por lo que esta biblioteca trabaja bajo éste concepto: guarda una imagen, la compara con la siguiente y almacena sólo las diferencias. Se alcanzan así grados de compresión muy elevados.

Los pasos a seguir para la instalación de la biblioteca **ffmpeg** en un equipo de cómputo con linux Debian se encuentran en el apéndice A.

Capítulo 3

Paralelización

3.1. Grado de paralelización

El algoritmo propuesto, tiene como base para su desarrollo un esquema implementado con MPEG, estableciendo algunas variantes que permitan realizar una codificación de manera paralela, y de esa manera reducir el tiempo de la misma. Para dicha codificación "paralelizada", se busca realizar un balance de carga para los nodos del clúster involucrados en el proceso, es decir, que para cada nodo haya la misma cantidad de trabajo.

Para este trabajo de codificación paralela, se utilizó un clúster de computadoras bajo el sistema operativo Linux Debian (clúster *Beowulf*, el cuál se explica en el siguiente capítulo), utilizando la biblioteca para paso de mensajes Open MPI[4], en el cual no todos los nodos involucrados tienen la misma función. Existe un nodo maestro, cuyo propósito es coordinar y administrar la repartición de los fotogramas o GOPs, (*Group Of Pictures*) durante el proceso de codificación; el resto de los nodos son esclavos, encargados de dicho proceso, ver figura 3.1.

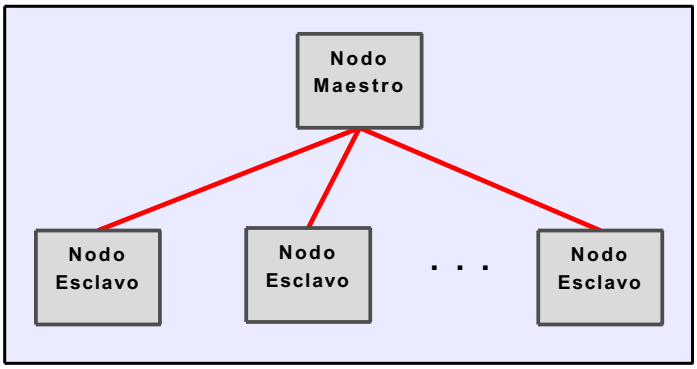

Figura 3.1: Nodos utilizados en la codificación.

Dado el hecho, de que para realizar la codificación de video para pasar del formato MPG ó VCD a MP4, trabajamos con la biblioteca **ffmpeg**, se instaló en cada uno de los nodos que formaron parte del clúster, por lo que la parte de paralelización consistió en el balanceo del trabajo para cada uno. El nodo maestro realizará una asignación de los fotogramas o GOPs que componen el archivo de video de entrada para codificar, basándose en el esquema de pre asignación, la cual está distribuida a lo largo de toda la secuencia de video, es decir que para 12 fotogramas y 4 nodos se tendría lo siguiente, al nodo 1 se le asigna el primer GOP, al nodo 2 el GOP 3 y así sucesivamente, ver figura 3.2.

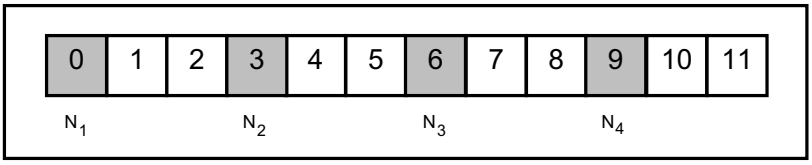

Figura 3.2: Esquema de distribución por pre asignación.

Para esta asignación se toma en consideración que el número de nodos es múltiplo del número de GOPs a comprimir. En caso contrario se trataría de hacer una distribución uniforme, siempre manteniendo un balance entre la carga de trabajo de cada uno de los nodos que componen el clúster, ver figura 3.3.

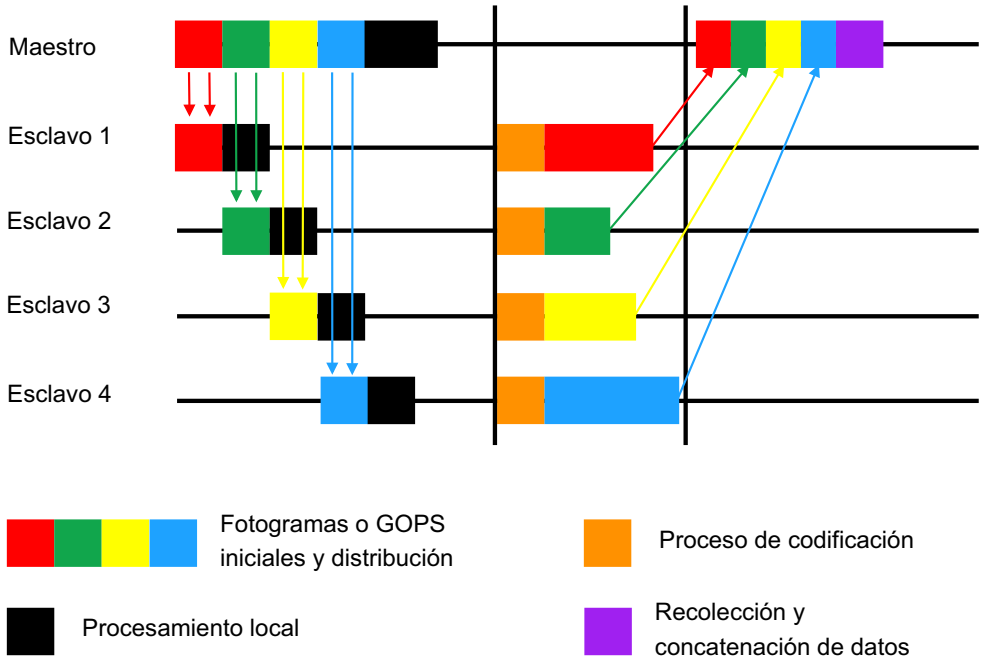

Figura 3.3: Proceso de codificación con envío de mensajes con *OpenMPI*.

3.2. Programación del algoritmo paralelizado

El programa paralelizado con Open MPI[5], recibe un archivo en formato MPG, ejemplo *video.mpg*, como fuente inicial de datos, el cual es codificado a formato mp4, proporcionando un archivo de salida.mp4, dicho archivo estará disponible para el usuario.

La codificación se realizó de acuerdo al siguiente algoritmo:

Algoritmo **mpg2mp4MPI**

```
|  inicio
|    inicialización del ambiente de MPI
|    si pid=0 entonces
|      leer archivo mpg
|      obtener tamaño de archivo mpg
|      copiar contenido del archivo mpg al buffer IN
|      distribuir contenido de IN a los procesadores del ambiente de MPI
|    si no
|      inicio proceso local
|        codificación a mp4 con ffmpeg
|        imprime tiempo total codificación
|      fin proceso local
|      inicio proceso global
|        leer archivo mp4
|        obtener tamaño de archivo mp4
|        copiar contenido archivo mp4 a buffer OUT
|        recolectar el contenido de OUT en el buffer FINAL
|      fin proceso global
|      leer contenido de FINAL
|      crear archivo final
|      escribir en final el contenido de FINAL
|    finalizar el ambiente de MPI
|  fin
```

Para conocer a detalle los códigos fuente escritos en lenguaje c, tanto del programa secuencial así como del paralelizado consultar el apéndice B.

Capítulo 4

Clúster

4.1. Introducción

El origen del término y del uso de este tipo de tecnología es desconocido pero se puede considerar que comenzó a finales de los años 50 y principios de los años 60. El término clúster se aplica a los conjuntos o conglomerados de computadoras construidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen una única computadora. Hoy en día desempeñan un papel importante en la solución de problemas de las ciencias, las ingenierías y del comercio moderno.

La tecnología de los clúster ha evolucionado en apoyo de actividades que van desde aplicaciones de super cómputo y software de misiones críticas, servidores web y comercio electrónico, hasta bases de datos de alto rendimiento, entre otros usos.

El cómputo con clúster surge como resultado de la convergencia de varias tendencias actuales que incluyen la disponibilidad de microprocesadores económicos de alto rendimiento y redes de alta velocidad, el desarrollo de herramientas de software para cómputo distribuido de alto rendimiento, así como la creciente necesidad de potencia computacional para aplicaciones que la requieran.

Simplemente, un clúster es un grupo de múltiples ordenadores unidos mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único ordenador, más potente que los comunes de escritorio.

4.2. Características

Para crear un clúster se necesitan al menos dos nodos. Una de las características principales de estas arquitecturas es que exista un medio de comunicación (red) donde los procesos puedan migrar para computarse en diferentes equipos de cómputo

paralelamente. Un solo nodo no cumple este requerimiento por su condición de aislamiento para poder compartir información. Por estas razones se deducen las siguientes características.

