Recibido**:** 15 de Mayo de 2010 Aceptado: 15 de Agosto de 2010
Publicado en línea: 30 de Diciembre de 2010

Algoritmos paralelos de codificación de video basado en H.264/AVC con balanceo predictivo de carga

Carlos Genis-Triana[1], Abelardo Rodríguez-León[2] y Rafael Rivera-López[3]
Departamento de Sistemas y Computación. Instituto Tecnológico de Veracruz
Calzada Miguel Ángel de Quevedo 2779, Veracruz, México
[1]carlosgenis@yahoo.com.mx, [2]arleon@itver.edu.mx, [3]rrivera@itver.edu.mx

Abstract. Este artículo muestra una descripción de dos algoritmos paralelos para la codificación de video con balanceo predictivo de carga: Uno basado en la *estimación de movimiento exhaustivo* y el otro *adaptivo*. Ambos se basan en el estándar H264 que incluye una paralelización a nivel de GOPs, haciendo una distribución de los mismos en un cluster por medio del protocolo estándar de paso de mensajes MPI. La manera en que se distribuye la carga es mediante la combinación de dos esquemas: inicialmente por preasignación y posteriormente predicción. La idea es que una vez hecha la repartición inicial (preasignación), se procede a determinar la complejidad de los GOPs en tiempo de codificación (predicción), para posteriormente hacer una planificación que permita balancear la codificación, enviando los GOPs más pesados a los procesadores que codificaron previamente GOPs ligeros y viceversa. Como el adaptivo es una versión mejorada del exhaustivo, la evaluación esta en base dicha mejora fin de determinar el promedio del aprovechamiento a partir de resultados obtenidos con diferentes números de nodos de procesamiento, diversos tipos y resoluciones de videos de tal forma que nos permita establecer una ecuación con la cual se puede calcular el número de nodos codificadores necesarios para obtener tiempo real.

1 Introducción

En los últimos años se han propuesto y se han desarrollado una variedad de estándares de compresión de video e imagen (H.26X, MPEG-X, JPEG200), sin embargo, no todos proporcionan las mismas características. Unos obtienen mayor calidad de imagen a costa de un mayor tiempo de procesamiento (tiempo de compresión), por lo que se ha tenido que buscar un equilibrio entre el tiempo de procesamiento y la calidad que se obtiene de las imágenes descomprimidas. Por tanto, si se aumentará la velocidad de compresión se podrían utilizar compresores que ofrecen una calidad mayor. Hasta hace poco los más destacados eran H263 y MPEG4, debido a los grupos que los respaldan. Sin embargo, recientemente surgió uno nuevo, el H264 propuesto por el Joint Video Team (JVT), el cual promete aún más que todos lo antes citados.

Ahora bien, la tarea de comprimir video, demanda sistemas de alto rendimiento para lo cual se tienen las siguientes alternativas: Sistemas fuertemente acoplados, tarjeta codificadora y sistemas débilmente acoplados. Para explotar un sistema multiprocesador (débilmente o fuertemente acoplado) en la compresión del video, se requiere utilizar alguna implementación de algún estándar de compresión, paralelizándolo con técnicas de programación paralela tales como: MPI (Interfaz para el paso de mensajes) desarrollada por el grupo Forum de MPI.

Cabe recordar, que un video esta formado por una secuencia de imágenes a las cuáles denominamos frames y agrupándolos para formar un GOP (Group Of Pictures)y así poder realizar una paralelización (segmentación y distribución) a nivel de GOPs entre los nodos codificadores.

Es importante señalar que la codificación de cada GOP se basa en un patrón bajo el cual, el primer frame de cada GOP se denomina I, el cual se comprime utilizando la compresión intracuadros, eliminando sólo la redundancia contenida en un solo frame. El resto de los frames, se comprimen utilizando la compresión intercuadros, denominados de tipo B ó P, dependiendo de cómo se haya aplicado la técnica: Frame P, si se comprime por medio de otro, un I ó P anterior y Frame B, siempre que se comprime a partir de dos frames, un I ó P anterior y un P posterior.

Este artículo muestra una descripción de dos versiones de algoritmos paralelos para la codificación de video con balanceo predictivo de la carga. Ambos.

Se basan en el estándar H264 e incluyen una paralelización a nivel de GOPs (15 frames), haciendo una distribución de los mismos en un cluster por medio del protocolo estándar de paso de mensajes MPI. La manera en que distribuyen la carga es mediante la combinación de dos esquemas: Inicialmente por *preasignación* y posteriormente por *predicción*, de ahí el nombre. La idea es que una vez hecha la repartición inicial (preasignación), se proceda a determinar la complejidad de los GOPs en tiempo de codificación (predicción), para posteriormente hacer una planificación que permita balancear la codificación, enviando los GOPs más pesados a los procesadores que codificaron previamente GOPs ligeros y viceversa.

Los dos algoritmos coinciden en ser "*Predictivos con estimación de movimiento*", sin embargo, de manera general podemos decir que la diferencia estriba en que el primero lleva acabo la estimación de manera *exhaustiva* y el segundo forma *adaptativa*, siendo éste último una mejora del anterior. Por lo anterior la evaluación aquí presentada es acerca de la última versión citada analizando su aprovechamiento con diferentes números de nodos de procesamiento y con diversas resoluciones de videos a fin de determinar una ecuación que me permita calcular el número nodos requeridos para obtener *tiempo real*. Esto significa que el codificador paralelo ha de proporcionar una productividad en GOPs codificados por segundo igual o mayor a los GOPs por segundo generados por la fuente de video.

