Paralelización Jerárquica del Codificador HEVC en Plataformas Heterogéneas

H. Migallón[1], P. Piñol[1], O. López-Granado[1], V. Galiano[1], E. Alcocer[2], M. Martínez-Rach[1] y M.P. Malumbres[1]

Resumen— **Existen diversas propuestas para reducir el tiempo total requerido por el estándar HEVC para codificar una secuencia de vídeo. En este trabajo proponemos una estrategia de paralelización jerárquica en el codificador HEVC basada en GOPs y *slices* para explotar de forma eficiente los recursos disponibles en arquitecturas paralelas y reducir de forma significativa el tiempo total de codificación de una secuencia de vídeo. La idea principal en esta aproximación jerárquica es dividir la secuencia en GOPs o grupos de GOPs que se asignarán a cada nodo de computación. Dentro de cada nodo, cada cuadro que pertenece a un GOP se dividirá en tantos *slices* como el número de procesos en dicho nodo. De esta forma, seremos capaces de aprovechar el paralelismo basado tanto en memoria distribuida como en memoria compartida. Los resultados muestran que se pueden obtener *speed-ups* de hasta 68,1x (usando 10 nodos y 11 procesos por nodo) cuando se combinan las técnicas basadas en GOPs y en *slices* para secuencias de vídeo de alta resolución**

Palabras clave— **Algoritmos paralelos, codificación de vídeo, HEVC, paralelismo jerárquico, multicore, GOP, particionado en slices, memoria distribuida, memoria compartida.**

I. Introducción

EL nuevo estándar de codificación de vídeo HEVC (High Efficiency Video Coding), ha sido desarrollado para poder hacer frente a las actuales y futuras tendencias en el mercado multimedia, como los contenidos con definición 4K u 8K y con una profundidad de color de 10 bits. El estándar HEVC mejora la eficiencia de codificación con respecto al codificador H.264/AVC proporcionando la misma calidad de vídeo necesitando aproximadamente la mitad de la tasa de bits [1]. Por otra parte, la complejidad del codificador HEVC es varios órdenes superior que su predecesor H.264/AVC [2], por lo que reducir su complejidad se ha convertido en una línea de investigación muy activa. Se puede encontrar una descripción amplia de HEVC en [3].

En la literatura se pueden encontrar diversos trabajos analizando la complejidad y presentando estrategias de paralelización del estándar HEVC como se puede consultar en [4] [5] [6]. La propia definición del estándar HEVC incluye nuevas características que permiten paralelización a alto nivel (a nivel de cuadro o sub-cuadro), como WPP (Wavefront Parallel Processing), *slices* y *tiles*, y otras características también novedosas que permiten paralelización a bajo nivel (dentro del proceso de codificación), como LPM (Local Parallel Method) [7] que permite realizar la estimación de movimiento de forma paralela.

La mayoría de las publicaciones que tratan sobre la paralelización de HEVC se centran en la parte del decodificador. Estas estrategias de paralelización se centran en diferentes niveles de paralelismo (nivel de sub-cuadro, nivel de cuadro, y nivel de secuencia). A nivel de sub-cuadro, en [8], los autores propone un optimización de grano fino en el módulo de estimación de movimiento del codificador HEVC, permitiendo realizar a la vez la predicción de todos los vectores de movimiento en todas las unidades de predicción (PU - Prediction Units) disponibles para una unidad de codificación (CU - Coding Unit). En [9], los autores proponen una paralelización dentro del módulo de predicción *intra* que consiste en eliminar las dependencias de los datos entre subbloques de una CU, obteniendo valores interesantes de *speed-up*. Otros trabajos recientes se centran en variaciones en el orden de escaneo. Por ejemplo, en [10], los autores proponen un orden de escaneo de CUs basado en una búsqueda en forma de diamante, obteniendo un buen esquema para procesado paralelo masivo. También, en [11], los autores proponen cambiar el orden del procesamiento del filtro de suavizado de bordes de bloques obteniendo reducciones de tiempo del 37,93% en plataformas *many-core*. Respecto a la paralelización a nivel de cuadro, en [12], los autores presentan una comparación entre aproximaciones usando *slices* y *tiles*, mostrando ambas estrategias un buen rendimiento paralelo. La estragegia de paralelización basada en *slices* obtiene *speed-ups* de hasta 7,3x para 10 procesos, mientras que la estrategia basada en *tiles* obtiene *speed-ups* de hasta 7.8x mostrando una menor degradación R/D (Rate/Distortion). En la paralelización a nivel de secuencia, en [13], los autores presentan una estrategia de alto nivel para plataformas de memoria compartida. En esta aproximación, basada en GOPs (Groups Of Pictures), cada proceso codifica simultáneamente varios grupos de cuadros consecutivos. Dependiendo de cómo estén formados y distribuidos estos GOPs, se puede obtener un buen rendimiento paralelo.

