Codificación de Vídeo MPEG-2 sobre una Red de Estaciones de Trabajo

Teresa Olivares Montes

Departamento de Informática.
Universidad de Castilla-La Mancha.
Campus Universitario s/n 02071 Albacete
teresa@info-ab.uclm.es

Resumen. Debido al volumen de datos utilizado en recientes aplicaciones y a la necesidad de almacenar, recuperar y transmitir imágenes de la mejor forma posible, la compresión de imágenes digital se está convirtiendo cada vez más en una técnica indispensable. Es difícil alcanzar compresión de vídeo en tiempo real, y muy caro, cuando el tamaño de la imagen es grande, o se necesita una alta calidad de la misma con un bajo caudal de datos. Por ello, el procesamiento paralelo, se convierte en la elección natural para obtener mejores tiempos de codificación. Veremos diversos trabajos realizados en codificación de vídeo en paralelo. Nos centraremos en uno de ellos y estudiaremos distintas técnicas de división de las imágenes entre los procesadores, con el objetivo de aumentar el rendimiento del sistema. Utilizaremos una red de estaciones de trabajo como la opción más atractiva en costo y rendimiento para esta aplicación.

1 Introducción

La transmisión de imágenes, con o sin movimiento, es una de las aplicaciones que consume más ancho de banda en la actualidad. El vídeo, como caso particular, se transmite casi invariablemente en forma comprimida. La esencia del proceso de compresión es intentar alcanzar una representación más compacta de la señal digital, mediante la eliminación de redundancias presentes en la misma, para minimizar el caudal de bits necesario para su transmisión o almacenamiento, intentando mantener la calidad. Los estudios de compresión de imágenes van orientados, generalmente, a analizar la calidad de la reconstrucción de la imagen comprimida, el factor de compresión alcanzado, y la complejidad y velocidad del algoritmo en sí.

Hay dos estándares aceptados actualmente para compresión de imágenes estáticas y en movimiento, son JPEG (Joint Pictures Expert Group) y MPEG (Moving Pictures Expert Group). Estos esquemas, proporcionan altos factores de compresión con buena calidad en la imagen reconstruida. Sin embargo, la cantidad de cálculos requeridos para ambos es elevadísima.

La idea básica detrás de la codificación MPEG es, explotar la localidad temporal. Exceptuando ciertos tipos de vídeos, como vídeos musicales, las imágenes no cambian mucho en pequeños intervalos de tiempo. MPEG, toma ventaja de esto,

codificando una imagen con relación a otras imágenes temporalmente cercanas a ellas [1]. Las principales técnicas usadas por **MPEG** y desarrolladas más adelante, son las siguientes [2]:

1) Transformación del espacio de color RGB, lo que produce automáticamente un factor de compresión de 2:1.
2) Codificación JPEG basada en DCT (Discrete Cosine Transform) y cuantización, seguidas de algún esquema de compresión sin pérdidas, que producen factores de compresión tan altos como 30:1, con una buena calidad de imagen.
3) Compensación de movimiento, en la que una imagen puede codificarse en términos de las imágenes previa y siguiente.

Sin embargo, estas técnicas limitan severamente la velocidad a la que una secuencia de imágenes puede comprimirse. El algoritmo de compresión MPEG es bastante asimétrico, es decir, la cantidad de cálculos necesarios para la codificación es mucho mayor que la necesaria para la decodificación. Algunos codificadores MPEG requieren de 15 a 20 minutos de cómputo para comprimir 1 minuto de vídeo. Por lo que es muy necesario obtener un tiempo de codificación que mejore los resultados de la versión secuencial. Una implementación software del codificador es más deseable, efectiva y flexible que algunas realizaciones con hardware de propósito especial, y además permite mejoras algorítmicas para dar nuevas soluciones.

Así, explotaremos el enorme poder computacional ofrecido por sistemas de computación paralelos con el codificador de vídeo MPEG-2 basado en software. Presentaremos cuatro métodos diferentes de distribución de datos sobre un grupo de procesadores, con paralelismo espacial y temporal, usando la librería MPI para las comunicaciones, sobre una red de estaciones de trabajo conectadas vía un conmutador ATM.

Se presentarán resultados de tiempos de codificación para diferente número de procesadores. Compararemos finalmente las técnicas de división de la imagen frente a la división de la secuencia con los mismos parámetros y secuencia de vídeo, obteniendo los mejores resultados con la técnica de división de la secuencia, por lo que se presentará una mejora de este último método. Acabando con prometedoras ideas que se van a realizar en un futuro inmediato

2 La codificación de vídeo

Una imagen, una secuencia de vídeo o las señales de audio se pueden comprimir debido a los siguientes factores:

a) Hay una considerable redundancia estadística en la señal
b) Hay bastante información en la señal, que es irrelevante desde el punto de vista perceptual humano.

Para una aplicación dada, los esquemas de compresión, pueden explotar uno o todos los factores anteriores, para alcanzar el factor de compresión de datos deseado.

Por la necesidad de establecer normas internacionales para estos esquemas de compresión, los organismos de estandarización mundiales han desarrollado diferentes estándares para el almacenamiento y transmisión de vídeo y su audio asociado. Algunos de ellos se muestran en la tabla 1 [2].

Tabla 1. Estándares de vídeo

JBIG	Joint Binary Image Group
JPEG	Joint Photographic Expert Group
Motion JPEG	Motion Photographic Expert Group
ITU H.261	Video Codec for Audiovisual Services at px64 Kbps
MPEG-1	Moving Picture Experts Group.Digital Storage Media up to 1.5 Mbps
MPEG-2	Moving Picture Experts Group. Generic coding of moving pictures and associated audio
MPEG-4	Multiple bit-rate Audiovisual coding up to 1024Kbps for video and up to 64Kbps for audio
MPEG-7	Multimedia Content Description Interface
ITU H.263	Expert Group on Very Low bit-rate Video Telephony
GA HDTV	Grand Alliance, FCC
DVB	Digital Video Broadcasting
CMTT	CMTT.723 and CMTT.721 Committee for Mixed Telephone and Television

2.1 JPEG

El estándar ISO/IEC 10918, "*Digital compression and coding of continuous-tone still images*" surge para responder a las necesidades de una norma internacional para la compresión de imágenes estáticas multinivel.

Su objetivo fue desarrollar un método general para compresión de imágenes que reuniese una serie de requisitos. Incluye dos métodos de compresión básicos: un método de compresión con pérdidas basado en la DCT, y un método predictivo para compresión sin pérdidas. La codificación de imágenes basada en DCT, es la base de todos los estándares de compresión de imágenes y vídeo.

Un sistema de codificación genérico basado en DCT se muestra en la siguiente figura:

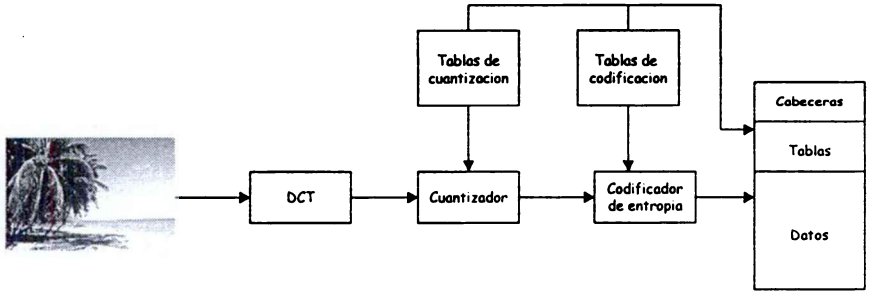

Figura 1. Diagrama de bloques del codificador JPEG

Especifica cuatro modos de operación: secuencial basado en DCT, progresivo basado en DCT, sin pérdidas y jerárquico.

Además de las imágenes estáticas, pueden codificarse también datos de vídeo usando Motion JPEG, que no emplea ninguna técnica de reducción de redundancia

entre imágenes. Además, cada imagen se codifica de forma independiente y el proceso de codificación produce un caudal de bits más alto [2], [10], [11].

2.2 MPEG-1

Es el primer Codec (codificador/decodificador) desarrollado por MPEG (Moving Pictures Expert Group). Se dirigía a aplicaciones que requieren calidad media y una codificación de vídeo y audio a un caudal de bits medio (sobre 1.5 Mbps).

Todos los estándares MPEG son genéricos, es decir, independientes de la aplicación. No especifican las operaciones del codificador. En cambio, especifican la sintaxis del flujo de bits codificado y el proceso de decodificación. Así, proporcionan bastante flexibilidad en las especificaciones, para que distintos vendedores puedan incluir elementos de optimización específicos.

El estándar MPEG aparece publicado en cuatro partes: *Systems, Vídeo, Audio y Conformance Testing*.