En el punto 1 se hace una introducción a la compresión de video y al procesamiento paralelo. 2. Se hace una descripción de los esquemas en los que se basa el predictivo. En el 3 se hace una descripción del algoritmo paralelo predictivo con estimación de movimiento exhaustivo. En el 4 se hace una descripción del algoritmo paralelo predictivo con estimación de movimiento adaptivo. En el 5 se muestran algunos resultados obtenidos de la evaluación a la que fue sometida la última versión del predictivo. Y por último en el 6 se hace mención sobre los trabajos que se podrían realizar más adelante.

2 Esquemas base para el balanceo predictivo

El esquema por ***preasignación*** es el primero el que se basa el predictivo para realizar la primera tanda de repartición de GOPs. La idea de esta estrategia es que cada proceso sepa, en base a su identificador, qué GOPs ha de codificar, repartiéndolos de la forma más equitativa posible. En principio el costo computacional de la codificación de los frames que componen un GOP puede ser

variable e impredecible. Esto plantea la conveniencia de hacer una distinción entre un GOP y otro, para hacer un reparto equitativo del trabajo. Con el fin de simplificar la idea de distribución asumiremos un costo computacional similar para cada GOP. Con esta hipótesis de partida, hacemos el siguiente esquema de distribución:

1. La secuencia de video se divide en GOPs, cada uno de los cuales contendrá una secuencia de cuadros. El primer cuadro a codificar siempre debe ser de tipo I, para hacer independiente la codificación de GOPs. Los procesadores son identificados con un índice entre 0 y el número de procesadores disponibles menos 1.
2. Se divide el número de GOPs entre el número de procesadores, y se asigna a cada procesador el número de GOPs consecutivos resultante, de manera cíclica. En caso de que el número de GOPs no sea múltiplo del número de procesadores, entonces el número de GOPs restante *n,* se asignarán a los *n* primeros procesadores, a razón de un GOP por procesador.

De esta forma cada procesador sabe cuales son los GOPs consecutivos que le tocan codificar. Una vez terminada la codificación de todos los GOPs de la secuencia, el proceso 0 espera que los demás nodos le envíen un mensaje de terminación para proceder con la unión de todos los GOPs codificados en un sólo archivo.

Este esquema presenta problemas cuando el número de procesadores no es múltiplo exacto del número de GOPs, ya que presenta un desbalanceo de carga que se produce al final de la codificación, en donde ciertos procesadores terminan antes que otros. Por ejemplo en un caso extremo, si se tienen 5 GOPs y tenemos 4 procesadores; al procesador 1 le corresponden 2 GOPs y a los otros 3 les toca solo un GOP. Esto, junto con el hecho de que los tiempos de codificación de GOPs no son iguales, lleva a que 3 procesadores deben esperar ociosos a que el primer procesador termine. El resultado de esta espera es que la eficiencia baja sustancialmente en estos casos, sobre todo cuando trabajamos con secuencias de video de corta duración. La **Figura 1** muestra esquemáticamente la funcionalidad de la versión de preasignación del codec.

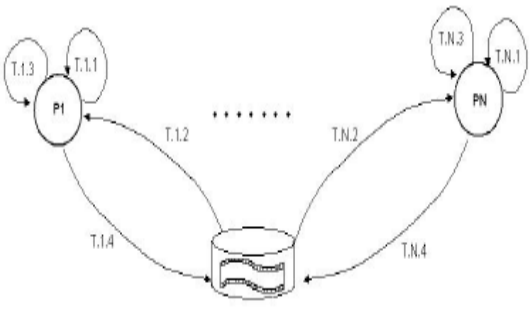

Figura 1. Esquema de Distribución de GOPs por Preasignación

Los estados del grafo de la **Figura 1** son descritos en la **Tabla 1**. Cada línea indica las

tareas que se procesan en paralelo. Cada columna muestra las tareas de cada procesador.

Tabla 1. Descripción de estados de GOPs por preasignación.

T.1.1. P1 calcula GOP a codificar	.....	T.N.1. PN calcula GOP a codificar
T.1.2. P1 lee GOP del disco.	.....	T.N.2. PN lee GOP del disco
T.1.3. P1 codifica GOP	.....	T.N.3. PN codifica GOP
T.1.4. P1 escribe GOP codificado	.....	T.N.4. PN escribe GOP codificado

Para hacer menos significativas las diferencias de tiempo final, existe un esquema de distribución en el cual se tiene un nodo distribuidor y coordinador más dinámico denominado por ***demanda***. La importancia de este esquema radica en que una vez hecha la primera tanda de repartición de los GOPs los algoritmos predictivos tienen como punto de partida éste esquema que permita mejorar el balanceo de la carga entre los procesadores codificadores.