La mayoría de las propuestas enumeradas anteriormente han sido diseñadas con una estrategia particular de paralelismo. Algunas de ellas se aplican a alto nivel (paralelismo de grano grueso), dividiendo la secuencia de vídeo en porciones o GOPs para ser procesados usando los nodos disponibles. En estos casos,

[1]Departamento de Física y Arquitectura de Computadores, Universidad Miguel Hernández de Elche, e-mail: {hmigallon, pablop, otoniel, vgaliano, mmrach, mels}@umh.es

[2]Departamento de Ciencia de Materiales, Óptica y Tecnología Electrónica, Universidad Miguel Hernández de Elche, e-mail: ealcocer@umh.com

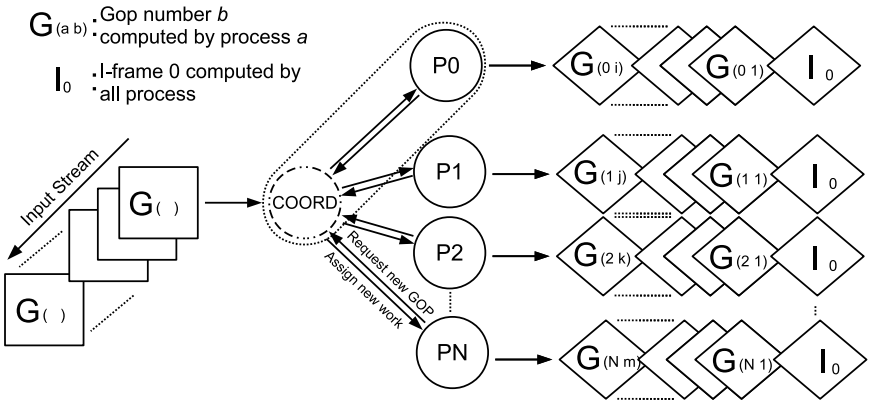

Fig. 1: Esquema del algoritmo DMG-1G.

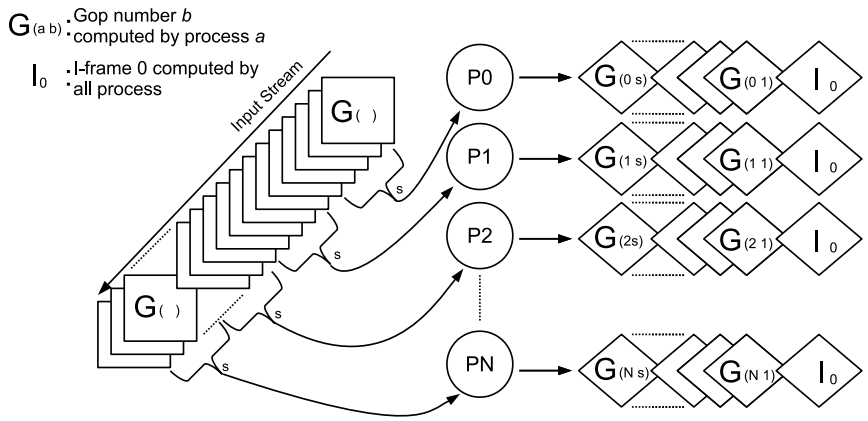

Fig. 2: Esquema del algoritmo DMG-1B.

la paralelización se realiza a nivel de secuencia. Otras aproximaciones se centran en el particionado de los datos a nivel de cuadro, como es el caso de la división en *slices* y *tiles* que explotan la paralelización dentro de un cuadro, teniendo en cuenta que tanto *slices* como *tiles* pueden ser codificados y decodificados de forma independiente. Siguiendo con la jerarquía de los datos de vídeo, se pueden encontrar esquemas de paralelización que trabajan a nivel de bloque (CU). Estos se centran en reducir el coste computacional de las operaciones que trabajan a nivel de CU, como la predicción *intra* y la estimación de movimiento.