El algoritmo de codificación **MPEG-1**, es un esquema de compresión con pérdidas, que puede aplicarse a un amplio rango de formatos de entrada y de aplicaciones. Sin embargo, se ha optimizado para aplicaciones que soportan un caudal de bits continuo de alrededor de 1.5 Mbits/s, tales como un CD-ROM.

MPEG-1 no reconoce fuentes entrelazadas. El vídeo entrelazado, tal como el generado por una cámara de TV, debe convertirse a código no entrelazado antes de la codificación. Este proceso de conversión no está especificado en el estándar.

Una fuente de vídeo es una secuencia de frames numerados, F_1, ..., F_n. Cada frame es una imagen estática. Un reproductor de vídeo muestra las imágenes consecutivas, frame tras frame, usualmente a un caudal cercano a los 30 frames/s (el caudal de la TV estándar). Los frames son digitalizados en un formato RGB estándar, con 24 bits por píxel (8 para cada uno de los colores, rojo, verde y azul).

El algoritmo MPEG-1 opera con imágenes representadas en el espacio de color YUV. Por lo tanto, si una imagen se almacena en formato RGB, debe, primero, convertirse al formato YUV, donde toda la información de luminancia (Y) se retiene, con 8 bits por píxel. Sin embargo, la información de crominancia (U, V) se submuestrea 2:1 en ambas direcciones, vertical y horizontal, reduciéndose la información en un factor de 4:1. Así, hay 2 bits por píxel de información U, y 2 bits por píxel de información V. Este submuestreo no afecta de forma drástica a la calidad, ya que el ojo humano es más sensible a la información de luminancia que a la de crominancia. El submuestreo es una etapa con pérdidas. Los 24 bits por píxel de la información RGB se reducen a 12 bits de información YUV, que automáticamente da una compresión de 2:1.

Hubo dos requisitos fundamentales durante el desarrollo del estándar MPEG-1, la necesidad de alta compresión y la necesidad de capacidad de acceso aleatorio. La codificación *Intraframe* por sí sola, es lo mejor para conseguir acceso aleatorio, pero no reúne los requisitos de alta compresión. Para solventar este inconveniente, MPEG decidió, combinar las técnicas de compresión *Intraframe* e *Interframe*. Además, para mejorar el factor de compresión, se propusieron aplicar dos esquemas de codificación, el predictivo y el interpolativo.

Las funciones de compresión incluidas en MPEG son las siguientes:

1. Reducción de los factores de muestreo en los dominios espacial y temporal, tanto de los componentes de luminancia como de crominancia.
2. DCT basada en bloques para la codificación intraframe e interframe.
3. Compensación de movimiento basado en bloques para interframe interpolativo y predictivo.
4. Codificación Huffman para la compresión sin pérdidas de los vectores de movimiento y los coeficientes DCT cuantizados.

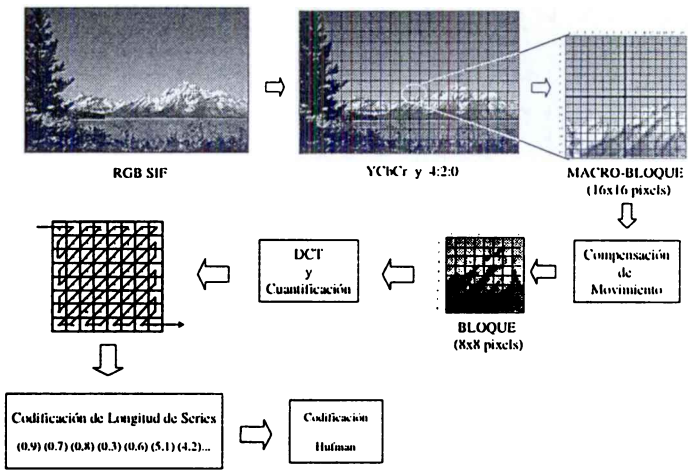

Figura 2. Etapas básicas del estándar MPEG-1

MPEG-1 define tres tipos diferentes de imágenes que ofrecen mayor flexibilidad al dilema planteado entre eficiencia en la codificación y acceso aleatorio.

- **Tipo I** (*Intra-frames*). Estas imágenes se comprimen usando codificación intraframe. Es decir, no hacen referencia a ninguna otra imagen del flujo de bits codificado. Proporcionan un rápido acceso aleatorio, pero ofrecen sólo compresión moderada.
- **Tipo P** (*Forward Predicted Frames*). Se codifican con relación a una imagen de referencia pasada, es decir, se codifican usando predicción de movimiento compensado de imágenes pasadas. Un frame de referencia es un frame de tipo P o de tipo I. Cada macrobloque en un frame P, puede codificarse como un macrobloque de tipo P, o como un macrobloque de tipo I. Un macrobloque de tipo **I**, se codificará como un macrobloque en un frame I. Un macrobloque de tipo **P**, es un área de 16x16 píxeles del frame de referencia pasado, más un término de error. La compresión para imágenes P es mejor que para las imágenes I, y pueden usarse como referencia para compensación de movimiento adicional.
- **Tipo B** (*Bidirectionally predicted frames*). Proporcionan el más alto grado de compresión. Se codifican con relación al frame de referencia pasado, al frame de referencia futuro, o a ambos, es decir, se codifican usando predicción de

movimiento compensado de otras imágenes I o P pasadas y/o futuras. La codificación para frames de tipo B es similar a los de tipo P, excepto que los vectores de movimiento se pueden referir a áreas en el frame de referencia futuro. Para macrobloques que usan ambos frames de referencia, el pasado y el futuro, las dos áreas de 16x16 se promedian. Como no se utilizan en la predicción de otras imágenes, pueden acomodar más distorsión y producir más compresión que las imágenes I o P. [18]

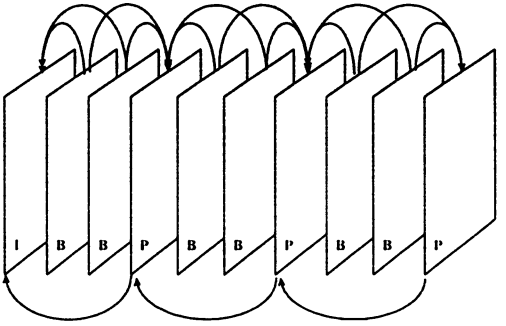

Figura 3. Frames I, P o B

Para especificar el área de 16x16 del frame de referencia, se transmite un *vector de movimiento*. Un vector de movimiento (0,0) indica que el área de 16x16 está en la misma posición que el macrobloque que está siendo codificado. Otros vectores distintos son relativos a esa posición. El término de error se transmite aplicándole la DCT, cuantización y codificación en series.

La búsqueda de los mejores vectores de movimiento (con términos de error más pequeños) es la clave principal de cualquier codificador de vídeo MPEG-1. Este es lo que hace al codificador inherentemente más lento que al decodificador

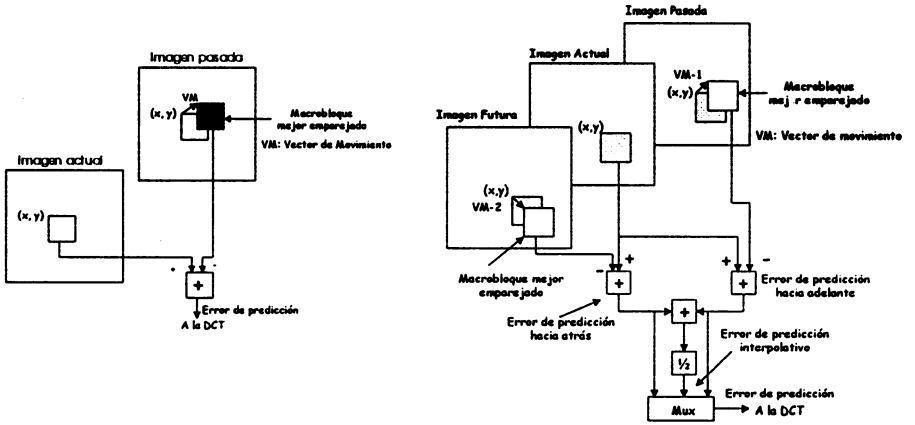

Figura 4. Compensación de movimiento hacia adelante (izquierda) y bidireccional (derecha)

Con respecto a la **sintaxis** especificada en el estándar, se tiene que una secuencia de vídeo MPEG-1 es un flujo ordenado de bits, con patrones de bits especiales que marcan el principio y final de una secuencia lógica. Cada secuencia de vídeo se compone de una serie de GOPs (Groups Of Pictures). Un GOP se compone de una secuencia de imágenes (frames). Un frame se compone de una serie de slices. Un slice se compone de una serie de macrobloques, y un macrobloque se compone de 6 bloques (4 de luminancia y 2 de crominancia) [2].