1. Un clúster consta de 2 o más nodos.

2. Los nodos de un clúster están conectados entre sí por al menos un canal de comunicación.

3. El clúster necesita software de control especializado.

4. Todos los elementos del clúster trabajan para cumplir una funcionalidad conjunta.

Existen varios tipos de software que pueden conformar un clúster.

- **A nivel de aplicación**. Se utilizan generalmente bibliotecas de carácter general que permiten la abstracción de un nodo a un sistema conjunto, este tipo de software suele generar elementos de proceso del tipo rutinas, procesos o tareas, que se ejecutan en cada nodo del clúster y se comunican entre sí a través de la red.

- **A nivel de sistema**. Suele estar implementado como parte del sistema operativo de cada nodo, o ser la totalidad de este. Es más crítico y complejo, por otro lado suele resolver problemas de carácter más general y su eficiencia, por norma general, es mayor.

4.3. Control

El parámetro de control de un clúster implica el modelo de gestión del clúster. Este modelo de gestión hace referencia a la manera de configurar el clúster y es dependiente del modelo de conexionado o colaboración que surgen entre los nodos. Puede ser de dos tipos.

- **Centralizado**. Se hace uso de un nodo maestro desde el cual se puede configurar el comportamiento de todo el sistema. Este nodo es un punto crítico del sistema aunque es una ventaja para una mejor gestión del clúster.

- **Descentralizado**. Suele ser utilizado en un modelo distribuido donde cada nodo debe administrarse y gestionarse.

4.4. Clasificación

El término clúster tiene diferentes connotaciones para diferentes grupos de personas. Los tipos de clúster, establecidos de acuerdo con el uso que se dé y los servicios que ofrecen, determinan el significado del término para el grupo que lo utiliza. Los clúster pueden clasificarse según sus características.

- **Alto rendimiento**. Son clúster en los cuales se ejecutan tareas que requieren de gran capacidad computacional, grandes cantidades de memoria, o ambos a la vez. El llevar a cabo estas tareas puede comprometer los recursos del clúster por largos periodos de tiempo.
- **Alta disponibilidad**. Son clúster cuyo objetivo de diseño es el de proveer disponibilidad y confiabilidad. Estos clúster tratan de brindar la máxima disponibilidad de los servicios que ofrecen. La confiabilidad se provee mediante software que detecta fallos y permite recuperarse frente a los mismos, mientras que en hardware se evita tener un único punto de fallos.
- **Alta eficiencia**. Son clúster cuyo objetivo de diseño es el ejecutar la mayor cantidad de tareas en el menor tiempo posible. Existe independencia de datos entre las tareas individuales. El retardo entre los nodos del clúster no es considerado un gran problema.

Los clúster pueden también clasificar como Clúster de IT Comerciales (Alta disponibilidad, Alta eficiencia) y Clúster Científicos (Alto rendimiento). A pesar de las discrepancias a nivel de requerimientos de las aplicaciones, muchas de las características de las arquitecturas de hardware y software, que están por debajo de las aplicaciones en todos estos clúster, son las mismas. Más aún, un clúster de determinado tipo, puede también presentar características de los otros.

4.5. Clúster Beowulf

El término Beowulf, al principio se refería a una computadora específica construida en 1994, Beowulf es un sistema de clúster de computadoras, similares al sistema construido originalmente por la NASA. Al principio desarrollado por Thomas Sterling y Donald Becker en la NASA, los sistemas Beowulf son desplegados ahora por todo el mundo, principalmente en apoyo del cómputo científico y el cómputo paralelo de alto rendimiento, utilizando hardware y PC's que son relativamente baratos.

Un clúster Beowulf es un grupo de computadoras con características similares ó idénticas que ejecutan software libre (*Open Source*) y sistema operativo Unix como OpenBSD, openSolaris ó GNU/Linux. En esta implementación los equipos están conectados en una pequeña LAN, tiene las bibliotecas y programas que permiten repartir procesos entre ellos, por ejemplo la *Message Passing Interface* (MPI) y *Parallel*

Virtual Machine (PVN). Ambas bibliotecas permiten al programador dividir una tarea en un grupo de equipos de cómputo dentro de la red, y recopilar los resultados del procesamiento.

En pocas palabras, el proyecto Beowulf, especifica cómo hacer un clúster de PC's con Linux.

Figura 4.1: Ejemplo de Clúster Beowulf.

Para conocer la configuración del clúster que se utilizó para realizar las pruebas del presente proyecto es necesario consultar el apéndice C.

Capítulo 5

Resultados

5.1. Pruebas en un nodo

El nodo en el que se realizaron las pruebas iniciales, es un equipo de cómputo con las siguientes características.

- Laptop DELL inspiron 1525
  - 2.0 GB en RAM
  - Procesador Intel Core 2 Duo CPU T5750@2.00GHz
  - Sistema Operativo Debian 5.0.3 lenny, Kernel Linux 2.6.26-1-686

Una vez que se identificó el algoritmo y el estándar de MPEG a utilizar se realizaron dos versiones en software del mismo. Una es secuencial y otra es la versión paralelizada con MPI, objetivo primordial de este proyecto.

El algoritmo secuencial, consiste en codificar un archivo *mpg* utilizando las bibliotecas que ofrece el paquete *ffmpeg*, entregando como salida un archivo *mp4*. Una vez terminado este algoritmo, se procedió a realizar pruebas para de esa manera obtener el tiempo que tarda en codificar un video mpg, estas pruebas se llevaron a cabo 10 veces cada una y con archivos de distintos tamaños.

Como era de esperar, los resultados arrojados por este análisis mostraron que mientras más grande era el archivo de entrada mayor era el tiempo que tarda en realizar la codificación, uno de los factores que influyeron en estos resultados, fue el hecho de que a pesar de contar con dos procesadores, durante el proceso de codificación sólo utilizaba uno de ellos, ver figura 5.1.

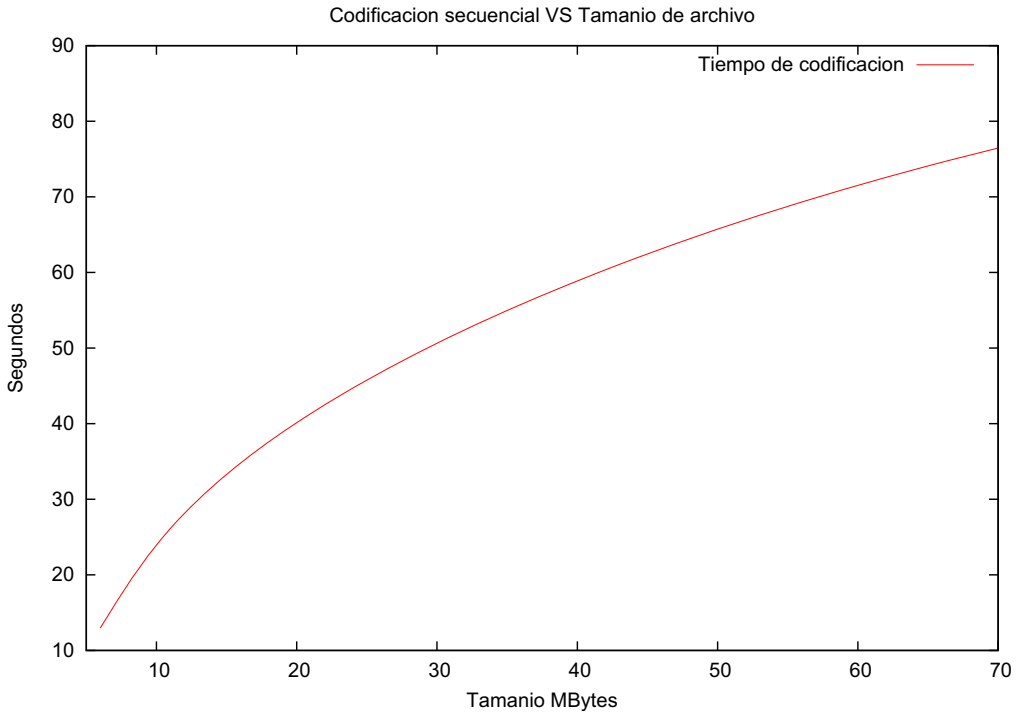

Figura 5.1: Ejecución en un nodo.

Sin embargo, el archivo de salida mostraba una reducción en tamaño bastante considerable, la cual variaba desde un 15 % a un 65 % con respecto al tamaño original, en la siguiente tabla se muestra la reducción en tamaño de los archivos de prueba.