Por lo tanto, se puede proponer un esquema de paralelización jerárquico desde el nivel más alto hasta el más bajo nivel de particionado de datos, en el proceso de codificación. Dependiendo de la arquitectura *hardware* donde se ejecuten las versiones paralelas de codificador, se podrán implementar diferentes esquemas paralelos. Por ejemplo, un *cluster* compuesto por un conjunto de nodos de procesamiento donde cada nodo está equipado con procesadores *multicore*, podría ser una arquitectura ideal para un algoritmo paralelo jerárquico consistente en la combinación de paralelización basada en GOPs (a nivel de secuencia) y paralelización basada en *tiles*/*slices*/WPP (a nivel de cuadro). Si cada nodo contara además con GPUs o FPGAs entonces se podría implementar también la paralelización a nivel de CU. Por tanto, el paralelismo se puede explotar a cada nivel de particionado de datos al mismo tiempo, mejorando la utilización de los recursos *hardware* disponibles en cada infraestructura particular.

El resto de este artículo está organizado de la siguiente manera: en la Seccción II se muestran las propuestas de paralelización jerárquica; en la Sección III analizamos el rendimiento de los algoritmos paralelos propuestos; finalmente, en la Sección IV se presentan las conclusiones finales del trabajo.

II. Estrategias jerárquicas de paralelización

En esta sección presentamos estrategias de paralelización jerárquica en dos niveles, que combinan dos aproximaciones basadas en GOP (a nivel de secuencia) con una aproximación basada en *slices* (a nivel de cuadro), con el fin de explotar de forma eficiente los recursos *hardware* de la arquitectura paralela y reducir de forma significativa el tiempo total de codificación de una secuencia de vídeo empleado por el estándar HEVC.

En [14], los autores presentan varias aproximaciones paralelas distribuidas basadas en GOPs para cada uno de los posibles modos de codificación en HEVC (*All Intra* (AI), *Low-Delay B* (LB), *Low-Delay P* (LP) y *Random Access* (RA)). En el algoritmo DMG-1G presentado en dicho trabajo (ver Figura 1) todos los procesos codifican el primer cuadro de tipo I y posteriormente, cuando un pro-

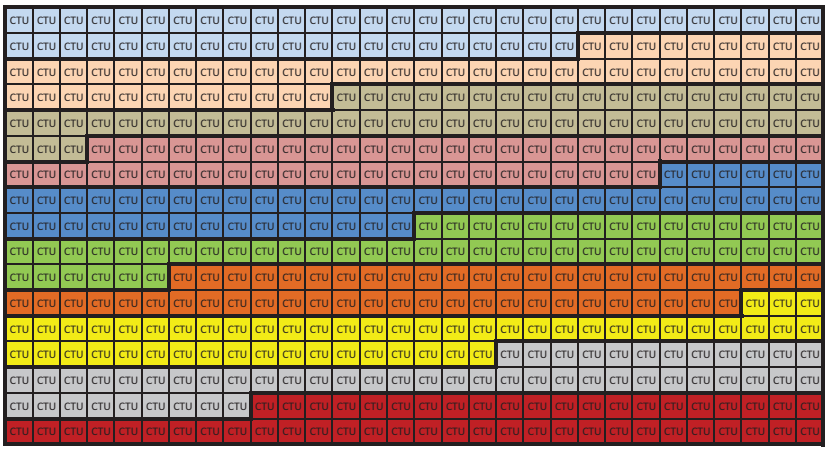

Fig. 3: División del un cuadro Full-HD (1920x1080) en 10 *slices* (51 CTUs/*slice*)

ceso termina su trabajo le pide al proceso coordinador el siguiente GOP a codificar. Los autores también presentan otro algoritmo llamado DMG-1B (ver Figura 2), en el que la secuencia de vídeo se divide en tantas partes como el número de procesadores disponibles y cada procesador realiza sus cálculos considerando las partes como secuencias de vídeo independientes. En el algoritmo DMG-1G, se necesita un proceso coordinador para realizar el balanceo de carga y las tareas de sincronización, mientras que en el algoritmo DMG-1B no se necesita ningún proceso coordinador, porque el balanceo de carga se realiza al comienzo del algoritmo.

Los resultados presentados en [14] muestran que el algoritmo DMG-1G es mucho más lento que el algoritmo DMG-1B, principalmente porque en el algoritmo DMG-1B los cuadros asignados a un proceso son contiguos en la secuencia de vídeo y por lo tanto la estimación de movimiento converge antes. El algoritmo DMG-1G obtiene *speed-ups* de hasta 8,45x, mientras que el algoritmo DMG-1B obtiene valores hasta 11,84x para la misma secuencia (BQ-Terrace 1920x1080) y el modo de codificación LB usando 12 procesadores. En cuanto al comportamiento R/D, ambos algoritmos producen un incremento en la tasa de bits final, especialmente el algoritmo DMG-1G, que incrementa de forma drástica la tasa de bits hasta el 159,5% para 12 procesos, mientras que el algoritmo DMG-1B introduce una penalización del 35,8%. Además, el algoritmo DMG-1G genera una reducción en la calidad del vídeo (PSNR - Peak Signal-to-Noise Ratio) de 1 dB cuando se usan 12 procesos. Por contra, en el algoritmo DMG-1B, el valor de PSNR permanece inalterado o incluso mejorado cuando el número de procesos se incrementa.