El nodo distribuidor se encarga de indicarle a los demás procesadores que GOPs deben codificar. En la primera ronda de reparto la asignación es secuencial. Pero después se va asignando los GOPs conforme son solicitados por los nodos codificadores (bajo demanda). Los nodos codificadores reciben la indicación del número de GOP a codificar, lo codifican y cuando terminan piden un nuevo GOP. El nodo coordinador les enviará el identificador del GOP siguiente o -1 si no hay más GOPs a codificar. Una vez terminado el proceso de repartición y codificación el nodo distribuidor se encarga de integrar los GOPs codificados en un solo flujo. Este proceso de asignación se puede observar en el grafo de estados de la **figura 2.**

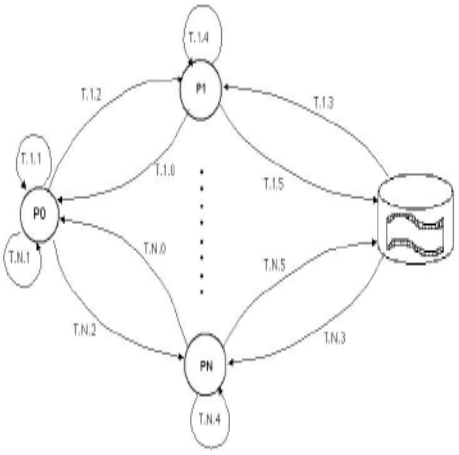

Figura 2. Grafo de Distribución de GOPs por Demanda

Los estados del grafo de la **Figura 2** son descritos en la **Tabla 2** y los detalles de implementación en el **Algoritmo 1**

Tabla 2. Descripción de estados de GOPs por demanda.

T.1.0. P1 avisa a P0 que está libre.	.....	T.N.0. PN avisa a P0 que está libre.
T.1.1. P0 decide GOP a codificar por P1	.....	T.N.1. P0 decide GOP a codificar por PN
T.1.2. P0 avisa a P1 GOP a codificar	.....	T.N.2. P0 avisa a PN GOP a codificar
T.1.3. P1 lee GOP de disco	.....	T.N.3. PN lee GOP de disco
T.1.4. P1 codifica GOP	.....	T.N.4. PN codifica GOP
T.1.5. P1 escribe GOP codificado	.....	T.N.4. PN escribe GOP codificado

Algoritmo 1. Distribución de GOPs por Demanda

```
for ( ng = 0 to n_GOPs )
{  if(ng<=nprocs)
   {     Proc=ng        }
   else
   {     Proc=Espera_Recp_Proc()    }
   Codifica_GOP(Proc,ng)

}
Detiene todos los Procesadores
Ensamblar GOPs codificados en fichero único
```

3 Algoritmo predictivo con estimación de movimiento exhaustiva

A este algoritmo y al siguiente se les llama Predictivos, ya que el algoritmo distribuye los GOPs basándose en una predicción efectuada por los nodos codificadores. Al mecanismo descrito aquí se le llama “con estimación de movimiento exhaustiva” porque hace una búsqueda de macrobloques con todos los posibles tamaños de bloque que permite el codec, a diferencia de como se hace en el algoritmo descrito en el punto siguiente.

La similitud con la asignación por demanda reside en el hecho de que los procesos codificadores informan al proceso 0 acerca de su disponibilidad para codificar un nuevo GOP (específicamente predicen el tiempo que falta para terminar de codificar el GOP actual). El proceso 0 (nodo coordinador/distribuidor) asigna los GOP a los procesos codificadores. Los procesos codificadores acceden a los GOPs directamente del disco, antes de terminar la codificación del GOP en curso notifican al nodo distribuidor una estimación del tiempo restante en la codificación del GOP actual. Justo antes de terminar, el nodo distribuidor les notifica cual es el siguiente GOP, cuando terminan de codificar el GOP actual, pueden comenzar a codificar el siguiente GOP asignado sin ningún tipo de espera.

La diferencia radica en dos partes. En primer lugar la asignación de los GOPs ya no es secuencial y en segundo lugar la respuesta del proceso codificador ya no se produce cuando termina la codificación de GOP, sino un poco antes. A continuación describiremos cada una de estas dos diferencias.

3.1 Ventanas de GOPs

Las ventanas de GOPs es un concepto lógico para dividir el flujo de video en Grupos de GOPs. Se tienen tantas ventanas en la secuencia, como procesos codificadores. El número de GOPs por ventana se puede calcular con la siguiente expresión:

$$n\_gop\_v = \begin{cases} \frac{n\_Gops}{n\_proc} + 1, & k < \mathrm{mod}(n\_Gops, nproc) \\ \frac{n\_Gops}{n\_proc}, & k \geq \mathrm{mod}(n\_Gops, nproc) \end{cases}$$

El acceso a la secuencia de video no se realiza de forma secuencial, sino que cada procesador codifica una ventana de GOPs. El equilibrio de la carga se pretende conseguir cambiando los procesadores de una ventana de GOPs a otra, en función del costo computacional que tenga la codificación de un GOP en una ventana determinada.

En la **Figura 3** se ilustra un ejemplo en el que inicialmente el GOP de la ventana 2 es el más pesado y el de la ventana 1 el más ligero. Tras la primera ronda de asignaciones, los procesadores de las ventanas 1 y 2 son intercambiados (procesadores 1 y 2 respectivamente). En esta segunda ronda la ventana 3 sigue siendo procesada por el procesador 3. Supongamos que en la segunda ronda la ventana más pesada es la 3 y la más ligera la 2. Se intercambian los procesadores de las ventanas 2 y 3 (procesadores 1 y 3 respectivamente).