Video	Tamaño MPG	Tamaño MP4
archivo 1	6 MB	4.8 MB
archivo 2	10.1 MB	6.4 MB
archivo 3	12.7 MB	10 MB
archivo 4	22.7 MB	7.2 MB
archivo 5	48 MB	23.8 MB
archivo 6	71.3 MB	25.5 MB

El algoritmo paralelizado **mpg2mp4MPI**, fue probado en principio en el nodo previamente mencionado, antes de finalizar la configuración del clúster con MPI en donde se realizaron las pruebas finales del algoritmo. Dichas pruebas fueron realizadas 10 veces seguidas con un archivo de video mpg de 72 MB.

Después de las pruebas se logró un tiempo promedio de ejecución de 77.039259 segundos, y al monitorear el estado de los procesadores se observó que ambos trabajaban durante la codificación del archivo. Por lo tanto esta reducción de 5.618579 segundos, es algo muy significativo en lo que a procesamiento de cómputo se refiere.

5.2. Pruebas en el clúster

Una vez que se configuró el clúster de computadoras, se realizaron pruebas tanto para el algoritmo secuencial y el paralelizado. Los equipos de cómputo que formaron parte del clúster contaban con las siguientes características.

- Equipo de cómputo DELL
  - 2.0 GB en RAM
  - 160 GB en Disco Duro
  - Procesador Intel Core 2 Duo
  - Tarjeta de Red Fast Ethernet (10/100 Mbps LAN)
  - Sistema Operativo Debian 5.0.3 lenny, Kernel Linux 2.6.26-1-686

En la figura 5.2 se puede observar un ajuste de cómo fueron variando los tiempos obtenidos durante las 10 pruebas realizadas con el algoritmo secuencial al codificar un archivo de video *mpg* de 11 MB.

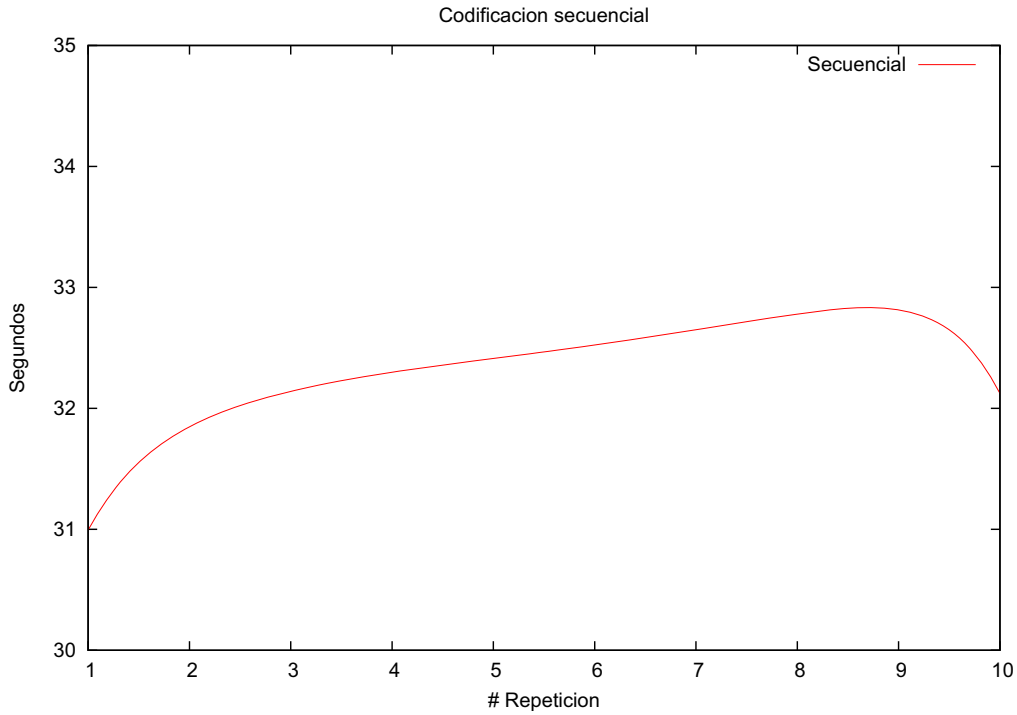

Figura 5.2: Ejecución secuencial.

Para cumplir el principal objetivo de este proyecto, se realizaron pruebas de codificación al ejecutar el programa que implementa el algoritmo **mpg2mp4MPI**, cada prueba se realizó 10 veces, después de las cuales se incrementaba el número de procesadores que participaban en el proceso de ejecución paralela del algoritmo, para de esa manera obtener un tiempo promedio de codificación.

Los resultados de los tiempos obtenidos durante la codificación de un video *mpg* de 11 MB con 2 procesadores en el clúster implementado se muestra su gráfica en la figura 5.3.

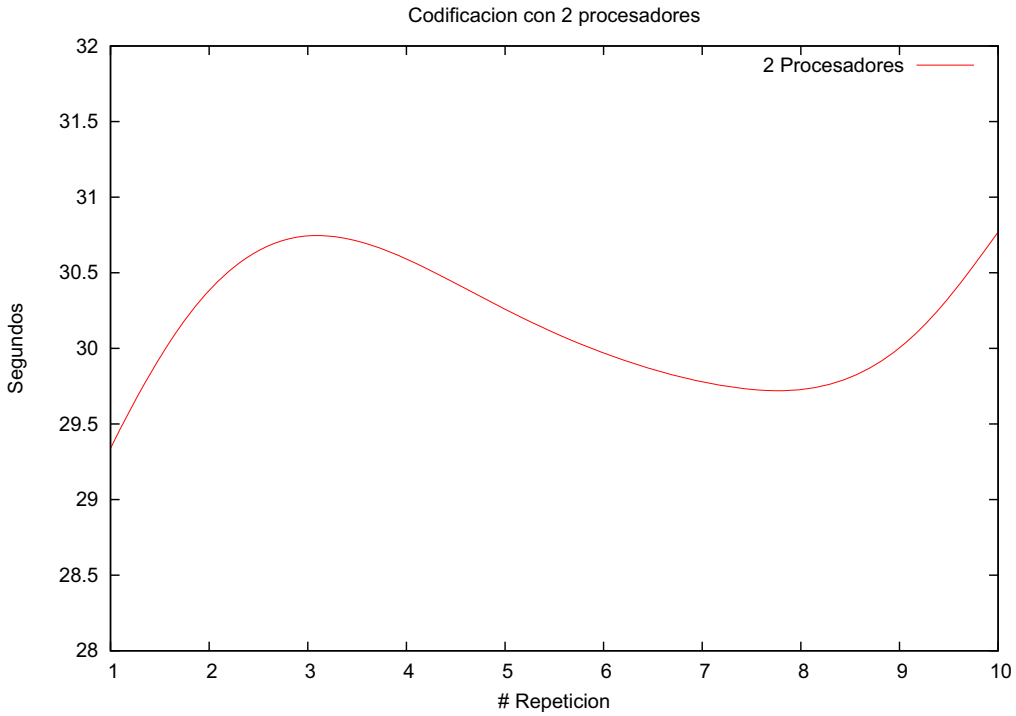

Figura 5.3: Ejecución paralela en 2 procesadores.

Para obtener pruebas suficientes fue necesario incrementar en cada ejecución el número de procesadores involucrados para llevar a cabo la codificación, por tal motivo en la figura 5.4 podemos observar la gráfica del tiempo de ejecución al trabajar con 4 procesadores.

Al ejecutar el programa paralelizado haciendo uso de 6 procesadores del clúster notamos que el tiempo de codificación entre una ejecución y otra no varía mucho, esto se observa en la figura 5.5.

La gráfica de los tiempos de ejecución al involucrar 8 procesadores se muestra en la figura 5.6.

El clúster estaba conformado por 5 equipos de cómputos lo que hizo un total de 10 procesadores, en la figura 5.7 vemos la gráfica de la ejecución haciendo uso de todos los procesadores del clúster.