En cuanto a arquitecturas de memoria compartida se refiere, en [15], los autores proponen un algoritmo paralelo a nivel de cuadro basado en *slices* que es especialmente adecuado para dichas arquitecturas. En su propuesta, cada cuadro de vídeo se divide en tantos *slices* como el número de procesos paralelos disponibles, de forma que si se usan 12 procesos paralelos, se divide cada cuadro en 12 *slices* y cada *slice* lo codifica un proceso diferente (ver Figura 3). En este caso se necesita un mecanismo de sincronización ya que cada proceso codifica un *slice* diferente de forma paralela pero todos los procesos deben sincronizarse antes de codificar el siguiente cuadro. Los autores eligen una partición uniforme del cuadro para homogeneizar el tiempo que cada proceso necesita para codificar su *slice*, reduciendo así el tiempo inactivo causado por el mecanismo de sincronización. Después de una evaluación en profundidad de todos los modos de codificación de HEVC, se obtuvieron *speed-ups* de 9,3x para el modo AI y de aproximadamente 8,5x para el resto de modos de codificación usando 12 procesos. Estos buenos valores de *speed-up* tienen como contrapartida un incremento en la tasa de bits del 21% de media.

En nuestra propuesta jerárquica mezclamos la paralelización a nivel de GOP para arquitecturas de memoria distribuida con la paralelización basada en *slices* que se han presentado previamente. Las plataformas paralelas distribuidas están compuestas típicamente por múltiples nodos que contienen varios *cores*. La idea principal de esta aproximación jerárquica es dividir la secuencia de vídeo en GOPs o grupos de GOPs que se asignarán a cada nodo. Dentro de cada nodo, cada cuadro que pertenezca a un GOP se dividirá en tantos *slices* como el número de procesos dentro de dicho nodo. De esta forma, seremos capaces de sumar las ventajas del paralelismo de memoria distribuida y de memoria compartida.

En la siguiente sección mostraremos los resultados experimentales para la arquitectura jerárquica propuesta en este trabajo. También compararemos esta propuesta híbrida con las propuestas específicas mencionadas previamente.

III. Resultados experimentales

En esta sección analizaremos los algoritmos paralelos que combinan las técnicas basadas en GOPs y las técnicas basadas en *slices* para explotar las plataformas de memoria heterogénea. Analizaremos su comportamiento paralelo y su rendimiento R/D (PSNR y tasa de bits). Para los experimentos se ha utilizado el *software* de referencia de HEVC HM v16.3 [16]. Para implementar el paralelismo basado en *slices* se ha uti-

TABLA I: Secuencias de vídeo.

Acrónimo	Secuencia de vídeo	Resolución (*pixels*)	Cuadros por segundo	Cuadros totales	Duración de la secuencia (s)	Tiempo de codificación (s)
TRAFFI	Traffic	2560×1600	30	150	5	23859
PARKSC	ParkScene	1920×1080	24	240	10	21475
FOURPE	FourPeople	1280×720	60	600	10	15829
PARTSC	PartyScene	832×480	50	500	10	10714

lizado la API (Application Programming Interface) OpenMP v3.1 [17] y para implementar el paralelismo basado en GOPs se ha utilizado MPI (Message Passing Interface) v2.2 [18].

La plataforma paralela usada está compuesta de 14 nodos HP Proliant SL390 G7, donde cada nodo está equipado con 2 procesadores Intel Xeon X5660. Cada X5660 incluye 6 *cores* a 2,8GHz y como red de comunicaciones se ha utilizado QDR Infiniband. Las secuencias de vídeo usadas en los experimentos junto con sus características se pueden consultar en la Tabla I. En todos los experimentos presentados se ha utilizado el modo LB y se ha fijado el parámetro de cuantización (QP) a un valor de 22 (alta calidad). En la Tabla I se puede comprobar que estamos codificando 5 segundos de vídeo en la secuencia TRAFFI y 10 segundos en el resto de secuencias. En la columna "Tiempo de codificación"se muestra el coste temporal de codificar cada una de las secuencias que hemos evaluado con el algoritmo secuencial.