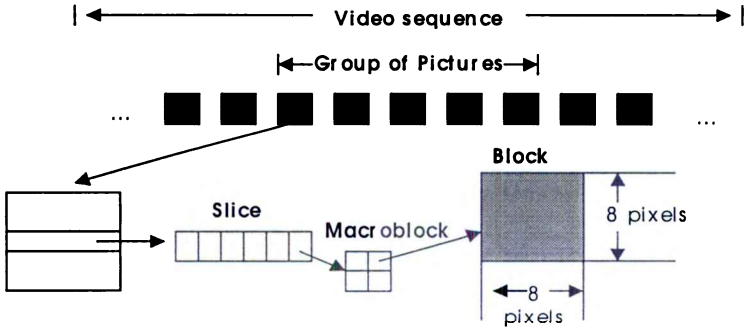

Figura 5.Estructura de una secuencia de vídeo

2.3 MPEG-2

MPEG-2, es la segunda fase del trabajo de MPEG. Su objetivo original fue definir un estándar genérico que pudiera soportar un amplio rango de aplicaciones, y soportar flujos de bits comprimidos a caudales cercanos a 5Mbits/s para calidad NTSC/PAL, o cerca de los 10 Mbits/s para calidad de vídeo de estudio. Entre los requisitos originales estaban: compatibilidad con MPEG-1, buena calidad de imagen, flexibilidad en el formato de entrada, capacidad de acceso aleatorio, rebobinado lento y rápido, y capacidad de movimiento lento, escalabilidad del flujo de bits, retrasos bajos en la comunicación y resistencia a errores.

Aunque poco después se plantearon las siguientes cuestiones:

- No había razón para restringir el caudal de bits máximo a 10 Mbits/s. MPEG debe soportar con éxito, caudales de bits más altos, por ejemplo, de 80 a 100 Mbits/s para aplicaciones de HDTV.
- Sería imposible definir un único estándar que satisficiera todos los requisitos.
- Muchas aplicaciones deben usar sólo un pequeño subconjunto de las características ofrecidas por el estándar. Por esto, MPEG, decidió adoptar una aproximación similar a la de una caja de herramientas, es decir, ofrecer MPEG-2 como una colección de herramientas definidas para satisfacer los requisitos de la mayoría de las aplicaciones específicas.

Proporciona, por tanto, una sintaxis estándar que ofrece una relación óptima entre costo y calidad. Por lo que está rápidamente emergiendo como el estándar preferido para compresión de vídeo a movimiento completo.

La sintaxis de la secuencia de bits se divide en subconjuntos conocidos como *Perfiles*, que especifican algunos requisitos de dicha sintaxis. Los perfiles se dividen

de nuevo en conjuntos de restricciones impuestas a los parámetros del flujo de bits, que se conocen como *Niveles*. Para cada perfil/nivel, MPEG-2 proporciona la sintaxis para el flujo de bits codificado y los requisitos de decodificación.

Un *perfil*, es un subconjunto definido de la sintaxis de un flujo de bits completo, especificada por MPEG-2. Los cuatro perfiles posibles son: Simple, Main, Main +, y Next. En un perfil, un *nivel*, se define como un conjunto de características impuestas a los parámetros del flujo de bits, tal como la resolución de la imagen o el caudal de bits máximo. Para cada perfil, los cuatro niveles son Low (para resolución SIF), Main (para CCIR-601), y High-1440 y High (para resolución HDTV). [19]

Además, MPEG-2 presenta dos espacios de color adicionales al 4:2:0, llamados:

Formato **4:4:4**, donde los planos de crominancia y luminancia son muestreados a la misma resolución.

Formato **4:2:2**, en el que el plano de crominancia se submuestrea a la mitad de resolución, en la dirección horizontal. [3]

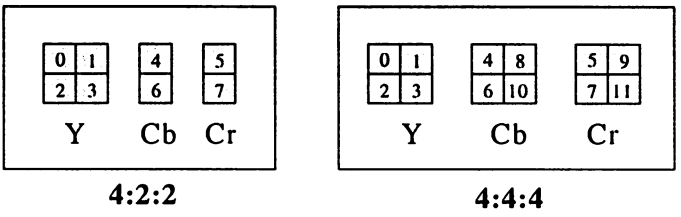

Figura 6. Estructura de los macrobloques en MPEG-2

MPEG-2 puede trabajar con vídeo entrelazado, ha mejorado la estimación de movimiento y permite el cálculo de vectores de movimiento a medio píxel. Además, toma también otras medidas para mejorar la calidad de la imagen, como son los cuatro modos de escalabilidad (Espacial, Temporal, SNR, y Data partitioning), que priorizan los datos de vídeo formando capas de vídeo comprimido, transmitiendo los datos más importantes con mayor seguridad en entornos provistos de errores. [4]

2.4 MPEG-4

El estándar MPEG-4 es el actual proyecto del grupo MPEG, que esta siendo desarrollado para aplicaciones de comunicaciones multimedia, a muy bajo caudal.

Una aplicación típica de este estándar sería la transmisión de imágenes en movimiento por Internet, a bajos caudales, y su interoperatibilidad con las aplicaciones del Web. Una de las cuestiones pendientes es la comunicación eficiente de vídeo y audio. El estándar MPEG-4 debe ser capaz de proporcionar vídeo y audio con cualquier tipo de interactividad con el usuario. En este sentido el grupo MPEG, en julio de 1993 se fijó el objetivo de desarrollar un estándar para codificaciones audiovisuales en aplicaciones multimedia que permitan interactividad, elevada compresión, escalabilidad del vídeo y del audio y soporte para contenidos de vídeo y audio tanto reales como sintéticos. Este estándar fue designado con el número 14496 por el ISO y se esperaba finalizar a finales de 1998

MPEG-4 realiza una codificación basada en objetos. Define una escena audiovisual como una representación codificada de *objetos audiovisuales* que tienen cierta relación en el tiempo y en el espacio. La noción de objeto permite mezclar objetos naturales y sintéticos (generados por un computador tales como gráficos o texto).

Al igual que MPEG-1 y MPEG-2, MPEG-4 también realiza una representación por capas de la información. Cada cuadro de vídeo se segmenta en un número de objetos, llamados *planos de objeto de vídeo* (VOP). Sucesivos VOPs perteneciendo a un mismo objeto físico se denominan objetos de vídeo (VO). Un VO puede ser visto en MPEG-4 como un GOP en los estándares MPEG-1 y MPEG-2. La forma, el movimiento y la información de textura de los VOPs pertenecientes a un mismo VO son codificados en una capa separada de objeto de vídeo (VOL). Información adicional necesaria para identificar cada una de las VOLs y como se componen los diferentes VOLs son también codificados. Esto permite la descodificación selectiva de VOPs y proporciona también una escalabilidad a nivel de objeto en el decodificador.

Para la codificación de la forma de los VOPs se suele recurrir a técnicas de compresión sin pérdidas, aunque también se da la posibilidad de aplicar codificación con pérdidas para obtener mayores niveles de compresión. [5]

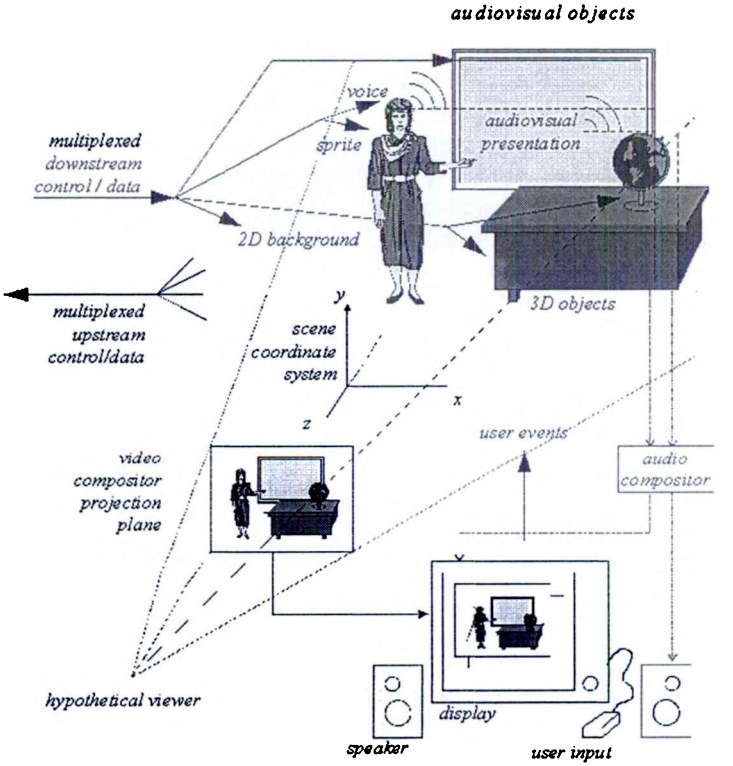

Figura 7. Un ejemplo de escena MPEG-4

3 Motivación al paralelismo

3.1 Primeros intentos

Se han realizado algunas aproximaciones, muchas de ellas usando hardware especial, para paralelizar las operaciones del Codec con las secuencias de vídeo. Por ejemplo, la codificación de vídeo en movimiento completo en CD-I (no estándar) se ha implementado en un computador paralelo. Emplea una aproximación donde el paralelismo se consigue dividiendo las etapas del Codec en tareas, y asignando una tarea a un grupo de procesadores. Esto tiene el siguiente problema, deben leerse muchas imágenes antes de que todos los procesadores tengan algunas tareas que ejecutar. Además, esta implementación requiere hardware de propósito especial.