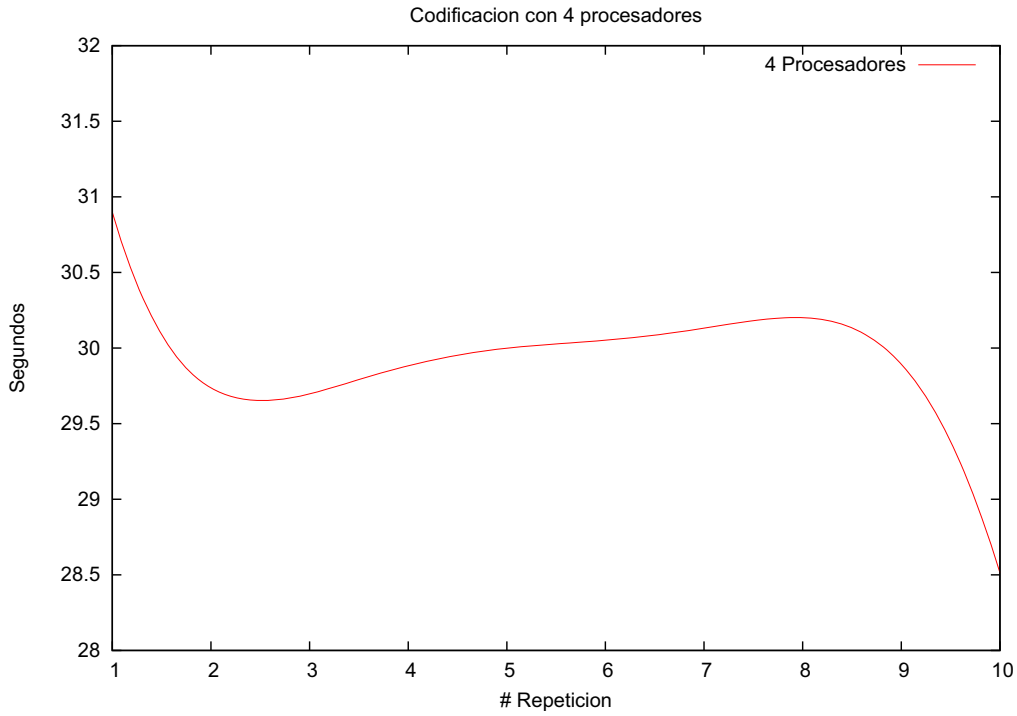

Figura 5.4: Ejecución paralela en 4 procesadores.

La finalidad de hacer varias pruebas, fue para tener promedios de tiempo de ejecución, en los que no intervinieran otros procesos que se estuvieran ejecutando en el mismo instante, es decir, tratar de obtener un balance en los tiempos de codificación resultantes, sin tomar en cuenta si al momento de la codificación se realizó una lectura ó una escritura al disco que haya consumido recursos y de esa manera afectar el tiempo final de ejecución.

Al gráficar los promedios de las ejecuciones obtenemos la siguiente gráfica (figura 5.8) en la cual notamos claramente que existe un reducción significativa del tiempo de ejecución al involucrar cada vez más procesadores en el proceso de codificación.

La tabulación de los promedios se muestra a continuación.

Procesadores	Tiempo
1	32.364529
2	30.171572
4	29.921556
6	29.547343
8	28.970591
10	27.842750

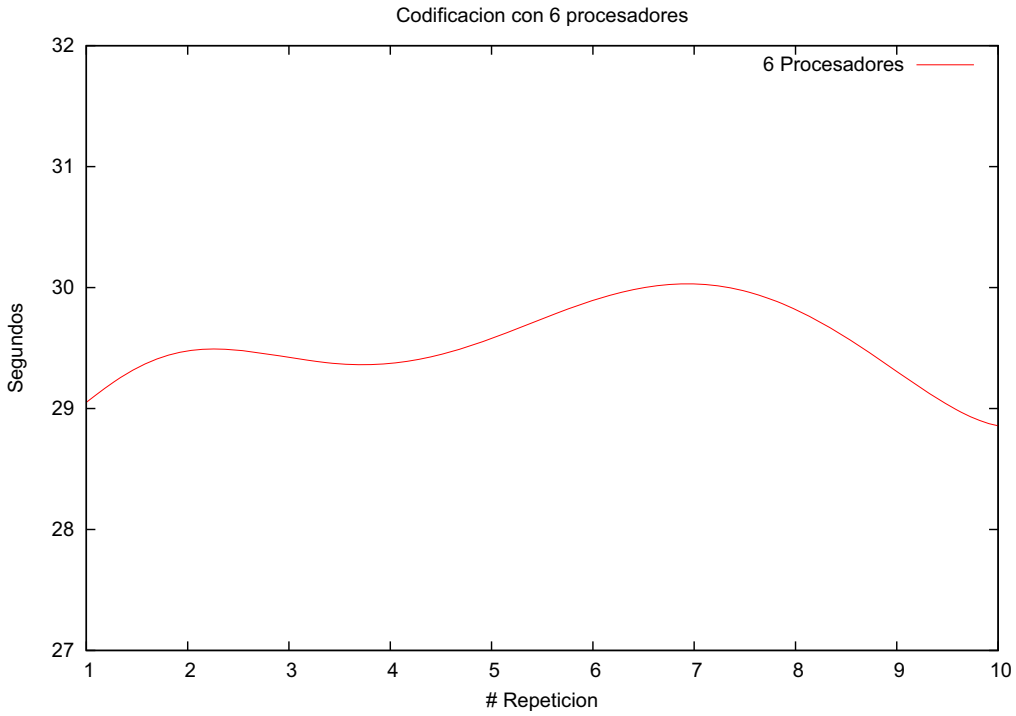

Figura 5.5: Ejecución paralela en 6 procesadores.

En la figura 5.9 se muestra una gráfica comparativa de todas las ejecuciones realizadas y se aprecia que a mayor número de procesadores menor tiempo de codificación.

Para una visualización agradable al usuario se diseñó una interfaz gráfica desde la cual es posible ejecutar el programa paralelizado en un clúster, las instrucciones a seguir se encuentran indicadas en el apéndice D.

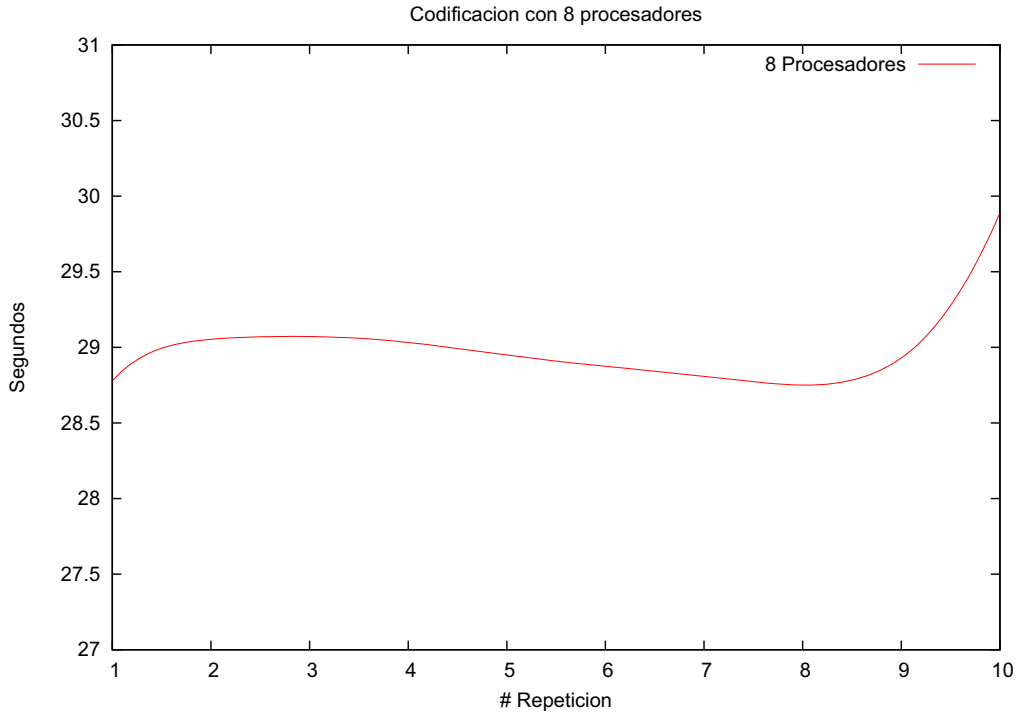

Figura 5.6: Ejecución paralela en 8 procesadores.

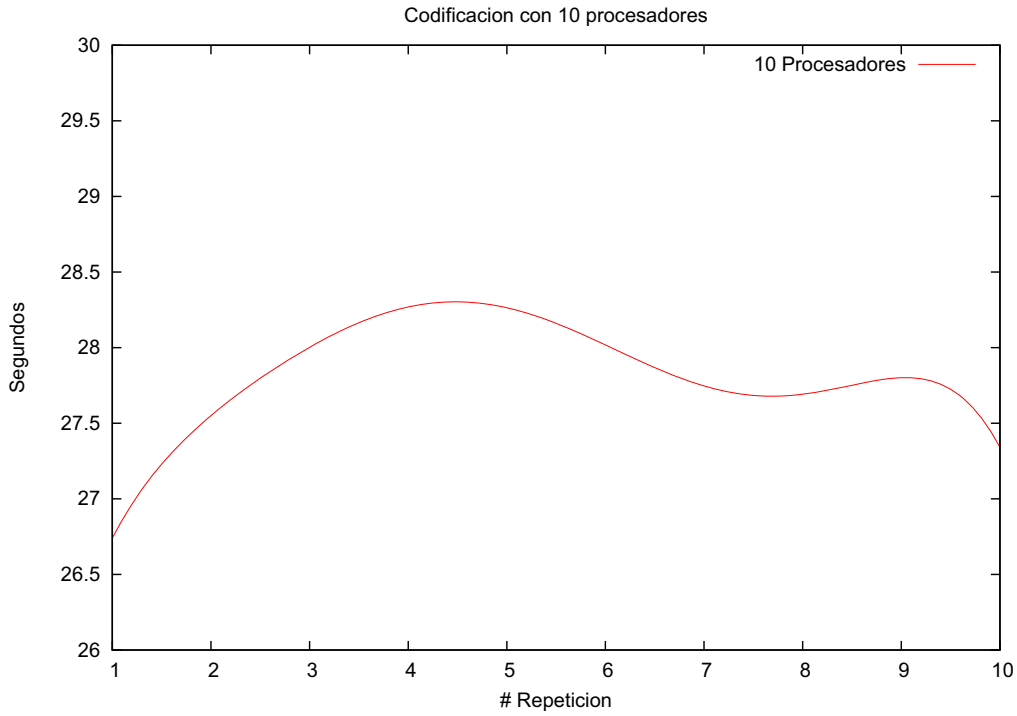

Figura 5.7: Ejecución paralela en 10 procesadores.