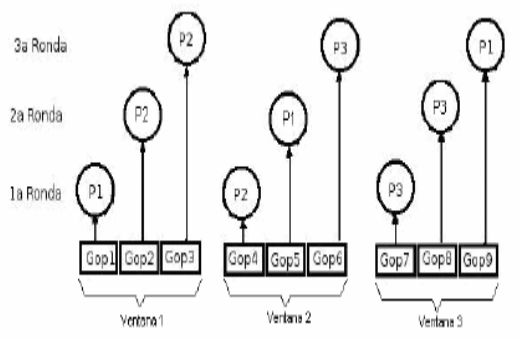

Figura 3. Rondas de intercambio de procesadores en la versión GOPs Predictiva

3.2 Predicción del tiempo de codificación de un GOP

Antes de explicar el esquema, cabe mencionar que en cada GOP se codifican 16 frames, aunque solo se almacenan los primeros 15. Lo anterior se hace por dos razones: Para conservar el patrón de codificación y para poder concatenar con el siguiente GOP.

De acuerdo al patrón de codificación empleado, los frames 13 y 14 deben ser B's y por tanto necesitan tener a ambos lados de ellos frames I's o P's. Como nuestro GOP es de 15 frames, nos vemos obligados a codificar un frame más para conservar el patrón de 2 B's entre I's o P's.

La segunda razón nos lleva a codificar el frame adicional como I para poder concatenar con el siguiente GOP. El último frame de un GOP es también el primero del siguiente (que

se codifica en otro procesador), por ello es necesario que sea I este
frame adicional.

Para saber el costo computacional de cada GOP, los procesos codificadores informan al proceso 0, cuando terminan de codificar el último frame I del GOP (el frame 15 en la **Tabla 3**).

Tabla 3. Patrón de codificación de frames

Tipo	I	P	B	B	P	B	B	P	B	B	P	B	B	I	B	B
Frame	0	3	1	2	6	4	5	9	7	8	12	10	11	15	13	14

Se determinó que este fuera el punto adecuado, ya que en base a un estudio estadístico, se observó que la precisión de la estimación del tiempo restante de codificación del GOP era muy elevado cuando se hacia tras codificar al menos el 80% de los frames de un GOP. Para obtener esta estimación, se calculan los tiempos promedios de cada tipo de frame y se obtiene el tiempo que falta para terminar. Al llegar al último frame P, se han codificado ya 2 frame I, 4 frames P y 8 frames B. Por lo tanto lo que falta es codificar solo dos Frames B. El primer procesador que llega al frame 12 (P) del GOP que está codificando, establece el tiempo restante como el límite de espera para lanzar la planificación. Mientras transcurre este tiempo los demás procesadores siguen informando de su avance. Si alguna de las predicciones recibidas dice que termina el GOP en un tiempo menor que el establecido previamente, el nodo distribuidor (nodo 0) reasigna este tiempo como nuevo límite de espera. Justo antes de que finalice el tiempo de espera se dispara un temporizador para realizar el proceso de planificación.

Cuando el temporizador vence, se lanza la planificación con el objeto de asignar los procesadores entre las ventanas de GOPs de acuerdo con las predicciones que se hayan recibido hasta ese momento. Así, los procesadores que terminan primero son asignados a las ventanas más pesadas. Las predicciones que no hayan llegado, se consideran que son las de las ventanas pesadas y por lo tanto se les asigna valores de predicción muy altos. Después de lanzar la planificación, el temporizador inicializa las tablas internas y está listo para recibir nuevas predicciones. Tras la planificación, el nodo distribuidor envía los mensajes de asignación a todos los nodos codificadores de tal forma que los nodos codificadores reciban el mensaje antes de terminar la codificación del GOP actual.

El **Algoritmo 2** muestra como funciona el proceso coordinador, como se describió en los párrafos anteriores.

```
Calcula el número de ciclos de asignación de GOPs (Nciclos)
Determina ventanas de GOPs
Asigna a cada procesador su primer GOP
for ( ciclo = 1 to Nciclos-1 )
{   for ( procesador = 0 to Nproc-1)
    {    Lee Predicción
         if( ¿primera predicción?)
         {    Inicializa Timer
              Lanza Timer (timeout = Prediccion)
         }
         else
         {    if( Predicción < timeout && ¿Planificación no hecha?)
              {    Detener Timer
                   Lanzar Timer (timeout=Prediccion)
              }
         }
    }
    if ( ¿no ha vencido el Timer?)
    {    Detener Timer
         Lanzar Planificación
    }
}
Detiene todos los Procesadores
Espera últimos mensajes de Predicción
Ensamblar GOPs codificados en fichero único
```

Algoritmo 2. Esquema predictivo por estimación de costo

En la **Figura 4** se puede observar el esquema funcional del codec de GOPs predictivo. Los estados se muestran en la **Tabla 4**:

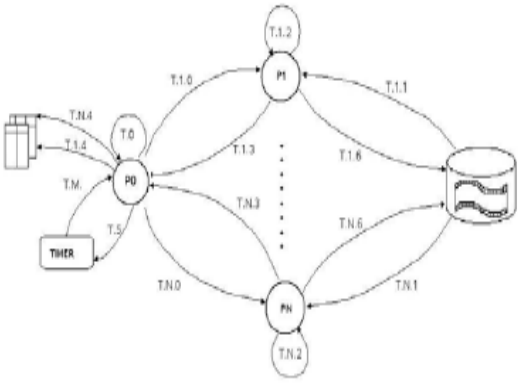

Figura 4. Esquema de funcionalidad del codec de GOPs Predictivo

Tabla 4. Descripción de estados de distribución de GOPs predictivo

T.0. – P0 determina los GOP de la primera ronda a enviar		
T.1.0 - P0 avisa a P1 GOP a codificar	.....	T.N.0.- P0 avisa a PN GOP a codificar
T.1.1.- P1 lee GOP de disco.	.....	T.N.1.- PN lee GOP de disco.
T.1.2.- P1 codifica GOP.	.....	T.N.2.- PN codifica GOP.
T.1.3.- P1 envía predicción	.....	T.N.3.- PN envía predicción.
T.1.4.- P0 almacena predicción	.....	T.N.4.- P0 almacena predicción
T.1.5.- P0 Inicializa o redefine Timer		
T.M. - Se dispara el Timer hace Planificación y la envía a P0		
T.0. – P0 asigna los GOP a enviar en 2a ronda		
T.1.6.- P1 Escribe GOP codificado	.....	T.N.6.- PN Escribe GOP codificado.

Este algoritmo resulta interesante, cuando la latencia no es un factor importante (por ejemplo codificación off-line de video almacenado), ya que a diferencia de los esquemas de preasignación y demanda, la codificación de los GOPs no es secuencial.

4. Predictivo con estimación de movimiento Adaptativo

Partiendo del esquema Predictivo, pensamos que podría mejorarse la efectividad de la codificación, si hacíamos que el codificador hiciera una búsqueda más intensiva cuando se codifican GOPs con

mucho movimiento y una búsqueda más ligera en el caso contrario.

Con esta idea, analizamos como poder lograr esto, explorando los parámetros de configuración del codec. Encontramos que cuenta con un parámetro llamado **Interseach** que sirve para indicar los diferentes tamaños de bloque que se usarán en la exploración de la estimación de movimiento.

Dicho parámetro define los siguientes tamaños de macrobloques: 16x16, 16x8, 8x16, 8x8, 8x4, 4x8 y 4x4.

Así, si la imagen cuenta con poco movimiento o este es constante, con un tamaño de bloque único de 16x16 se puede codificar de forma certera el movimiento, haciendo innecesaria la exploración con los demás tamaños de bloque. En cambio, si tenemos una imagen con mucho movimiento (ampliación de imagen, dispersión de objetos, objetos pequeños en movimiento o giros bruscos) entonces es necesario hacer una exploración más completa con los diferentes tamaños para encontrar la codificación del movimiento más adecuada.

Cabe señalar que el codificar algunos GOPs con menos bloques en la estimación de movimiento, trae consigo dos variaciones importantes con respecto a los resultados de la codificación exhaustiva. La primera es un ligero incremento en el tamaño del stream codificado. La segunda es una reducción del tiempo de codificación, ya que al estimar el movimiento con menos bloques, la codificación es más rápida.

Por defecto, la configuración del codificador tiene activada la búsqueda de movimiento con todos los tamaños de bloque, pudiéndose desactivar algunos tamaños de acuerdo a las necesidades de codificación.

El siguiente problema con que nos encontramos fue como identificar, antes de proceder a la codificación, las características de movimiento en el video, no encontrando una solución sencilla y automática a este problema.

Sin embargo, el mecanismo de estimación de movimiento adaptativa (es decir activando solo algunos tamaños de bloques) es una idea que resulta interesante y por tanto debíamos primero identificar si valdría la pena intentar encontrar una solución al
problema anterior.

Para identificar si vale la pena seguir esta línea de trabajo, decidimos fijar manualmente los tamaños de los macrobloques a usar, dependiendo de la cantidad de movimiento presente en cada GOP.

Para identificar la cantidad de movimiento, se usó el PSNR de los GOPs codificados usando tres configuraciones:

- *E0: Búsqueda exhaustiva.*
- *E1: Búsqueda con bloques de tamaño 16x16*
  16x8, 8x16 y 8x8
- *E2: Búsqueda solo con bloques de 16x16.*

Se aplicaron estas configuraciones a 2 secuencias de video distintas: Foreman_CIF y Stockholm_HDTV720.

Con las diferencias del PSNR entre las configuraciones de búsquedas descritas (E1-E2 y E0-E2), se estableció un rango de variación de E0-E2 con respecto a E1-E2, con el objetivo de poder determinar la configuración a usar en cada GOP.

Si E0-E2 es menor rango establecido, entonces se utiliza la configuración E0 para codificar ese GOP. Si cae dentro del rango establecido se escoge la configuración E1 y si resulta mayor se usa la configuración E2.

De esta forma, las configuraciones usadas para cada GOP en las dos secuencias se muestran en la **Figura 5**. En las gráficas se muestran como se pueden codificar diferentes GOPs usando algunos de los esquemas planteados. El eje de las ordenadas muestras los tres posibles esquemas E0, E1 y E2. El eje de las abscisas indica el número de GOP de la secuencia de video.