El codificador original se desarrolló en un computador VAX-8800. Cuanto más larga era la secuencia de vídeo a comprimir, más lento era el sistema, por lo que se decidió llevar el algoritmo a un computador paralelo [6]. Como la etapa de preprocesamiento consume una pequeña parte del tiempo de codificación, no se incorpora en la versión paralela, centrándose por tanto en las dos etapas centrales del proceso: Estimación de movimiento y Compresión de vídeo. La máquina que se utilizó fue la **POOMA** (*Parallel Object Oriented Machine*), un computador paralelo experimental desarrollado en *Philips Research.* Es una máquina con una arquitectura MIMD débilmente acoplada. El prototipo está compuesto por 100 nodos. Un nodo consta de un procesador de datos (DP), memoria, un procesador de comunicaciones (CP) e interfaces de E/S.

Los programas para esta máquina, se escriben en POOL (*Parallel Object Oriented Language*), un lenguaje de programación diseñado en el proyecto POOMA, en C extendido con sistema operativo para creación de procesos, comunicación y sincronización. El codificador original, se dividió en tareas, asignando una tarea a cada procesador. Se realizó un test para evaluar la implementación paralela del codificador, con una secuencia de vídeo de 50 segundos y frecuencia de 30 Hz a pantalla completa. La resolución es de 352x240 píxeles para las imágenes de luminancia

Una prueba en el VAX-8800 es 1650 veces más lenta que en tiempo real. El procesador de datos del POOMA es un 68020 de Motorola. Para la codificación planteada, este procesador es 2.75 veces más lento que el Vax. Pero un programa que use de forma óptima los 100 procesadores disponibles, podrá obtener un factor de velocidad de 100/2.75=36. La medida actual es de 32 (89% del caso teórico). Esto corresponde a una velocidad 51 veces más lenta que el tiempo real, o menos de 2 imágenes por segundo. Analizando las causas de estas diferencias se llega a la conclusión de que es bastante duro hacer cualquier mejora.El significado práctico del incremento de velocidad es que una aplicación de CD interactivo, por ejemplo, tiene una duración aproximada de 30 minutos. Para procesar esta cantidad de vídeo en un Vax se necesitaría un mes, mientras que el POOMA sólo necesita un día.

3.2 Situación actual

Actualmente, el amplio campo del procesamiento paralelo está causando un gran impacto en áreas tales como las comunicaciones de datos visuales y de audio. Se han

desarrollado muchos tipos de técnicas de procesamiento paralelo que incluyen memoria distribuida y compartida, procesamiento débil y fuertemente acoplado, conmutación de paquetes y paso de mensajes, grano fino y grueso, y distintas topologías.

Por lo tanto, la idea de almacenar y transmitir grandes cantidades de datos digitales audiovisuales, lleva a la necesidad de alto poder computacional y, en consecuencia, al uso del procesamiento paralelo en el campo de reducción de datos.

Una secuencia de vídeo puede verse como una señal tridimensional, dos dimensiones en el dominio espacial y una en el dominio temporal. Así, pueden usarse dos aproximaciones diferentes en la paralelización del algoritmo de codificación, la espacial y la temporal.

Con el **paralelismo espacial**, cada imagen de una secuencia de vídeo se divide en partes y cada parte se asigna a un elemento de proceso. La secuencia de vídeo se procesa imagen a imagen. El retardo en los resultados es mínimo, pero, evidentemente, este procedimiento de disminuir el tiempo de cómputo aumentando el número de procesadores, tiene un límite, debido a la resolución espacial limitada de una secuencia de vídeo.

Con **paralelismo temporal**, diferentes elementos de proceso, procesan diferentes imágenes de vídeo de una secuencia. No hay límite superior en el número de procesadores, pero hay un retardo en el rendimiento total.

En los últimos años, se han realizado diversas aproximaciones para implementar un codificador MPEG en paralelo. Posiblemente a nivel software, la más destacable, es el codificador desarrollado por la Universidad de Berkeley, en California. [1],[7].

Se han usado diferentes métodos para incrementar la velocidad de ejecución, y los avances más importantes han estado orientados en dos direcciones:

1) Distribuir la tarea de compresión entre varios procesadores, bien sean éstos los componentes de un cluster de estaciones de trabajo, o de un multicomputador.

En esta línea hay que destacar los trabajos de la Universidad de Purdue, que han llevado el codificador paralelo de Berkeley al Paragon de Intel, obteniendo mejoras significativas [8]

El codificador paralelo basado en Split-C, desarrollado inicialmente sobre el CM-5 y pensado para ejecutarse sobre redes de estaciones de trabajo conectadas mediante redes de alta velocidad conmutadas, dentro del proyecto NOW de la Universidad de California, en Berkeley [9].

Por otro lado, la distribución de tareas, se puede realizar, tal y como plantean Yu y Anastassiou, sobre un grupo de estaciones de trabajo SUN, conectadas mediante Ethernet, obteniendo un caudal de ejecución de 4.7 imágenes/s. [10].

También hay que destacar los trabajos realizados para codificar datos de audio. Uno de los más recientes es el realizado en la Universidad de Miami en la codificación en paralelo de MPEG-1, utilizando un array de procesadores conectados en malla y con estructura cúbica [16].

2) Utilizar aceleradores hardware específicos de esta aplicación. Un ejemplo de esto, es el desarrollado por Texas Instruments, el Procesador de Vídeo Multimedia (MVP) [15]. Este acelerador comprime secuencias de vídeo con formato CIF en tiempo real. Existiendo también actualmente en el mercado otras implementaciones del mismo tipo.

3.3 La codificación de vídeo MPEG-2 en paralelo

Vamos a describir ahora el codificador paralelo de la Universidad de Hong-Kong, como modelo de referencia para nuestros estudios.

No se emplea ningún hardware de propósito especial, ni primitivas de programación, sino que es completamente portable, flexible y escalable. La implementación se realizó en el Paragon, así como en el iPSC/860 de Intel, y en varios tipos de redes de estaciones de trabajo [12].

La implementación paralela del MPEG-2 se ha llevado a cabo utilizando un paradigma de programación SPMD o *data-parallel* [20]. Este paradigma escrito en C, bajo *Express*, en la primera versión y en MPI posteriormente, permite que el software sea portable a través de un amplio rango de arquitecturas, desde sistemas escalables de alto rendimiento, hasta redes de estaciones de trabajo de alta velocidad.

Para hacer la implementación de MPEG-2 escalable, se asume que la topología de procesadores adoptada es una rejilla bidimensional. A los procesadores se le asignan las coordenadas x-y, que facilitan la comunicación entre ellos, e identifican los procesadores vecinos. Los datos se mapearán en esa rejilla virtual de procesadores. Para una computación paralela eficiente, es muy importante el diseño de las estructuras de datos distribuidas. Este diseño debe hacerse desde el punto de vista de balancear lo más posible la carga computacional en los procesadores individuales, y de ganar el máximo paralelismo de las comunicaciones de datos entre procesadores [13].

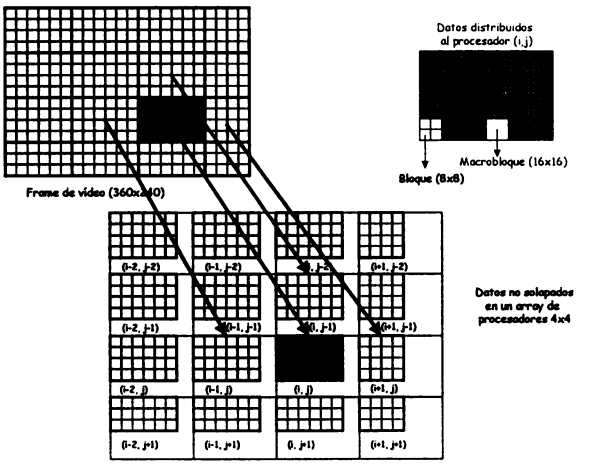

Figura 8. Distribución de un frame en la topología de red virtual

Los datos de un frame se distribuyen entre los procesadores. Debido a la inevitable comunicación, que será el factor limitante de la velocidad de codificación, hay que tener cuidado con el reparto de datos entre los procesadores, como factor fundamental en la posterior comunicación. La distribución de datos entre los procesadores, es bastante simple. Se reparte un frame entero, tan uniformemente como sea posible. Es posible partir los datos, repartiendo la parte de requisitos del

frame de datos a los procesadores correspondientes, tal como son mapeados en la rejilla bidimensional, como se muestra en la figura siguiente.