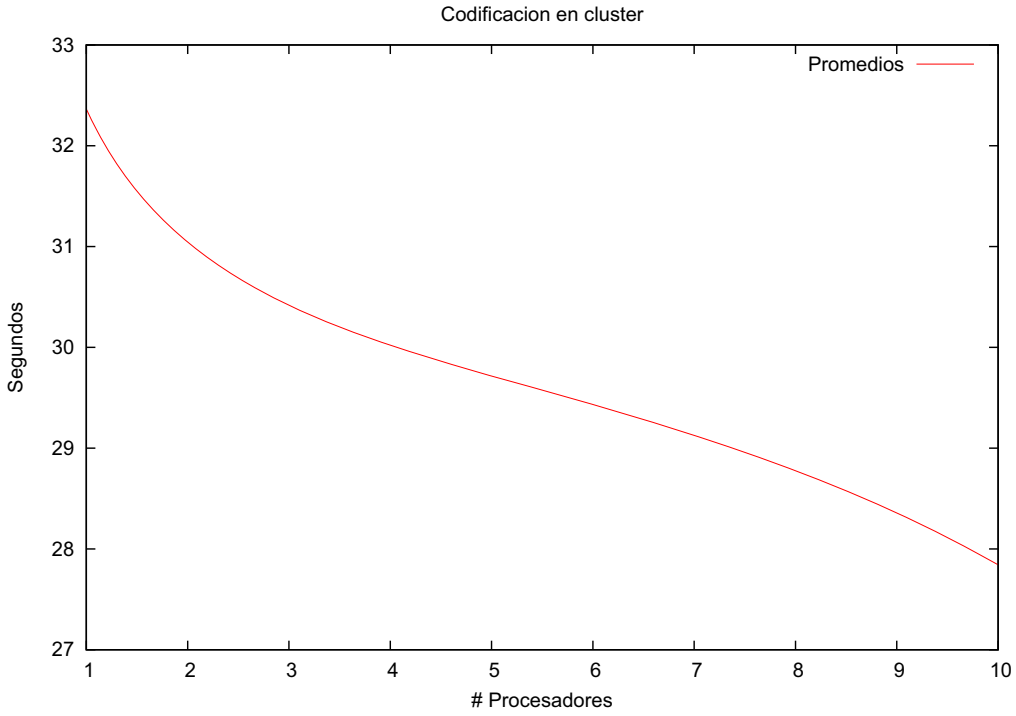

Figura 5.8: Ejecución en clúster.

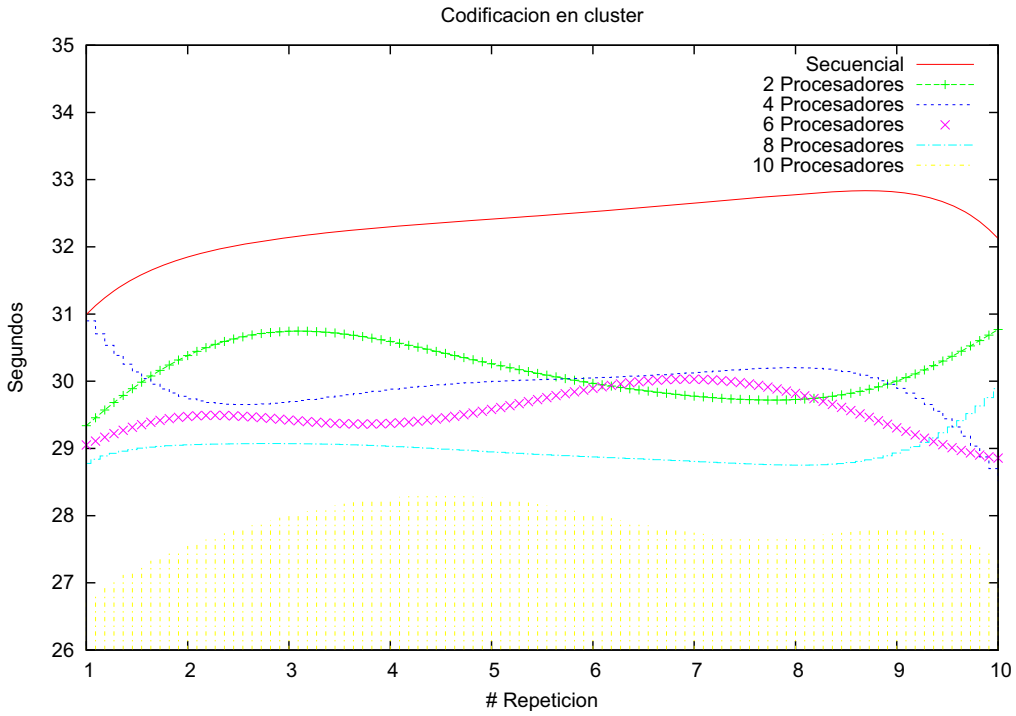

Figura 5.9: Gráfica comparativa.

Conclusiones

La razón por la que decidimos realizar este proyecto tiene que ver con el hecho del amplio crecimiento que ha y está teniendo el video digital en nuestra vida diaria, debido a la alta demanda se requiere alta calidad en el mismo, de ahí la elección del formato MPEG-4 ó mp4.

Al realizar las primeras pruebas con archivos de video mpg de distintos tamaños, logramos comprobar una hipótesis planteada al inicio del proyecto, teníamos la idea de que mientras más grande era el archivo de video de entrada más tiempo tardaría la codificación del mismo, esto fue evidente al trabajar el algoritmo secuencial en un sólo nodo.

Por otro lado, al realizar pruebas con el algoritmo paralelizado en el clúster implementado, notamos que mientras más procesadores utilizábamos para realizar la codificación de una entrada de video menor era el tiempo en que se realizaba, tanto para archivos pequeños como grandes, claro que la diferencia en tiempo de procesamiento entre uno y otro eran distintos, pues su diferencia en tamaño era proporcional al tiempo de codificación.

Durante dichas pruebas, a pesar de que inicialmente la ejecución del algoritmo de codificación paralela se llevo a cabo en un sólo equipo de cómputo, logramos identificar desde el primer instante que al hacer uso de la biblioteca de Interfaz de Paso de Mensajes *OpenMPI*, se obtuvo una reducción en el tiempo de codificación de un formato a otro, lo cual se debe al uso de todos los procesadores involucrados en el ambiente de MPI.

A lo largo de las pruebas que realizamos notamos que si el archivo *my-hostfile* utilizado al momento de la ejecución del programa paralelizado no contiene todas las identificaciones de los nodos que forman el clúster, con sus respectivos *hostname* y número de *slots*, la codificación tarda más tiempo en realizarse. De igual manera sucede si el número de procesadores especificados en la línea de comando excede a los contenidos en dicho archivo.

Para paralelizar un algoritmo es importante entender claramente cada una de las

partes del algoritmo secuencial, todo esto con la finalidad de identificar la parte del procesamiento que se puede llevar a cabo de manera paralela, de realizarse correctamente podrá presentarse una optimización.

Finalmente, cabe señalar dos puntos importantes de este proyecto: el primero que se logró cumplir con el objetivo principal, pues el tiempo de codificación se vio reducido aún más al estar involucrados más procesadores en el ambiente de MPI, tal vez parezca mínima dicha reducción, pero es bastante significativa, ya que en la actualidad el aprovechamiento eficaz y eficiente de nuestros equipos de cómputo es primordial para el cumplimiento de nuestras tareas, investigaciones, desarrollos de proyectos, entre otras. El segundo punto importante es el hecho de que también se obtuvo una reducción del tamaño del archivo de salida gracias al formato elegido MPEG-4, este formato proporciona mayores ventajas comparado con otros apoyados en el mismo estándar de codificación, ofrece al usuario una mayor calidad, ideal para los nuevos dispositivos.