En primer lugar analizaremos el rendimiento del algoritmo basado en *slices* para ver cómo el particionado afecta tanto al *speed-up* como a la eficiencia. Esta propuesta se ha evaluado usando 3, 5, 7, 8, 9 y 11 procesos. La Tabla II muestra el *speed-up* y la eficiencia del algoritmo basado en *slices*. Para todas las secuencias de vídeo la eficiencia disminuye cuando el número de procesos disminuye. Además, en [15] se confirmó experimentalmente que tanto la tasa de bits como la eficiencia de codificación (R/D) empeoran cuando el número de *slices* aumenta. Como se observa en [15], si se incrementa el número de procesos de memoria compartida el tamaño de cada *slice* disminuye, lo que implica una pérdida tanto en la eficiencia de codificación como en el comportamiento paralelo (ver Tabla II). Si nos centramos en la secuencia PARKSC, el tiempo de computación para el algoritmo secuencial para codificar 1 segundo de vídeo es de media 2148 segundos, o sea, unos 36 minutos. Para reducir drásticamente el tiempo de codificación se necesita incrementar el número de procesos de cálculo.

En este trabajo se usan dos algoritmos paralelos basados en GOPs, el algoritmo DMG-1B y el algoritmo DMG-1G, citados anteriormente. Los dos algoritmos cambian el orden de codificación del algoritmo secuencial. Cuando se usa el modo de codificación LB, HEVC usa la predicción temporal para optimizar el proceso de codificación, es decir, HEVC usa información de cuadros cercanos previamente codificados. Por lo tanto, cambiar el orden de codificación provoca que la similaridad entre cuadros normalmente decrece, resultando en una disminución de la eficiencia de codificación. En las figuras 1 y 2 se puede comprobar cómo en el algoritmo DMG-1B el orden de codificación secuencial se rompe sólo una vez para cada proceso paralelo (excepto para el proceso raíz). Por contra, en el algoritmo DMG-1G, el orden de codificación secuencial puede cambiar en cada asignación de GOP, siendo los cambios más abruptos cuando el número de procesos crece, es decir, la distancia o número de GOPs entre dos GOPs consecutivos asignados a un proceso aumenta. Si nos fijamos en los resultados mostrados en la Tabla III, se observa que el algoritmo DMG-1B obtiene eficiencias cercanas a las ideales, siendo ligeramente inferior para las dos secuencias de menor resolución. En el caso de la secuencia TRAFFI y considerando que el tamaño del GOP para el modo LB es de 4 cuadros, esta secuencia tiene sólo 38 GOPs, por lo tanto no podemos incrementar de forma drástica el número de procesos distribuidos, aunque tengamos recursos disponibles. Sin embargo, aunque para el algoritmo DMG-1G las eficiencias obtenidas son siempre superiores al 80%, este algoritmo proporciona un rendimiento inferior que el algoritmo DMG-1B. Como se ha comentado en la Sección II, el algoritmo DMG-1G incluye un proceso coordinador que asigna dinámicamente, en tiempo de ejecución, los GOPs a codificar por cada proceso. Por lo tanto, incluye un sobrecoste paralelo debido a las comunicaciones entre los procesos que codifican los GOPs y el proceso coordinador. Es importante remarcar que este algoritmo incluye intrínsecamente un sistema de balanceo de carga.

Para reducir el tiempo total de codificación es necesario mezclar ambas técnicas paralelas, GOPs (DMG-1B, DMG-1G) y *slices*, explotando la potencia de cálculo de las plataformas paralelas de memoria heterogénea. En primer lugar analizaremos el comportamiento paralelo del algoritmo DMG-1G-SLICE, donde los procesos MPI se alinean con el número de nodos (multiprocesadores de memoria distribuida), y los procesos OpenMP coinciden con el número de *cores* (procesadores de memoria compartida) usados en cada nodo. La Tabla IV muestra el *speed-up* obtenido para dicho algoritmo. Se observan valores de *speed-up* de hasta 31,2x y 26,5x usando 4 nodos, para las secuencias de mayor resolución (TRAFFI) y menor resolución (PARTSC) respectivamente, y de hasta 63,0x y 68,1x cuando se usan 10 nodos. Las eficiencias correspondientes

TABLA II: *Speed-up* y eficiencia del algoritmo basado en *slices*.