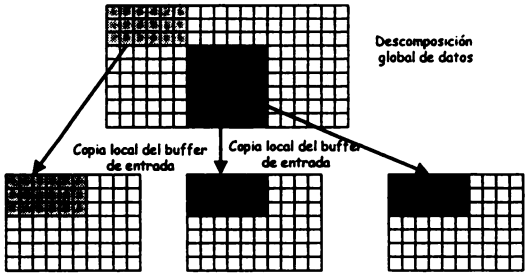

Figura 9.Reparto de datos sin solapamiento

Pero, en este caso, es necesaria la comunicación entre los nodos, conforme la ventana de búsqueda se mueve por los bordes. (Fig.10)

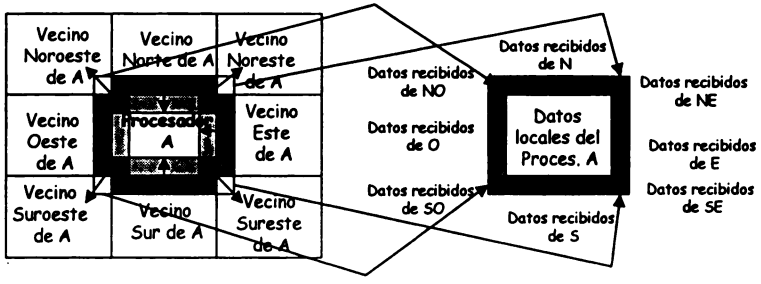

Figura 10. Flujos de datos de un procesador a sus procesadores vecinos

Como cada procesador tiene suficiente memoria como para almacenar la ventana de búsqueda completa, es posible eliminar esta enorme cantidad de comunicación, distribuyendo los datos del frame de forma solapada entre los procesadores (Fig.11). A cada procesador se le asignan algunos datos redundantes, necesarios para formar el área de búsqueda completa.

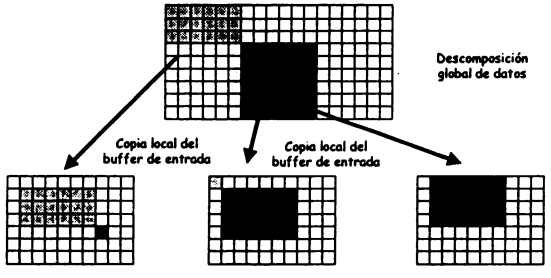

Figura 11.Reparto de datos con solapamiento

Los experimentos se realizaron en el Paragon de Intel, variando el número de procesadores. El tiempo medido se promedió sobre 50 frames de una secuencia de vídeo. Se utilizaron 5 secuencias: *Football* (360x240), *Table Tennis* (360x240), *Salesman* (360x288), *Miss America* (352x288) y *Swing* (352x288). Todas estas secuencias son representativas de distintas clases de movimiento y muy útiles para probar la estimación de movimiento.

Como el tiempo para procesar los 50 frames de la secuencia no eran los mismos en cada procesador, se tomó la media sobre todos los procesadores. Algunos de tales conjuntos de medidas se hicieron usando 1, 2, 4, 8, 16, 32, 64, 128, 256 y 330 procesadores para cada conjunto.

Se usó un caudal constante de 5 Mbps, un GOP de 12, la ventana de búsqueda de +/- 11 píxeles, para imágenes de tipo P, y de +/- 10 píxeles para imágenes de tipo B. Se utilizó tanto la búsqueda completa como la logarítmica.

Algunas de las conclusiones que se deducen de estas primeras pruebas son, que la etapa de estimación de movimiento es la que consume más tiempo, que para obtener una codificación más rápida hay que emplear búsqueda logarítmica, y otros resultados de interés que demuestran las mejoras con respecto a implementaciones anteriores.

Los primeros estudios, además de estudiar varias estrategias de optimización del código, iban encaminados a realizar distintas pruebas con algoritmos de estimación de movimiento diferentes y a utilizar otras librerías de paso de mensajes tales como PVM y MPI (*Message-Passing Interfaz*).

Veamos algo más sobre el rendimiento de un codificador de vídeo software con calidad MPEG-2 en varias plataformas paralelas y distribuidas, que incluyen el Paragon XP/S de Intel, el computador paralelo hipercubo iPSC/860, así como varias redes de estaciones de trabajo: Un cluster de estaciones de trabajo HP, un cluster de estaciones de trabajo SGI, y un cluster de estaciones de trabajo SUN.

Su rendimiento se escala de acuerdo al número de procesadores disponibles. Además, el codificador permite controlar varios parámetros tales como el tamaño de la ventana de búsqueda, gestión del buffer, y caudal de bits. Los resultados incluyen comparaciones de tiempos de ejecución, velocidades, y caudales de codificación de frames en varios sistemas.

Las redes de estaciones de trabajo, utilizan entornos de comunicación tales como Express, PVM y MPI [14], y se están convirtiendo cada vez más, en plataformas más rápidas y baratas.

El objetivo al desarrollar este codificador fue conseguir un algoritmo de compresión de vídeo independiente de la arquitectura, sin comprometer la velocidad de procesamiento. La idea es, no emplear ningún hardware de propósito especial, ni primitivas de programación concretas, para hacerlo completamente portable, flexible y escalable.

Bajo el paradigma SPMD los datos se parten en pequeñas piezas que se asignan a procesadores diferentes. Un único programa se escribe en todos los procesadores, que lo ejecutarán asíncronamente para sus datos locales.

Con el objetivo de alcanzar velocidad en la estimación de movimiento, se realizó búsqueda logarítmica 2D, y se registró el correspondiente rendimiento. Para medir la calidad del vídeo, se utilizó la PSNR (Peak signal to Noise Ratio). Valores grandes de la misma indican mejor calidad.

Señalar, que no hubo pérdida de rendimiento debido a la paralelización, ya que, en el modelo de programación utilizado, varios módulos de MPEG-2, en cada procesador, usan la misma lógica computacional que la usada en la versión secuencial. La magnitud de la PSNR, depende del algoritmo de estimación de movimiento y debe mejorarse con una técnica de búsqueda exhaustiva.

Los resultados se muestran en función del caudal de codificación de frames en distintas estaciones de trabajo, y distintas gráficas de velocidades de varios módulos computacionales, así como la velocidad total [13]

El crecimiento lineal de las curvas para los distintos módulos del algoritmo de codificación, demuestra que el utilizar más procesadores, aumenta la velocidad de procesamiento de la imagen. Igual que la curva para la computación total, crece casi linealmente, indicando que con más procesadores, se requerirá menos tiempo para codificar un frame.

Si se utilizan más procesadores y se divide la carga de trabajo asignada a cada procesador, se pueden codificar frames de mayor tamaño en la misma cantidad de tiempo.

Comparando, por ejemplo, el caudal de codificación de frames en una red de estaciones de trabajo SUN, con un cluster SGI, se puede ver, que el rendimiento del cluster SUN es mucho más bajo. Un simple cálculo pone de manifiesto que se necesitará una red de 230 estaciones de trabajo SUN de propósito general, para vídeo en tiempo real.

El rendimiento del cluster HP es comparable al cluster SGI, y mejor que el cluster SUN. Haciendo una basta estimación, con una red de 124 estaciones de trabajo HP, se alcanza codificación de vídeo en tiempo real.

En el iPSC/860 no son tan prometedores los resultados como en el Paragon. La naturaleza lineal de las curvas indica, que puede alcanzarse procesamiento en tiempo real usando unos 452 procesadores del hipercubo.

Los resultados indican que varios componentes de MPEG-2 pueden paralelizarse muy eficientemente. También se comparan clusters de estaciones de trabajo con máquinas paralelas. Ya que el alto rendimiento de las estaciones de trabajo crece rápidamente, la codificación de vídeo es posible, con un gran número de tales estaciones de trabajo y además optimizando el software.

4 Análisis de diferentes técnicas de distribución de datos

Con el codificador que acabamos de definir como base [17], vamos a describir primero tres métodos de reparto de datos diferentes. Con estos tres métodos vamos asignar trozos diferentes del mismo frame a diferentes procesadores.

Después veremos una nueva técnica en la que dividimos la secuencia completa entre los procesadores y compararemos los resultados en los cuatro métodos.

Por último, en vista de los resultados del cuarto método, lo vamos a refinar aún más para conseguir mejores resultados.

4.1 División del frame

El primer estudio presenta una comparación de tres métodos diferentes a nivel de codificación de frames en paralelo. Los datos deben distribuirse entre los procesadores tal que cada procesador codifique uno o más macrobloques.