La continuidad del proyecto, es factible realizarla a raíz de nuevos estándares de codificación ó compresión y nuevas tecnologías multimedia, al grado de realizar nuevas adaptaciones y/o diseños sobre esos estándares, las cuales permitan escoger el formato de codificación que se desea emplear sobre un archivo de video en particular.

La aplicación aquí desarrollada, puede ser mejorada al implementar tanto un algoritmo que permita visualizar al usuario todos los archivos MPG disponibles en su computadora y ponerlos en una lista en la que el usuario sólo tenga que seleccionar el video deseado y así facilitar la tarea de buscar los videos que se quiere cambiar de formato.

De manera conjunta desarrollar una interfaz gráfica que utilice la lista generada por el algoritmo descrito anteriormente, y que de igual forma permita monitorear el estado de todos los procesadores involucrados en la codificación.

Apéndice A

Manual de instalación biblioteca ffmpeg

Introducción

FFmpeg es una colección de software libre escrita en lenguaje C. Es una herramienta en línea de comando que puede grabar, convertir videos de un formato a otro y hacer *streamings* de audio y vídeo, ya que incluye libavcodec, una biblioteca de códecs. **FFmpeg** está desarrollado en GNU/Linux, pero puede ser compilado en la mayoría de los sistemas operativos, incluyendo Windows.

Instalación

FFmpeg está disponible para todas las distribuciones de Linux existentes, en nuestro caso particular trabajamos con **Debian lenny 5.0.3, kernel linux 2.6.26-1-686**. Por lo tanto, se debe instaló dicha biblioteca con la herramienta **apt-get**.

Cabe señalar que para todo proceso de instalación el usuario responsable debe ser **root**. A continuación se muestra el comando para instalar la biblioteca **ffmpeg**.

```
# apt-get install ffmpeg
```

Con esta instrucción, se instalan las características más básicas de la biblioteca, las cuales ya permiten trabajar con video en formato **mpeg-1** ó **mpg**, pero por otra parte aún no soporta formatos más recientes y con alta calidad de video como lo es **mpeg-4** ó **mp4**, a continuación se muestra como instalar la versión más reciente de la biblioteca **ffmpeg** y el encoder **x264**, con esto ya podrá trabajar con el formato **mp4**, necesario en el proyecto.

Instalar dependencias

Lo siguiente es, instalar todos los paquetes de dependencias que requieren para utilizar **ffmpeg** y **x264**.

```
# apt-get update
# apt-get install build-essential subversion git-core checkinstall yasm
texi2html libfaad-dev libsdl1.2-dev libtheora-dev libx11-dev libxfixes-
dev zlib1g-dev
```

Instalar x264

Ahora es necesario obtener los archivos fuentes más recientes de los repositorios de x264, compilarlos e instalarlos.

```
$ cd
$ git clone git://git.videolan.org/x264.git
$ cd x264
$ ./configure
$ make
# checkinstall --pkgname=x264 --pkgversion "2:0.'grep X264_BUILD x264.h
-m1 | cut -d''-f3'.'git rev-list HEAD | wc -l'+git'git rev-list HEAD
-n 1 | head -c 7'" --backup=no --default
```

Instalar ffmpeg

Por último, con los siguientes comandos se obtienen los archivos fuentes más recientes del SVN de FFmpeg, se compilan y se instalan.

```
$ cd
$ svn checkout svn://svn.ffmpeg.org/ffmpeg/trunk ffmpeg
$ cd ffmpeg
$ ./configure --enable-gpl --enable-version3 --enable-nonfree --enable-
postproc --enable-pthreads --enable-libfaad --enable-libtheora --enable
-libxvid --enable-x11grab
$ make
# checkinstall --pkgname=ffmpeg --pkgversion "4:SVN-r'svn info | grep
Revision | awk '{ print $NF }''" --backup=no --default
$ hash x264 ffmpeg
```

Por último para probar que todo se instaló bien y funciona de manera correcta, teclee el siguiente comando en la línea de comando:

```
$ ffmpeg -i ArchivoIN.mpg ArchivoOUT.mp4
```

En dónde ArchivoIN.mpg es el nombre y/o ruta de ubicación de un archivo de video mpg, el cuál será cambiado a formato mp4 (ArchivoOUT.mp4). Si el proceso se completa de manera satisfactoria todo se instaló y funciona correctamente, de lo contrario repetir los pasos de instalación.

Para anular todos los cambios hechos en su sistema indicados en este manual, utilice el siguiente comando:

```
# apt-get remove x264 ffmpeg build-essential subversion git-core
checkinstall yasm texi2html libfaad-dev libsdl1.2-dev libtheora-dev
libx11-dev libxfixes-dev libxvidcore-dev zlib1g-dev
```

Apéndice B

Códigos fuente

Archivo: `codecSecuencial.c` (*Codificación secuencial*)

```
1  /*Programa secuencial, para la codificación de video de formato
2  * mpg a mp4 utilizando la biblioteca "ffmpeg".
3  *
4  * Modo de uso:
5  * ./codecSecuencial <arch-in>
6  *
7  * En donde: "arch-in", es el nombre del archivo de entrada .mpg
8  * que se quiere cambiar de formato; el cual tendrá el nombre
9  * "Output" para el archivo de salida .mp4
10 */
11
12 #include <stdlib.h>
13 #include <stdio.h>
14 #include <string.h>
15 #include <time.h>
16
17 int main(int argc, char *argv[]){
18
19     char comando[100];
20     struct timeval inicio,fin;
21     float runtime;
22
23     if(argc<2)
24         printf("ERROR de ejecución...\nUso:\n%s <archivo-In>",↩
               argv[0]);
25     else{
26         printf("\nInicio codificación...\n");
27         sprintf(comando,"ffmpeg -i %s -y Output.mp4",argv[1]);
28         gettimeofday(&inicio,(struct timezone *)0);
```

```
29          system(comando);
30          gettimeofday(&fin,(struct timezone *)0);
31          runtime=(float)(fin.tv_sec-inicio.tv_sec)*1000000+(float↩
                )(fin.tv_usec-inicio.tv_usec);
32          runtime /=1000000.0;
33          printf("\nFin codificación...\nTiempo de codificacion: %↩
                f\n",runtime);
34      }
35  }
```

Archivo: `mpg2mp4MPI.c` (*Codificación paralela*)

```
1  #include <mpi.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5  #include <unistd.h>
6  #include <fcntl.h>
7  #include <sys/stat.h>
8
9  int main(int argc, char*argv[])
10 {
11      int myid, numprocs, namelen;
12
13      FILE *fd_in, *fd_out, *fd_final;
14      struct stat archivo_in, archivo_out;
15      double tmpinic=0.0, tmpfin;
16
17      struct timeval inicio,fin;
18      float runtime;
19
20      char comandoFfmpeg[100];
21      char processor_name[MPI_MAX_PROCESSOR_NAME];
22
23      MPI_Init(&argc,&argv);
24      MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
25      MPI_Comm_rank(MPI_COMM_WORLD,&myid);
26      MPI_Get_processor_name(processor_name, &namelen);
27
28      if (myid == 0) /*Abre el archivo de entrada e inicializa los↩
            datos*/
29      {
30          if((fd_in=fopen(argv[1],"r"))==NULL)
31          {
```

```
32              printf("\nError al leer el archivo de entrada\n");
33              exit (1);
34          }
35
36          printf("\nArchivo abierto... ");
37          stat(argv[1], &archivo_in);
38          printf("Con %ld bytes\n", (long int) archivo_in.st_size)↩
                ;
39      }
40
41      if((fd_out=fopen("Output.mp4","rw"))==NULL)
42      {
43          printf("\nError al leer el archivo de salida\n");
44          exit (1);
45      }
46
47      /*Distribución de los datos*/
48      MPI_Scatter(fd_in, (long int)archivo_in.st_size/numprocs, ↩
            MPI_INT, fd_out, (long int)archivo_in.st_size/numprocs, ↩
            MPI_INT, 0, MPI_COMM_WORLD);
49
50      /*Codificación de datos local*/
51      sprintf(comandoFfmpeg,"ffmpeg -i %s -y Output.mp4",argv[1]);
52      printf("\nInicio de codificación...\n");
53      gettimeofday(&inicio,(struct timezone *)0);
54      system(comandoFfmpeg);
55      gettimeofday(&fin,(struct timezone *)0);
56      runtime=(float)(fin.tv_sec-inicio.tv_sec)*1000000+(float)(↩
            fin.tv_usec-inicio.tv_usec);
57      runtime /=1000000.0;
58      printf("\nFin de codificación...\nTiempo de codificación: %f↩
            \t\tNodo %d\n",runtime,myid);
59
60      printf("\nArchivo nuevo... ");
61      stat("Output.mp4", &archivo_out);
62      printf("Con %ld bytes\n", (long int) archivo_out.st_size);
63
64      /*Recolección y unión de los datos*/
65      MPI_Gather(fd_out, (long int)archivo_out.st_size/numprocs, ↩
            MPI_INT, fd_final, (long int)archivo_out.st_size/numprocs↩
            , MPI_INT, 0, MPI_COMM_WORLD);
66
67      /*Finaliza ambiente MPI*/
68      MPI_Finalize();
69
70      fclose(fd_in);
```

```
71      fclose(fd_out);
72      fclose(fd_final);
73  }
```

Apéndice C

Manual de configuración de un clúster beowulf

Introducción

El término clúster se aplica a los conjuntos o conglomerados de computadoras construidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen una única computadora.