Estos valores se han establecido de forma manual simulando una función de costo de tres niveles para cada GOP. A este esquema lo llamamos E-A (Esquema Adaptativo). A consecuencia de aplicar lo anterior se obtienen importantes reducciones del costo temporal del algoritmo, si lo comparamos con la codificación estándar con estimación de movimiento exhaustiva (E0).

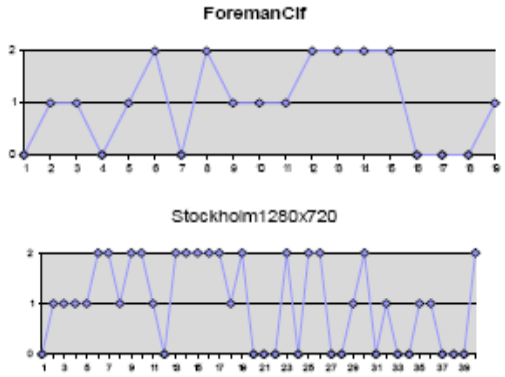

Figura 5. Selección adaptativa de los esquemas de estimación de movimiento.

En la **Figura 6**, se muestra la variación del costo temporal de la codificación de GOPs en los niveles de estimación definidos: E0, E1, E2 y EA. La función de costo de EA busca reducir al mínimo el tiempo de codificación, manteniendo al máximo la calidad, sin incrementar demasiado el tamaño del bitstream, como se puede observar en las **Figuras 7 y 8**.

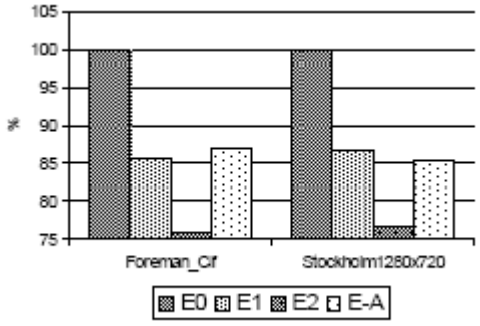

Figura 6. Variación porcentual del tiempo de codificación adaptativa.

En la **Figura 7** se puede observar como la pérdida de calidad en EA corresponde a centésimas de decibelio. Esta pérdida de calidad la podemos considerar despreciable, siendo este un factor prioritario a la hora de codificar video de alta calidad.

En la **Figura 8** se puede ver el efecto adverso del incremento del bitstream en donde el esquema E2 obtiene los incrementos más significativos en el tamaño del bitstream, siendo de nuevo el esquema EA el que obtiene un valor intermedio (al rededor del 3%). Por tanto, será el incremento en la tasa de bits el precio que asumimos como contrapartida cuando empleamos estimación de movimiento adaptativa.Los valores de E0 mostrados son solo de referencia.

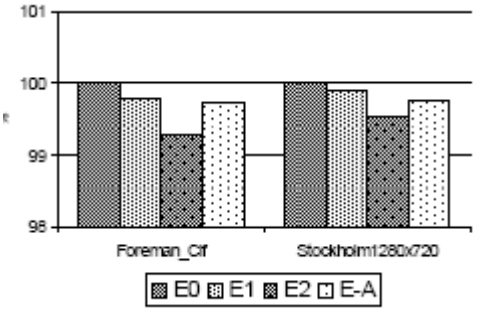

Figura 7. Variación porcentual del PSNR en codificación adaptativa

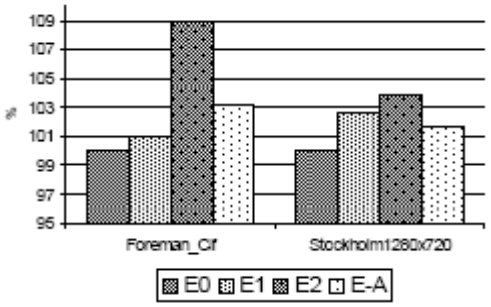

Figura 8. Variación porcentual del Bitrate en codificación adaptativa

Por tanto, podemos decir que utilizar un esquema de estimación de movimiento adaptativo es interesante ya que reduce significativamente el tiempo de codificación (~15%). El PSNR no se reduce significativamente (~0.12%) y solo se incrementa ligeramente el tamaño del bistream (~2%). Este incremento del bitrate no será determinante para un gran abanico de aplicaciones (difusión de TV digital en formatos HDTV, videoconferencia de alta calidad, etc.), en las que el retardo del codificador es un parámetro crítico y se debe mantener la calidad de la señal al máximo.

El desarrollo de un algoritmo para la estimación de movimiento adaptativa automática [KC98], no es una tarea trivial y considerando que se aparta de los objetivos centrales de este trabajo, se decidió posponerlo para trabajos futuros. Sin embargo, si el costo computacional de esta tarea es significativo, entonces la reducción que se obtiene al aplicar la estimación de movimiento adaptativa se diluye, no siendo útil la aplicación de esta propuesta. Por tanto,

habría que buscar soluciones que no supongan un incremento substancial en el costo computacional del codificador

5. Resultado obtenidos

Se realizaron pruebas exhaustivas del último algoritmo predictivo descrito. Para ello se utilizó el Cluster Mozart, que cuenta con 4 nodos biprocesador AMD Opteron 246 de 2Ghz, disponiendo de un total de 8 procesadores útiles para MPI. La red de interconexión de los nodos utiliza tecnología Gigabit Ethernet.