(a) TRAFFI (2560 × 1600)

Núm. procesos	*Speed-up*	Eficiencia
3	2,6x	86%
5	3,9x	78%
7	4,7x	67%
9	5,3x	59%
11	5,9x	54%

(b) PARKSC (1920 × 1080)

Núm. procesos	*Speed-up*	Eficiencia
3	2,7x	90%
5	4,2x	84%
7	5,2x	75%
9	5,9x	65%
11	6,5x	59%

(c) FOURPE (1280 × 720)

Núm. procesos	*Speed-up*	Eficiencia
3	2,2x	74%
5	3,4x	68%
7	4,5x	64%
9	5,7x	63%
11	7,0x	63%

(d) PARTSC (832 × 480)

Núm. procesos	*Speed-up*	Eficiencia
3	2,6x	86%
5	4,2x	84%
7	5,7x	81%
9	7,0x	78%
11	8,1x	74%

TABLA III: Eficiencia de los algoritmos basados en GOPs.

(a) TRAFFI (2560 × 1600)

Núm. procesos	DMG-1B	DMG-1G
2	100%	95%
4	100%	88%
6	100%	83%
8	100%	86%
10	100%	86%

(b) PARKSC (1920 × 1080)

Núm. procesos	DMG-1B	DMG-1G
2	98%	87%
4	100%	85%
6	100%	83%
8	100%	79%
10	100%	81%

(c) FOURPE (1280 × 720)

Núm. procesos	DMG-1B	DMG-1G
2	95%	92%
4	92%	89%
6	93%	85%
8	92%	86%
10	94%	83%

(d) PARTSC (832 × 480)

Núm. procesos	DMG-1B	DMG-1G
2	93%	91%
4	93%	87%
6	95%	83%
8	96%	82%
10	95%	81%

a dichos valores de *speed-up* son del 71% y del 60% respectivamente para 4 nodos y del 57% y del 62% respectivamente para 10 nodos.

A continuación analizaremos el algoritmo heterogéneo DMG-1B-SLICE, que combina el algoritmo DMG-1B con el algoritmo basado en *slices*. En la Tabla V se muestran los valores de *speed-up* correspondientes al algoritmo DMG-1B-SLICE. Con este algoritmo se obtienen valores de hasta 60,8x para la secuencia PARTSC usando 10 nodos y 11 procesos OpenMP. Si comparamos los resultados de las tablas IV y V, el algoritmo DMG-1G-SLICE obtiene mejor rendimiento paralelo que el algoritmo DMG-1B-SLICE, justo al contrario de lo esperado (ya que DMG-1B obtiene mejores resultados que DMG-1G). DMG-1G-SLICE incluye un sistema de balanceo de carga que mejora el comportamiento del algoritmo. Se ha de tener en cuenta que el coste computacional de codificar un cuadro difiere de unos a otros debido al contenido de cada cuadro. Dichas diferencias son compensadas por el algoritmo DMG-1G-SLICE pero no por el algoritmo DMG-1B-SLICE. Como se puede observar, ambas estrategias paralelas jerárquicas superan ampliamente los valores de *speed-up* obtenidos por las aproximaciones independientes basadas en GOPs y en *slices*.

Una vez analizados los algoritmos propuestos con respecto al rendimiento paralelo, analizaremos el rendimiento R/D en términos de PSNR y tasa de bits. Las tablas VI and VII muestran los valores de PSNR para ambos algoritmos. La primera fila de cada tabla (para 1 proceso MPI y 1 proceso OpenMP) corresponde a los valores PSNR de la ejecución del algoritmo secuencial. La Tabla VI muestra una ligera degradación en la calidad del vídeo reconstruído (PSNR) para el algoritmo DMG-1G-SLICE. Sin embargo, los resultados para el algoritmo DMG-1B-SLICE (Tabla VII) muestran valores de PSNR muy similares a los del algoritmo secuencial o incluso en algunos casos una ligera mejora.

Finalmente, las tablas VIII y IX muestran la tasa de bits correspondiente a los mismos experimentos que los mostrados en las tablas VI y VII. Se puede observar un incremento en la tasa de bits que no se debe despreciar. Este incremento no se puede evitar en el modo de codificación LB, porque es debido a la alteración de las dependencias dentro de cada GOP. Cuando la dependencia interna de un GOP se

TABLA IV: *Speed-up* del algoritmo DMG-1G-SLICE para 4 y 10 procesos MPI.