En concreto, el proyecto Beowulf especifica como implementar un clúster de Pc's bajo Linux.

Hardware requerido

Nodo Maestro

- CPU
- Disco duro
- Memoria RAM
- Tarjeta de red

Nodo Esclavo

- CPU
- Memoria RAM
- Tarjeta de red

Interconexión

- Red de alta velocidad Fast Ethernet ó Gigabit Ethernet.

Servicios en los nodos

Es necesario activar los siguientes servicios

1. rsh
2. rlogin
3. rexec
4. nfs

Instalación

En todos los nodos

```
# apt-get install rsh-server rsh-client
```

En el nodo maestro del clúster

```
# apt-get install nfs-common nfs-kernel-server
```

En los nodos esclavos que pertenecen al clúster

```
# apt-get install nfs-common nfs-user-server
```

Inicialización

```
$ /etc/init.d/xinetd restart
```

Identificación de los nodos

Para los nodos que pertenecen al clúster, es necesario identificarlos dentro del archivo **/etc/hosts** del sistema, es decir, escribir el nombre de las máquinas y la IP correspondiente.

```
# vim /etc/hosts
  127.0.0.1 servidor
  148.206.74.121 servidor
  148.206.74.223 cliente1
  148.206.74.222 cliente2
```

```
148.206.74.210 cliente3
148.206.74.221 cliente4
...
```

Acceso remoto

Para el acceso remoto en el clúster, es necesario renombrar a los nodos con sus respectivos usuarios en el archivo **/etc/hosts.equiv**

```
# vim /etc/hosts.equiv
  servidor f307-user
  cliente1 f307-user
  cliente2 f307-user
  cliente3 f307-user
  cliente4 f307-user
  ...
```

Servidor NFS

Para el equipo que fungirá como servidor nfs (nodo maestro), es necesario realizar algunos cambios en el archivo **/etc/exports** para registrar a los nodos esclavos. Con lo cual permitirá exportar el *home* del nodo maestro a los nodos esclavos, esto se facilitará si los nodos cuentan con un mismo usuario.

```
# vim /etc/exports
  /home/f307-user cliente1(rw, sync)|
  /home/f307-user cliente1(rw, all squash, anonuid=150, anongid=100)|
  /home/f307-user cliente1(rw, sync, no_subtree_check)|
  /home/f307-user cliente1(rw, sync, no_subtree_check, anounid=150,
                           anongid=500)
```

Para utilizar las opciones que vienen por defecto en la configuración del **nfs** para exportar el */home* del nodo maestro a los distintos nodos esclavos, basta con escribir la siguiente línea para cada uno de los nodos esclavos que componen el clúster:

```
/home/usuario hostname()
```

```
# vim /etc/exports
  /home/f307-user cliente1()
  ...
```

Por último, es necesario reiniciar el servicio **nfs-kernel-server** en el nodo maestro, para así poder exportar el */home* a los nodos esclavos registrados.

```
# /etc/init.d/nfs-kernel-server restart
```

Cliente NFS

Todos los nodos esclavos que van a montar el */home* del nodo maestro necesitan:

```
# iptables -F
# mount -t nfs servidor:/home/f307-user /home/f307-user
```

Instalación *OpenMPI*

Paquetes necesarios para la instalación y utilización de *OpenMPI*

```
# apt-get install openmpi-doc openmpi-common openmpi-bin libopenmpi-
dev libopenmpi-dbg libopenmpi1
```

Administración

El siguiente comando es necesario para la administración de OpenMPI en los nodos del clúster

```
#ln -s mpicc-wrapper-data.txt mpicc.openmpi-wrapper-data.txt
```

Seguridad

```
# ssh-keygen -t rsa
# ssh f307-user@cliente1 mkdir -p .ssh
# cat .ssh/id rsa.pub | ssh f307-user@cliente1 'cat >>.ssh/authorized
keys'
```

Topología

A través del archivo **/etc/my-hostfile** es posible identificar el número de slots con los que cuenta cada nodo del clúster, y de esa manera administrar y distribuir los procesos dentro de los nodos del clúster.

```
# vim /etc/my-hostfile
```

```
servidor slots=2
cliente1 slots=2
cliente2 slots=2
cliente3 slots=2
cliente4 slots=2
...
```

Compilación y ejecución *OpenMPI*

Compilación

```
$ mpicc ARCHIVO.c -o EJECUTABLE
```

En dónde, ARCHIVO.c es el código fuente del programa con instrucciones de *OpenMPI* y EJECUTABLE es el nombre del archivo ejecutable correspondiente al archivo fuente.

Ejecución

```
$ mpirun --hostfile my-hostfile -np 2 EJECUTABLE
```

En dónde, *my-hostfile* es el archivo que previamente se configuró y el *2* corresponde al número de *slots* indicados en ese archivo para cada nodo, por último EJECUTABLE es el archivo ejecutable.

Apéndice D

Guía de uso Interfaz mpg2mp4GUI

Introducción

Dentro de la carpeta PT, se encuentran dos carpetas más GUI_MPI, que contiene los archivos de código fuente en c++ de la interfaz **mpg2mp4GUI** y programaMPI, en esta carpeta se encuentra el código del programa con MPI escrito en lenguaje C y su correspondiente ejecutable **mpg2mp4MPI**.

mpg2mp4GUI, es el programa utilizado por el usuario para llevar a cabo la codificación de los videos en formato mpg a mp4, objetivo principal de este proyecto. Es una interfaz gráfica de usuario desarrollada con Qt Creator, el cual es un IDE creado por *Trolltech* para el desarrollo de aplicaciones con las bibliotecas Qt, en su versión 4.x. Este IDE es software open source con licencia GPL, por lo que puede ser descargado desde el siguiente enlace `http://qt.nokia.com/downloads`, más especificamente `http://get.qt.nokia.com/qtsdk/qt-sdk-linux-x86-opensource-2010.04.bin`.

La razón por la que se decidió utilizar este IDE fue el hecho que nos permitía utilizar programación visual, la cual es muy útil para el desarrollo de interfaces.

Dentro de la carpeta GUI_MPI, están los archivos

- *mpg2mp4GUI.pro*, proyecto de Qt Creator
- *main.cpp*, archivo principal que ejecuta la aplicación
- *mainwindow.ui*, en donde se realiza la programación visual de la interfaz
- *mainwindow.h*, archivo con la declaración de los elementos y slots de la clase *MainWindow*, utilizada por la interfaz
- *mainwindow.cpp*, archivo con el código de los slots definidos en el archivo *.h*

Para la compilación de los archivos hay que utilizar el IDE Qt Creator y de esa manera emplear sus herramientas de compilación, para posteriores compilaciones basta con utilizar la línea de comandos y teclear, make, este Makefile es creado por el IDE, y no es necesario realizar procesos adicionales. Para la ejecución, hay dos maneras de hacerlo desde el IDE con sus herramientas, y la otra nuevamente con la línea de comandos `$./mpg2mp4GUI`.

La interfaz **mpg2mp4GUI** utiliza el ejecutable **mpg2mp4MPI**, por esa razón esos dos archivos ejecutables se encuentran juntos fuera de sus respectivas carpetas contenedoras, es decir en la carpeta PT. Al ejecutar la interfaz (`$./mpg2mp4GUI`) de igual manera se ejecutará mpg2mp4MPI, este es el que realizará la codificación realmente, por lo que es importante mencionar que dentro de la carpeta PT debe de estar el archivo *my-hostfile*, del cual es descrito su uso en el apéndice anterior.