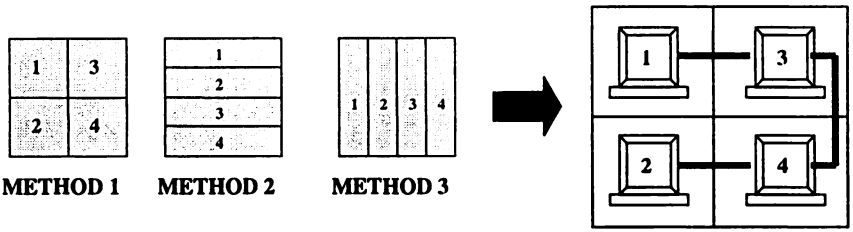

Figura 12. Diferentes métodos de división del frame

Tendremos el tamaño del frame igual a MxN (en píxeles) donde M=hxtamaño_Macrobloque y N=vxtamaño_Macrobloque, con tamaño_Macrobloque=16 y h, v serán el número de macrobloques y slices, respectivamente. El tamaño de la rejilla de procesadores 2-D será mxn. El conjunto de procesadores que tendremos será ((Pij)), i= 1, 2, ...k, ...m, j= 1, 2, ..., l, ...n. Y los siguientes valores:

- Píxeles no asignados en la dimensión m: $\alpha = MOD(M, (mx16))$
- Píxels asignados a cada procesador en la dimensión m: $\beta = (M-\alpha)/m$
- Píxeles no asignados en la dimensión n: $\gamma = MOD(N, (nx6))$
- Píxeles asignados a cada procesador en la dimensión n: $\delta = (N-\gamma)/n$
- De ésto: $\alpha = sx16$ y $\gamma = tx16$, **s** y **t** son enteros positivos, los macrobloques no asignados en cada dimensión.
- Los procesadores se numeran en orden consecutivo por columnas, denotaremos ese número por **r**.

4.1.1 Método 1: división en bloques

Para este método el tamaño local del frame de datos en el procesador Pij se determina de la siguiente manera [21]:

En la actual implementación, el tamaño local de datos es, el tamaño calculado anteriormente, más algunos datos solapados (necesarios para formar la ventana de búsqueda). Esto se hace de forma similar en los siguientes métodos.

$$\text{Local-size at Pij} = \begin{cases} (\beta+16) \times (\delta+16) & \text{if } i \le k, j \le l \\ \beta \times (\delta+16) & \text{if } i > k, j \le l \\ (\beta+16) \times \delta & \text{if } i \le k, j > l \\ \beta \times \delta & \text{if } i > k, j > l \end{cases} \quad \begin{matrix} k \le s \\ l \le t \end{matrix}$$

4.1.2 Método 2: División horizontal del frame

Para este método tendremos: α=0, β= M, γ = MOD (N, (mxnx16)), δ = (N-γ) / (mxn), γ = tx16, ya que el tamaño local de datos para todos los procesadores será M × una fracción de N, M × (N/m×n). N se dividirá ahora entre todos los procesadores y no sólo entre los procesadores situados en la dimensión n de la rejilla de procesadores. Así, el tamaño local del frame en el procesador Pij se determina como sigue:

$$\text{Local-size at Pij} = \begin{cases} \beta \times (\delta+16) & \text{if } r < t \\ \beta \times \delta & \text{if } r >= t \end{cases}$$

4.1.3 Método 3: división vertical del frame

Para este método los nuevos valores son:

α=MOD(M, (mxnx16)), β = (M-α) / (nxn), γ= 0, δ=N, α = sx16. Ahora M se dividirá entre todos los procesadores. Así, el tamaño local de datos en el procesador Pij se puede calcular como:

$$\text{Local-size at Pij} = \begin{cases} (\beta+16) \times \delta & \text{if } r < s \\ \beta \times \delta & \text{if } r >= s \end{cases}$$

4.1.4 Resultados Experimentales

Hemos usado los primeros 16 frames de una secuencia de vídeo de resolución CCIR-601, llamada *football*. Para los tres métodos tenemos el caudal de codificación (frames/segundo) con operaciones de entrada/salida (*fps-I/O*) y sin operaciones de entrada/salida (*fps*), para diferente número de procesadores (1, 2, 4, 8 y 16). También los tiempos de codificación por frame, con operaciones de entrada/salida *(ET-I/O)* y sin operaciones de entrada salida *(ET)*

El tiempo no fue el mismo en cada procesador, por lo que se tomó el promedio de los tiempos dados por todos los procesadores. Como diferentes trozos de un mismo frame pueden presentar diferentes características, el tiempo de codificación no es el mismo para cada procesador, por lo tanto, los tiempos de codificación en los procesadores son diferentes con esta distribución estática. Cuando tomemos un nuevo esquema dinámico, todos los procesadores darán resultados similares a los promedios expuestos aquí. En las siguientes gráficas presentamos los tiempos de codificación por frame, con y sin operaciones de entrada salida.

Como se puede ver no hay mucha diferencia entre los resultados obtenidos con estos tres métodos, y además, los resultados obtenidos con 4, 8 y 16 procesadores, con operaciones de E/S son muy similares. Esta es la principal motivación para buscar otra idea que mejore los resultados en el proceso de distribución de datos. Vamos a

cambiar nuestra división espacial por un paralelismo temporal, con el desafiante objetivo de mejorar estos primeros resultados.

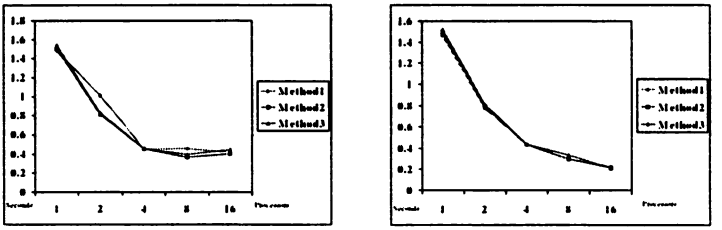

Figura 13. ET-I/O (izquierda) y ET (derecha)

4.1.5 División de la secuencia

En esta sección describimos otro método para distribuir el vídeo entre los procesadores. Consideremos los siguientes valores: *nframes* es el número de frames en la secuencia, *N*, el número de frames en un GOP, *ngops* será nframes/N, *mxn* el tamaño de la rejilla de procesadores 2D y:

- α = MOD (ngops, (m×n)), número de GOPs no asignados
- β = (ngops - α) / (mxn), número de GOPs asignados a los procesadores
- Los GOPs asignados al procesador Pij, con rango(pij) = r, serán calculados como:

$$ng = r + (k \times m \times n)$$

Con las siguientes condiciones:

si $\alpha = 0$ entonces k = 0 ... β-1

si $\alpha \neq 0$ entonces

si $r < \alpha$ entonces k = 0 ...β

sino si $r \geq \alpha$ then k=0 ... β-1.

Después de ésto, los cálculos para obtener los frames asignados a cada procesador, *nf*, son los siguientes, con N = número de frames en un GOP. Si ng = x entonces *nf = x, ..., (x+N-1)*

4.1.6 Resultados experimentales

Los experimentos comparan los cuatro métodos descritos con 96 frames de una secuencia de vídeo CCIR-601 para 1, 2, 4, 8 and 16 procesadores, calculando resultados de caudal de codificación de frames con y sin operaciones de E/S y mostrando gráficamente los correspondientes tiempos de codificación por frame, con y sin operaciones de E/S.

Podemos ver como el último método da mejores resultados que los otros, la principal diferencia está en el proceso completo, teniendo en cuenta operaciones de E/S, proceso de lectura de los frames, aspecto fundamental en paralelismo.

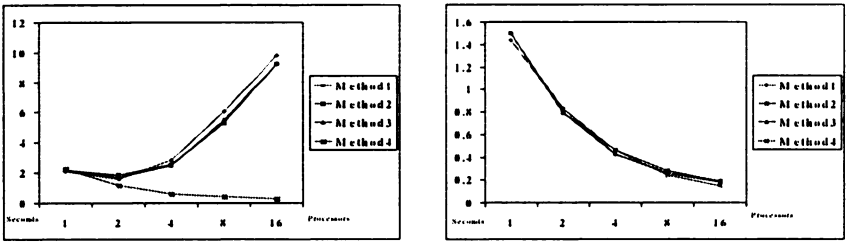

Figura 14. ET-I/O (izquierda) y ET (derecha)

4.1.7 Método de división de la secuencia mejorado

Ya que obtenemos los mejores resultados con el último método, vamos a mejorarlo un poco más. La idea es que todos los procesadores tengan los GOPs que le correspondan en sus discos duros locales y así puedan leer los frames de su disco local en lugar de leer del disco servidor donde está almacenada la secuencia completa. Así en este nuevo método (método4 local), es necesario copiar previamente del disco servidor al disco local los frames que le tocan a cada uno, para después empezar el proceso de codificación.