(a) TRAFFI (2560 × 1600)

Procesos MPI	Procesos OpenMP	*Speed-up*
4	3	9,5x
4	5	15,6x
4	7	20,8x
4	9	25,9x
4	11	31,2x

(b) PARTSC (832 × 480)

Procesos MPI	Procesos OpenMP	*Speed-up*
4	3	9,0x
4	5	14,5x
4	7	19,7x
4	9	23,6x
4	11	26,5x

(c) TRAFFI (2560 × 1600)

Procesos MPI	Procesos OpenMP	*Speed-up*
10	3	21,2x
10	5	33,8x
10	7	45,5x
10	9	55,1x
10	11	63,0x

(d) PARTSC (832 × 480)

Procesos MPI	Procesos OpenMP	*Speed-up*
10	3	22,5x
10	5	36,8x
10	7	49,0x
10	9	60,2x
10	11	68,1x

TABLA V: *Speed-up* del algoritmo DMG-1B-SLICE para 4 y 10 procesos MPI.

(a) TRAFFI (2560 × 1600)

Procesos MPI	Procesos OpenMP	*Speed-up*
4	3	10,7x
4	5	17,2x
4	7	20,6x
4	9	23,0x
4	11	25,1x

(b) PARTSC (832 × 480)

Procesos MPI	Procesos OpenMP	*Speed-up*
4	3	9,2x
4	5	13,3x
4	7	16,0x
4	9	18,1x
4	11	19,6x

(c) TRAFFI (2560 × 1600)

Procesos MPI	Procesos OpenMP	*Speed-up*
10	3	24,9x
10	5	36,9x
10	7	45,0x
10	9	51,7x
10	11	57,5x

(d) PARTSC (832 × 480)

Procesos MPI	Procesos OpenMP	*Speed-up*
10	3	25,1x
10	5	39,9x
10	7	51,8x
10	9	58,9x
10	11	60,8x

cambia los siguientes cuadros se codificarán probablemente sin poder aprovechar las redundancias temporales presentes en cuadros contiguos y por lo tanto se tendrá que codificar una mayor cantidad de datos residuales, causando un aumento en la tasa de bits. El aumento en la tasa de bits es significativamente superior en el algoritmo DMG-1G-SLICE en el que la cantidad de dependencias alteradas en los GOPs es superior a la del algoritmo DMG-1B-SLICE.

IV. Conclusiones

En este artículo se han presentado dos aproximaciones paralelas jerárquicas para el estándar de codificación de vídeo HEVC que combinan la paralelización a nivel de secuencia (basada en GOPs) que se ajusta perfectamente a las arquitecturas distribuidas y la paralelización a nivel de cuadro (basada en *slices*) que es especialmente adecuada para las plataformas con memoria compartida. El algoritmo paralelo DMG-1G-SLICE requiere la presencia de un proceso coordinador para realizar las tareas de balanceo de carga, mientras que el algoritmo DMG-1B-SLICE no requiere dicha coordinación.

Se han analizado ambos algoritmos en términos de *speed-up* y de rendimiento R/D. De los resultados presentados en este trabajo se puede llegar a la conclusión de que el algoritmo DMG-1G-SLICE obtiene mejores valores de *speed-up*, especialmente para las secuencias de vídeo de alta resolución. Para estas resoluciones se obtienen valores de hasta 68,1x cuando se utilizan 10 nodos distribuidos y 11 procesos por nodo para el algoritmo DMG-1G-SLICE y valores de hasta 60,8x para la misma configuración híbrida usando el algoritmo DMG-1B-SLICE.

Con respecto a la degradación en términos de R/D, ambos algoritmos producen un incremento en la tasa de bits, especialmente el algoritmo DMG-1G-SLICE. El máximo incremento en la tasa de bits producido por el algoritmo DMG-1G-SLICE es del 56% usando 10 nodos y 11 procesos por nodo para la secuencia con la máxima resolución. El máximo incremento en la tasa de bits del algoritmo DMG-1B-SLICE es del 41% para la misma configuración. Además, cuando se utiliza el algoritmo DMG-1G-SLICE la calidad

TABLA VI: Valores de PSNR para el algoritmo DMG-1G-SLICE para 4 y 10 procesos MPI.

(a) TRAFFI (2560 × 1600)

Procesos MPI	Procesos OpenMP	PSNR (dB)
1	1	41,8111
4	1	41,5535
4	3	41,5548
4	5	41,5499
4	7	41,5495
4	9	41,5472
4	11	41,5441

(b) PARTSC (832 × 480)

Procesos MPI	Procesos OpenMP	PSNR (dB)
1	1	39,0766
4	1	38,7135
4	3	38,7078
4	5	38,7036
4	7	38,6984
4	9	38,6970
4	11	38,6945

(c) TRAFFI (2560 × 1600)

Procesos MPI	Procesos OpenMP	PSNR (dB)
1	1	41,8111
10	1	41,5019
10	3	41,4996
10	5	41,4972
10	7	41,4958
10	9	41,4943
10	11	41,4918

(d) PARTSC (832 × 480)

Procesos MPI	Procesos OpenMP	PSNR (dB)
1	1	39,0766
10	1	38,5310
10	3	38,5265
10	5	38,5212
10	7	38,5154
10	9	38,5143
10	11	38,5147

TABLA VII: Valores de PSNR para el algoritmo DMG-1B-SLICE para 4 y 10 procesos MPI.