Código fuente de la interfaz

Archivo: `mainwindow.h`

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QtGui/QMainWindow>
5
6  namespace Ui
7  {
8      class MainWindow;
9  }
10
11 class MainWindow : public QMainWindow
12 {
13     Q_OBJECT
14
15 public:
16     MainWindow(QWidget *parent = 0);
17     ~MainWindow();
18
19 private:
20     Ui::MainWindow *ui;
21
22 private slots:
23     void on_commandLinkButton_clicked();
24     void on_Inicio_clicked();
25     void on_findFile_clicked();
26 };
```

```
27
28 #endif // MAINWINDOW_H
```

Archivo: `mainwindow.cpp`

```
1  #include "mainwindow.h"
2  #include "ui_mainwindow.h"
3
4  #include <iostream>
5  #include <QFileDialog>
6  #include <QMessageBox>
7
8  using namespace std;
9
10 MainWindow::MainWindow(QWidget *parent)
11     : QMainWindow(parent), ui(new Ui::MainWindow)
12 {
13     ui->setupUi(this);
14 }
15
16 MainWindow::~MainWindow()
17 {
18     delete ui;
19 }
20
21 void MainWindow::on_findFile_clicked()
22 {
23     /*
24       * Aquí se selecciona un archivo MPG mediante un filtro, ↩
             posteriormente se copia el nombre del archivo
25       * seleccionado y se coloca en la casilla 'Nombre del video↩
             ' en la interfaz, además de que servirá para
26       * pasar como parametro a la aplicación mpg2mp4MPI
27     */
28     QString nombre;
29     nombre=QFileDialog::getOpenFileName(this, "Seleccionar ↩
           archivo de video", "", tr("Archivos MPG (*.mpg)"));
30     ui->fileName->setText(nombre);
31 }
32
33 void MainWindow::on_Inicio_clicked()
34 {
35     char num[10];
36     char nombre[150];
37     char ejecutaCodecMPI[200];
```

```
38      int min=0, max=100, a=0, i, j, valor=0;
39
40      /*Se asignan valores máximos y mínimos en la barra de estado↩
            */
41      ui->progressBar->setMinimum(min);
42      ui->progressBar->setMaximum(max);
43      strcpy(num,ui->spinBoxNProcs->text().toAscii());
44      strcpy(nombre,ui->fileName->text().toAscii());
45      /*Se genera la cadena que contiene el comando que ejecutará ↩
            la aplicación mpg2mp4MPI*/
46      sprintf(ejecutaCodecMPI,"mpirun --hostfile my-hostfile -np %↩
            s mpg2mp4MPI %s",num,nombre);
47      a=atoi(num); valor=max/a;
48      QMessageBox::information(this, "Notificacion", "Iniciando ↩
            codificacion mpg a mp4...");
49      /*Ejecución de la aplicación mpg2mp4MPI mediante 'system'*/
50      system(ejecutaCodecMPI);
51      ui->progressBar->setValue(10);
52      for(i=1; i<=a; i++){
53          for(j=1; j<=valor; j++)
54              if(j==valor)
55                  ui->progressBar->setValue(valor*i);
56      }
57      QMessageBox::information(this, "Notificacion", "Codificacion↩
            finalizada...");
58  }
59
60  void MainWindow::on_commandLinkButton_clicked(){ }
```

Archivo: `main.cpp`

```
1  #include <QtGui/QApplication>
2  #include "mainwindow.h"
3
4  int main(int argc, char *argv[])
5  {
6      QApplication a(argc, argv);
7      MainWindow w;
8      w.show();
9      return a.exec();
10 }
```

Descripción interfaz mpg2mp4GUI

La interfaz consta de 5 partes, ver figura D.1.

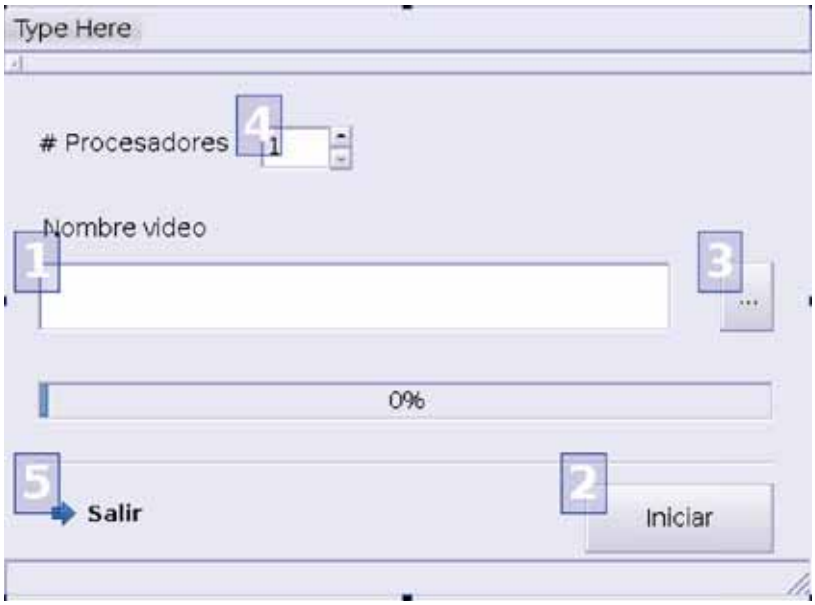

Figura D.1: Componentes de la mpg2mp4GUI.

1. Caja de texto donde se escribe el nombre del archivo que se codificará de mpg a mp4, con su correspondiente ruta de ubicación.

2. Botón de Inicio, el cual ejecuta el programa que realiza la codificación del video, cuyo nombre esta escrito en la caja de texto descrita anteriormente.

3. Botón de exploración, el cual permite al usuario seleccionar un archivo de video mpg que este almacenado en su equipo.

4. Spin box con los números del 1 al 10 que permiten al usuario indicar el número de procesadores que desea utilizar para la codificación de su archivo de video. Cabe señalar que números mayores a 2 es preferible utilizarlos con el clúster en el cuál se trabajará, pero no es indispensable hacerlo.

5. Botón de comando, con el que el usuario sale de la aplicación y termina el proceso de codificación.

La ejecución de la interfaz desde la línea de comando, ver figura D.2, muestra que en el spin box el menor número que es posible utilizar es 1, en la caja de texto se escribe la ruta completa de ubicación del archivo seleccionado. De igual manera, esta interfaz cuenta con una barra de progreso del proceso de ejecución de la codificación de mpg a mp4. Antes de iniciar la codificación del archivo aparece un mensaje indicando

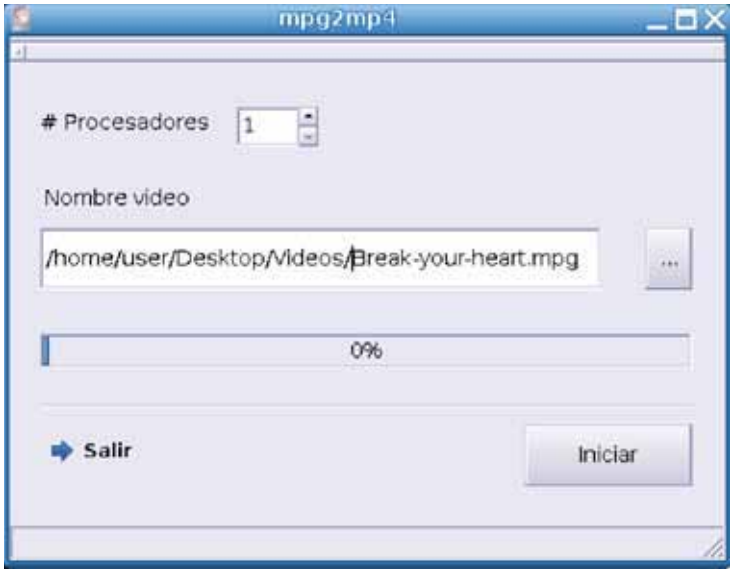

Figura D.2: Ejecución mpg2mp4GUI.

al usuario que se llevará a cabo la misma, ver figura D.3. Al finalizar la codificación de igual manera aparece un mensaje indicándolo al usuario, ver figura D.4.

El archivo de salida generado (Output.mp4), estará contenido en la carpeta que contiene esta interfaz (PT), para de esa manera permitirle al usuario copiarlo a la ubicación que sea más conveniente para él, al igual que podrá renombrar el archivo mp4 con el nombre que desee, de ahí el hecho de llamar el archivo de salida Output.mp4.

Figura D.3: Notificación inicio de la codificación mpg2mp4.

Figura D.4: Notificación finalización de la codificación mpg2mp4 y estado de la barra de progreso.

Bibliografía

[1] Sitio oficial de open mpi. http://www.open-mpi.org/.

[2] Mario Rubiales Gómez & Antonio Benítez Corbacho. *Video Digital, Guía Práctica para usuarios*. Guías prácticas. Ediciones Anaya Multimedia, 2004.

[3] Sitio oficial ffmpeg. http://www.ffmpeg.org/.

[4] Barry Wilkinson & Michael Allen. *Parallel programming : techniques and applications using networked, workstations and parallel computers*. Editorial Prentice Hall, 1999.

[5] Peter S. Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann Publishers, 1997.