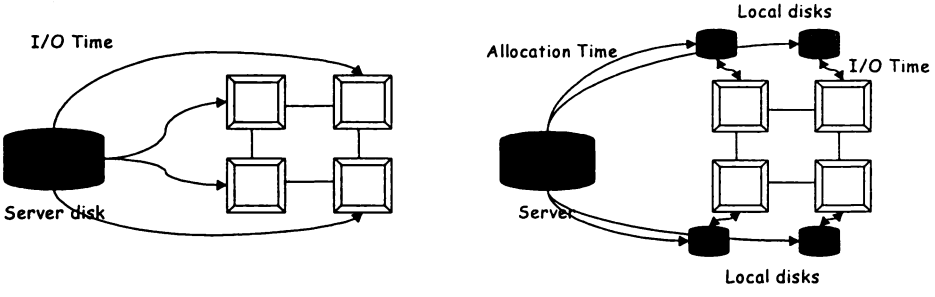

Figura 15.Método 4 (izquierda) y Método 4 local (derecha)

En este nuevo método, el reparto de frames del disco servidor al disco local de cada procesador, puede realizarse de dos formas diferentes:

1. Con un procesador dedicado a la distribución de los frames de vídeo a todos los procesadores
2. Todos los procesadores toman los frames que le correspondan del disco servidor y los almacenan en sus discos locales

Los diagramas de bloques para estas dos posibilidades, muestran como la primera opción necesitará una elevada comunicación entre los procesadores y además, los procesadores estarán esperando recibir sus frames (fig.16 izquierda). Con la segunda opción los procesadores están trabajando, sin necesidad de comunicarse entre ellos y sin paradas ni esperas (Fig.16 derecha).

Por lo tanto, tomaremos la segunda opción donde cada procesador va a calcular los frames que le toca codificar de forma similar al método 4, después cada uno de ellos va a leer del disco servidor los ficheros con los frames adecuados, los escribirán

en sus discos duros locales y después empezarán el procesamiento de los frames. Podemos justificar esta elección después de leer los resultados obtenidos al elegir la opción 1 en [22].

Así, se tiene para el *método 4* los siguientes valores de tiempos:

Tiempo total = Tiempo de procesamiento

Tiempo de procesamiento = Tiempo de E/S + Tiempo de codificación

Mientras que para el *método 4 local* se tendrá lo siguiente:

Tiempo total = Tiempo de reparto + Tiempo de procesamiento

Tiempo de procesamiento = Tiempo de E/S + Tiempo de codificación

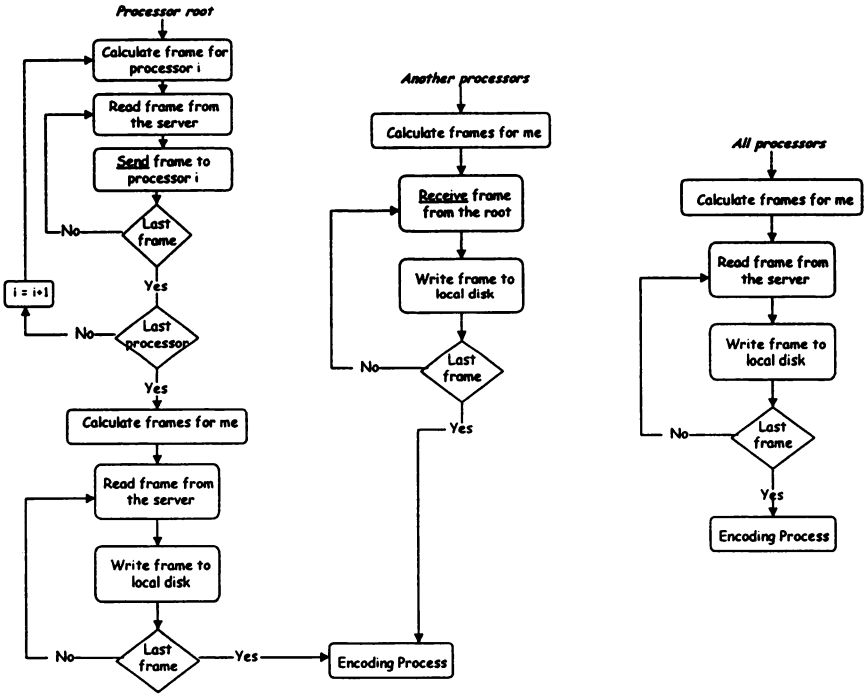

Figura 16. Diagrama de bloques para la opción 1 (izquierda) y para la opción 2 (derecha)

Comparemos estos tiempos (en segundos) sólo para un frame en un procesador:

Método 4	Método 4 local
Tiempo total = T. Procesamiento = 1.27	Tiempo total = 1.5281
Tiempo de E/S = 0.023	Tiempo de reparto = 0.2627
Tiempo de codificación = 1.2460	Tiempo de procesamiento = 1.2655
	Tiempo de E/S = 0.0162
	Tiempo de codificación = 1.2493
(A)	(B)

A primera vista, tenemos una ganancia de tiempo en la E/S para el Método 4 local y una pérdida de tiempo en el reparto de frames a los discos locales. Esta ganancia es más pequeña que la pérdida de tiempo. Por otro lado, los dos tiempos de procesamiento son muy similares. Fijémonos en la siguiente ecuación:

T. procesamiento – T. E/S (A) > T. procesamiento – T. E/S (B) (1)

En este caso, esta ecuación no se cumple. Bajo este esquema, el conflicto de los procesadores para acceder al disco servidor produce importantes pérdidas de tiempo. Sin embargo, si el proceso de adquisición de frames se separa de la etapa de procesamiento, entonces, los procesadores no sufrirán ningún tipo de conflicto una vez que la etapa de procesamiento haya empezado.

Comparemos las diferencias en el tiempo total, con diferente número de procesadores (1, 2, 4, 8, y 16) sobre una secuencia de vídeo de 96 frames. Y veamos también, las diferencias entre los tiempos de E/S.

TiempoTotal

Procesadores	1	2	4	8	16
Método 4	215.82	113.345	61.7925	43.61	26.0056
M4local	254.89	145.524	71.8999	42.2316	14.9054

Tiempo de E/S

Procesadores	1	2	4	8	16
Método 4	71.978	33.5475	16.3388	20.6407	10.3854
M4local	20.121	7.0181	3.0455	0.8869	0.2739

Podemos ver como los tiempos de reparto van a ser menores para el Método 4 local conforme aumenta el numero de procesadores, lo que implicará una reducción del tiempo total. Y como , los tiempo de E/S son mucho más pequeños para el Método 4 local que para el Método 4.

Si calculamos la diferencia media de los tiempos totales, será de 13.607 segundos menos para el Método 4. Pero si calculamos la diferencia media de los tiempos de E/S, será de 24.309 segundos menos para el Método 4 local.

Para más de cuatro procesadores, el Método 4 local, obtiene mejores tiempos que el Método 4, así las diferencias de tiempos sólo se mantienen para los casos con 1, 2, y 4 procesadores. En estos tres casos la diferencia de tiempos totales será de 26.8382 segundos menos para el Método 4, pero la diferencia de tiempos de E/S será de 30.5599 segundos menos para el Método 4 local. Y la anterior ecuación (1) será verdad para todos los casos.

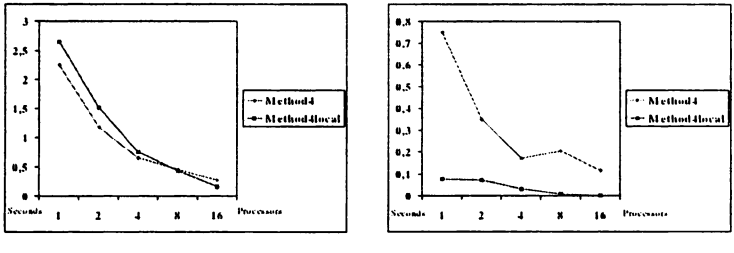

Figura 17. Tiempos totales por frame (izquierda) y tiempos de E/S (derecha)

Podemos decir, por tanto, que en un sistema distribuido actual formado por una red de estaciones de trabajo codificando una secuencia de vídeo completa, el Método4 local será una buena elección y se podrán obtener muy buenos resultados. En las siguientes gráficas mostramos el tiempo de codificación por frame y el tiempos de E/S por frame, calculados de la codificación de una secuencia de 96 frames.

4.1.8 Resultados experimentales

Comparemos ahora estos dos últimos métodos con una secuencia de vídeo CCIR-601 de 96 frames con diferente número de procesadores (1, 2, 4, 8, y 16). Mostramos en las siguientes gráficas los correspondientes tiempos de codificación con operaciones de E/S, *ET-I/O*, y sin operaciones de E/S, *ET*.