De entre varias secuencias disponibles en formato HDTV1080, elegimos Riverbed por contar con características que ponen a trabajar de forma intensiva la estimación de movimiento del codec. El contenido de la secuencia es un encuadre del agua pasando en el lecho de un rió, donde se ven en el fondo pequeños guijarros. Al tener el agua un movimiento no uniforme, presenta un reto para la búsqueda y precisión de los vectores de movimiento.

Con esta secuencia en formato YUV 4:2:0, se procedió a crear otras 3 secuencias en tamaños HDTV1088, HDTV720 y DV NTSC. Por lo tanto, las secuencias de tests seleccionadas tendrán las siguientes resoluciones: 1920x1088, 1280x720 y 720x480. Cabe mencionar que el Codec H264 v7.6 solo maneja resoluciones que sean múltiplos de 16, ya que éste es el tamaño del macrobloque.

Cada algoritmo se evaluó con distintas configuraciones de 2, 4 y 8 procesadores. Cada una de las evaluaciones se repitió 5 veces para obtener resultados más fiables.

El objetivo de usar 3 tamaños diferentes de secuencias de video es para observar el efecto que puede tener el cambio de resolución en el rendimiento del último algoritmo predictivo que se desarrolló. Se han utilizado 3 configuraciones diferentes de procesadores para observar el efecto que surte de la agregación de procesadores a la codificación y determinar con ello la escalabilidad del algoritmo. La repetición de cada prueba permite descartar estadísticamente la variación debida al funcionamiento normal de una computadora (cache, procesos locales, etc).

En la **Tabla 5** podemos observar la eficiencia obtenida del último algoritmo predictivo descrito en cada resolución. En base a los resultados obtenidos realizaremos una estimación del número de procesadores requeridos para obtener tiempo real. La estimación se hace mediante la *ley de Little*. Esta ley operacional requiere como única condición el equilibrio de flujo y se fórmula como: $N = X \cdot R$, donde:

- N Número de GOPs procesados en Paralelo
- R Tiempo de codificación de 1 GOP en paralelo

- X GOPs codificados por segundo (throughput)

Tabla 5. Eficiencia de algoritmo predictivo

	Resolución	Ef-2P	Ef-4P	Ef-8P	Media
Predictivo	720x480	0,9545	0,9638	0,9478	0,9554
	1280x720	0,9527	0,9525	0,9309	0,9453
	1920x1088	0,9633	0,9527	0,9536	0,9565
	Media	0,9568	0,9563	0,9441	0,9524

En paralelo N es el número de GOPs que se procesan a la vez, por tanto en los esquemas presentados en este apartado será igual al número de procesadores. Si consideramos que el tiempo de codificación en el codificador paralelo de un GOP es R_{PAR}, podemos decir que la productividad en paralelo es:

$$X_{PAR} = \frac{N}{R_{PAR}}$$

En secuencial, $N = 1$, y si el tiempo de codificación secuencial de un GOP es R_{SEQ} , la productividad será

$$X_{SEQ} = \frac{1}{R_{SEQ}}$$

El *SpeedUp* es el cociente del tiempo de codificación secuencial de todos los GOPs, entre el tiempo de codificación en paralelo. Suponiendo que el número total de GOPs a codificar es L, de las dos fórmulas anteriores tendremos:

$$Sp = \frac{\frac{L}{X_{SEQ}}}{\frac{L}{X_{PAR}}} = \frac{X_{PAR}}{X_{SEQ}} = \frac{\frac{N}{R_{PAR}}}{\frac{1}{R_{SEQ}}} = N\frac{R_{SEQ}}{R_{PAR}}$$

Donde N es el número de GOPs a codificar, R_{SEQ} es el tiempo de codificación secuencial de un GOP y R_{PAR} es el tiempo de codificación de un GOP en el codificador paralelo.

La eficiencia es el cociente del *SpeedUp* dividido entre el número de procesadores, así pues tendremos:

$$E_G = \frac{Sp}{N} = \frac{N \bullet \frac{R_{SEQ}}{R_{PAR}}}{N} = \frac{R_{SEQ}}{R_{PAR}}$$

Entonces el tiempo de codificación de un GOP en el codificador paralelo es:

$$R_{PAR} = \frac{R_{seq}}{E_G}$$

Según la ley de Little tenemos que el número de procesadores requerido para obtener una productividad X_{PAR} es:

$$N = R_{PAR} \bullet X_{PAR} = \frac{R_{seq}}{E_G} \bullet X_{PAR}$$

En base a resultados mostrados en la **Tabla 5**, la eficiencia promedio de los algoritmos desarrollados es E_G=0.95. Si consideramos además que (por ejemplo) un GOP en resolución 720x480 se codifica secuencialmente en R_{SEQ}=252.06 segundos y que la productividad para alcanzar la codificación en tiempo real es X_{par}=30 frames por segundo o X_{par}=2 GOPs por segundo; podemos decir que la cantidad de nodos necesarios para alcanzar dicha productividad serian al menos:

$$N = \frac{252.06seg}{0.95} \bullet 2Gops/seg = 531Nodos$$

6. Trabajos Futuros

Una vez hecho la publicación de este artículo en el cual se obtuvo la eficiencia del algoritmo predictivo para diferentes resoluciones de videos, número de GOPs y nodos codificadores estableciendo de esta forma una ecuación que me permitirá calcular el número de nodos codificadores que se requieren para obtener tiempo real. Se propone realizar pruebas más exhaustivas, a fin de determinar si el algoritmo es escalable y así poder saber si su rendimiento mejora conforme se van incrementando el número de GOPs. También para determinar para que tipo de video es más adecuado, considerando otras características de los mismos, tales como: No. de fondos dinámicos y/o estáticos, No. de figuras centrales, No. de objetos en movimiento, etc. Otro de los posibles trabajos es realizar una paralelización en otros niveles, tales como: slices, instrucciones multimedia o implementando una combinación de niveles.

Referencias

1. Dongarra, J., Foster, I., Fox, G., Gropp-William, K. K., Torczon, L. y White A. “SourceBook of Parallel Computing”, Ed. Morgan Kaufmann (2003).[ES99] Effelsberg, W. y Steinmetz, R. “Video Compression Techniques”. Ed. Morgan Kauffman (1999).
2. J. C. Fernández y M. P. Malumbres, "A Parallel Implementation of H.26L Video Encoder", Lectures Notes on Computer Sciences No. 2400, Springer-Verlag pp. 830-833, 2002.
3. Genis-Triana C., Rodríguez-León A. “Evaluación de una versión paralela para el Codec H.264/AVC”. Artículo publicado en las memorias del CORE del CIC-IPN (Congreso Nacional del Centro de Investigación en Computación del Instituto Politécnico Nacional) cuyo título es: Recientes avances en la ciencia de la computación en México con el ISBN: 970-36-0149-9. Mayo del 2004.
4. Genis-Triana C., Rodríguez-León A. “Characterization of the load balance of a parallel implementation of the H264 by the GAP of DISCA-UPV”. Artículo publicado en

las memorias del CIICC'04 del ITTLA (11vo. Congreso Internacional de Investigación en Ciencias Computacionales del Instituto Tecnológico de Tlalnepantla) con el ISBN: 968-5823-10-3. Septiembre de 2004.

5. Genis-Triana C. "Caracterización de un Algoritmo Paralelo con Balanceo Predictivo de Carga para la Compresión de Secuencias de Video con H.264". Tesis para obtener al Grado de Maestro en Ciencias de la Computación en el ITV. Mayo del 2006.
6. Rodríguez-León Abelardo. "Diseño e implementación de algoritmos paralelos para la compresión de secuencias de video MPEG4". Reporte Interno de Investigación DISCA - Universidad Politécnica de Valencia. Noviembre de 2002.
7. Rodríguez-León Abelardo. "Desarrollo y Evaluación de algoritmos paralelos para la compresión de secuencias de video". Tesis doctoral. Universidad Politécnica de Valencia. Noviembre de 2007.
8. Sánchez-Zavaleta M. "Monografía de MPI". Tecnológico de Monterrey campus Morelos. Enero de 2001
9. Schäfer R., Wiegand T. and Schwarz H. "H.264/AVC The emerging standard". Heinrich Hertz Institute, Berlin, Germany. January 2003.
10. Wiegand, T., Sullivan, G.J., Bjontegaard, G. y Luthra, A., "Overview of the H.264/AVC Video Coding Standard", en IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, No. 7, pp. 560-576, July 2003.

Carlos Genis Triana es Ingeniero en Sistemas Computacionales egresado del Instituto Tecnológico de Veracruz en 2001. Obtuvo su grado de Maestro en Ciencias de la Computación por el Instituto Tecnológico de Veracruz en 2006. Es profesor del área de Sistemas y Computación en el Instituto Tecnológico de Veracruz y en otras instituciones educativas privadas en Veracruz, Ver. Sus áreas de interés son el cómputo distribuido y la programación orientada a objetos.

Abelardo Rodríguez León recibió su título de Ingeniero en Sistemas Computacionales en el Instituto Tecnológico de Veracruz en 1990, el grado de maestro en Ciencias de la Computación con la especialidad en Ingeniería de Software, en

la Universidad Veracruzana en 1996 y el grado de Doctor en Ciencias de la Computación en la Universidad Politécnica de Valencia, España, en 2007. Actualmente es profesor investigador del área de Sistemas y Computación del Instituto Tecnológico de Veracruz. Sus áreas de interés incluyen el cómputo en grid, la programación en paralelo y la optimización.

Rafael Rivera López recibió su título de Ingeniero en Sistemas Computacionales en el Instituto Tecnológico de Veracruz en 1991, el grado de maestro en Ciencias de la Computación en el Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Cuernavaca, en 2000 y actualmente está trabajando para la obtención de su grado de Doctor en Ciencias de la Computación por el mismo instituto. Es profesor investigador del área de Sistemas y Computación del Instituto Tecnológico de Veracruz. Sus áreas de interés incluyen la optimización, la programación orientada a objetos y la inteligencia artificial.