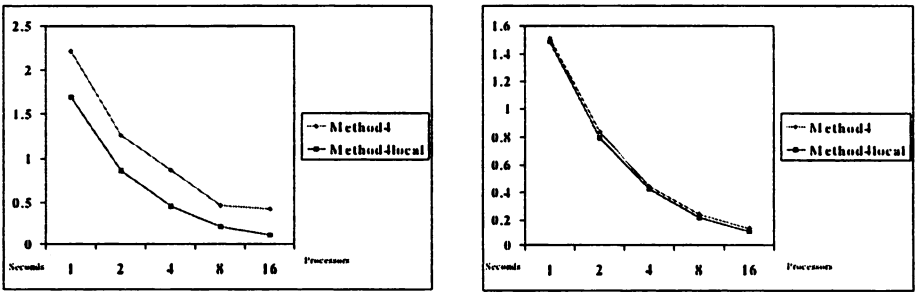

Figura 18. ET-I/O (izquierda) y ET (derecha)

Podemos ver cómo la diferencia es más significativa con operaciones de E/S y que ahora esa diferencia es más importante. Este cambio en la forma de leer los frames ha sido bastante buena y hemos obtenido una disminución del tiempo de codificación final.

En la siguiente figura mostramos el *Factor de Mejora (FOI)* producido en los tiempos de codificación con operaciones de E/S y sin operaciones de E/S para los dos métodos estudiados. También se calcula el *speedup* producido, mostrando la ganancia de la versión paralela con los dos métodos anteriores frente a la ejecución secuencial con sólo un procesador, distinguiendo también entre utilizar o no operaciones de E/S.

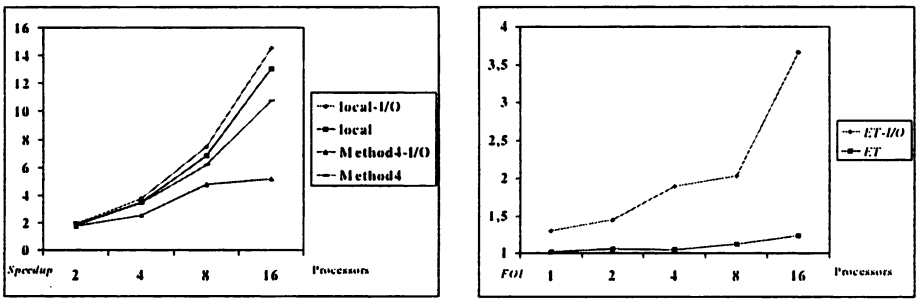

Figura 19. Speedup (izquierda) y Factor de Mejora (derecha)

5 Conclusiones

Presentamos en este trabajo un recorrido por los estándares de compresión de más reciente actualidad, generados por la incipiente demanda producida por el nacimiento de multitud de nuevas e interesantes aplicaciones multimedia. El estudio de estos estándares nos muestra la complejidad computacional del proceso de compresión, abriendo una senda por el procesamiento en paralelo, que nos lleva a encontrar diversos trabajos realizados en este asunto. El más reciente e interesante a nuestra vista fueron los realizados en la Universidad de Ciencia y Tecnología de Hong-Kong sobre la codificación en paralelo de estándar MPEG-2. Con ellos hemos podido trabajar, ya que planteaban una gran cantidad de tareas pendiente aún, y el aspecto elegido para mejorar y trabajar a fondo fue la distribución o reparto de la carga, de la secuencia de video a codificar entre los procesadores. Se ha presentado un estudio exhaustivo de los distintos métodos de reparto estáticos. Poniendo de relieve las ganancias en tiempo aplicando paralelismo temporal frente a paralelismo espacial. Ejemplo de ello es el importante mejora obtenida para el último método con operaciones de E/S.

Se ha usado diferente número de procesadores sobre la red de estaciones de trabajo Sun Ultra-1, conectadas mediante un conmutador ATM, comprobando como la escalabilidad del problema con el aumento en el número de procesadores.

Nuestro trabajo actual va enfocado al desarrollo de un nuevo método basado en planificación dinámica, ya que hemos obtenido resultados muy diferentes con el mismo tipo de máquinas y en similares condiciones de carga, esto se debe principalmente a las diferencias entre frames, a los cambios de escena y a la presencia de diferentes objetos en ellas. Esta será una interesante mejora que llevará aun equilibrio a los frames por segundo obtenidos en cada procesador.

Tenemos una propuesta firme de completar y perfeccionar un método definitivo de reparto de la carga que mejore nuestro codificador paralelo y lo convierta en una herramienta competitiva que pueda utilizarse en múltiples aplicaciones que necesitan velocidad y rapidez de procesamiento.

6 Referencias

1. K.L. Gong, "Parallel MPEG-1 video encoding" MS thesis, Department of EECS, University of California at Berkeley, May 1994. Issued as Technical Report.
2. Bhaskaran, V. Konstantinides, K. "Image and Video Compression Standars. algorithms and Architectures" Second Edition. Kluwer Academic publishers.1997.
3. ISO/IEC JTC1 SC29 WG11 13818 Generic Coding of Moving Pictures and Associated Audio. Part 1: Systems, Part 2: Video, Part 3: Audio. ISO, 1994
4. K.R.Rao, J.J.Hwang. "Techniques & Standars for Image, Video & Audio Coding". Prentice Hall.1996.
5. http://drogo.cselt.stet.it/mpeg/standards/mpeg-4/mpeg-4.htm
6. F. Sijstermans, J.van der Meer, "CD-I full motion video encoding on a parallel computer". Com. ACM 34, 4 (Abril.1991), 81-91
7. ftp://mm-ftp.cs.berkeley.edu/pub/multimedia/mpeg/encode

8. K. Shen, L.A. Rowe, and E.J. Delp "A Parallel implementation of an MPEG1 encoder : Faster than Real-Time" Proceeding of the SPIE conference of Digital Video Compression: Algorithm and Technology 5-10 Feb. 1995. San José-California, pp. 407-418.
9. N. Kralevich, "The NOW Split-C MPEG Encoder" Internal Report Final Project CS267, University of California at Berkeley. Issued as Technical Report.
10. Y. Yu and D. Anastassiou "Software implementation of MPEG2 video encoding using socket programming in LAN" Proceeding of the SPIE conference of Digital Video Compression on Personal Computer: Algorithm and Technology 7-8 Feb. 1994. San José-California, pp. 229-240.
11. G.W. Cook and E.J. Delp. "The use of high performance computing in JPEG image compression" Proceedings of the Twenty-Seventh Asilomar Conference on Signals, Systems, and Computers. November 1993, Pacific Grove, California.
12. S.M. Akramullah, I. Ahmad, M.L. Liou. "A data-parallel approach for real-time MPEG-2 video encoding". *The Journal of parallel and Distributed Computing.* November 1995
13. S.M. Akramullah, I. Ahmad, M.L. Liou. "Performance of Software-based MPEG-2 video encoder on parallel and distributed systems". *IEEE Transactions on Circuits and Systems for Video Technology,* 1997
14. P.S.Pacheco, University of San Francisco. *Parallel Programming with MPI.* Morgan Kaufman Publishers, Inc.
15. R.J. Gove "The MVP : a highly-integrated video compression chip" Proceeding of IEEE Data Compression Conference, 28-31 March. 1994. Utah, pp. 215-224.
16. Luis, F. Martínez. "An efficient ISO/MPEG-1 layer II encoder using parallel processing techniques". Proyecto de investigación de la Universidad de Miami, Florida 1995
17. S.M.Akramullah, I.Ahmad, M.L.Liou *"Parallelization of MPEG-2 Video Encoder for Parallel and Distributed Computing Systems"*. Proc.of 38th Midwest Symposium on Circuits and Systems. Rio de Janeiro (Brazil). August 13-16,1995, Vol.2, pp.834-837.
18. T. Olivares, P. Cuenca, F. J. Quiles, A. Garrido, J. L. Sanchez and J. Duato. "Interconnection network behavior on a multicomputer in the parallelization of the MPEG coding algorithm. Worm-hole vs Packet-Switching Routing". Proceedings on the IEEE High Performance Computing, HiPC'97.December 18-21, 1997Bangalore, India, pp.48-53.
19. S.M.Akramullah, I.Ahmad, M.L.Liou *"A Portable and Scalable MPEG-2 Video Encoder on Parallel and Distributed Computing Systems"*. Proc.of SPIE, Vol.2727, Pt.2, Symposium on Visual Communications and Image Procesing'96, Orlando, Florida, March 17-20, pp.973-984.
20. A.Karp, *"Programming for Parallelism"*. IEEE Computer, May 1987, pp.43-57.
21. S.M.Akramullah, I.Ahmad, M.L.Liou, *"A Software based H.263 Video Encoder using a Cluster of workstations"*, Proc. Of SPIE, Vol.3166, SPIE's 1997 Optical Science, Engineering and Instrumentation Symposium, San Diego, CA, USA, 27 July-1 August, 1997.
22. S.M.Akramullah, I.Ahmad, M.L.Liou, *"Parallel MPEG-2 Encoder on ATM and Ethernet-conected Workstations"*, Lecture Notes in Computer Science, 4th International Conference of the ACPC (ACPC'99), Salzburg, Austria (to appear).