(a) TRAFFI (2560 × 1600)

Procesos MPI	Procesos OpenMP	PSNR (dB)
1	1	41,8111
4	1	41,8017
4	3	41,8003
4	5	41,7977
4	7	41,7971
4	9	41,7954
4	11	41,7920

(b) PARTSC (832 × 480)

Procesos MPI	Procesos OpenMP	PSNR (dB)
1	1	39,0766
4	1	39,1137
4	3	39,1084
4	5	39,1053
4	7	39,0985
4	9	39,0939
4	11	39,0920

(c) TRAFFI (2560 × 1600)

Procesos MPI	Procesos OpenMP	PSNR (dB)
1	1	41,8111
10	1	41,8236
10	3	41,8215
10	5	41,8196
10	7	41,8172
10	9	41,8155
10	11	41,8127

(d) PARTSC (832 × 480)

Procesos MPI	Procesos OpenMP	PSNR (dB)
1	1	39,0766
10	1	39,1465
10	3	39,1410
10	5	39,1374
10	7	39,1309
10	9	39,1281
10	11	39,1266

PSNR sufre una ligera degradación, mientras que el algoritmo DMG-1B-SLICE no introduce pérdidas de calidad.

El algoritmo DMG-1B-SLICE resulta ser el más apropiado de los dos algoritmos propuestos en este trabajo, ya que sufre de un menor incremento en la tasa de bits, sin introducir pérdidas en la calidad del vídeo, obteniendo una aceleración del tiempo de codificación de hasta 60,8x, aunque en términos generales obtenga menores valores de *speed-up* que el algoritmo DMG-1G-SLICE.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio Español de Economía y Competitividad mediante el proyecto TIN2015-66972-C5-4-R, cofinanciado por fondos FEDER (MINECO/FEDER/UE).

Referencias

[1] J. Ohm, G.J. Sullivan, H. Schwarz, Thiow Keng Tan, and T. Wiegand, "Comparison of the coding efficiency of video coding standards - including high efficiency video coding (hevc)," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1669–1684, 2012.

[2] ITU-T and ISO/IEC JTC 1, "Advanced video coding for generic audiovisual services," *ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC) version 16, 2012*, 2012.

[3] G.J. Sullivan, J.R. Ohm, W.J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *Circuits and systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1648 –1667, December 2012.

[4] F. Bossen, B. Bross, K. Suhring, and D. Flynn, "HEVC

TABLA VIII: Tasa de bits para el algoritmo DMG-1G-SLICE para 4 y 10 procesos MPI.

(a) TRAFFI (2560 × 1600)

Procesos MPI	Procesos OpenMP	Tasa de bits (Kbps)
1	1	13765,021
4	1	18873,730
4	3	18929,608
4	5	18962,306
4	7	18985,240
4	9	19018,432
4	11	19066,938

(b) PARTSC (832 × 480)

Procesos MPI	Procesos OpenMP	Tasa de bits (Kbps)
1	1	8047,106
4	1	10909,551
4	3	10957,132
4	5	11003,734
4	7	11040,546
4	9	11069,246
4	11	11085,995

(c) TRAFFI (2560 × 1600)

Procesos MPI	Procesos OpenMP	Tasa de bits (Kbps)
1	1	13765,021
10	1	21352,982
10	3	21406,906
10	5	21444,557
10	7	21453,266
10	9	21501,931
10	11	21563,352

(d) PARTSC (832 × 480)

Procesos MPI	Procesos OpenMP	Tasa de bits (Kbps)
1	1	8047,106
10	1	12611,406
10	3	12651,714
10	5	12715,100
10	7	12759,924
10	9	12796,774
10	11	12787,986

TABLA IX: Tasa de bits para el algoritmo DMG-1B-SLICE para 4 y 10 procesos MPI.

(a) TRAFFI (2560 × 1600)

Procesos MPI	Procesos OpenMP	Tasa de bits (Kbps)
1	1	13765,021
4	1	15632,483
4	3	15655,669
4	5	15672,304
4	7	15706,214
4	9	15728,826
4	11	15753,096

(b) PARTSC (832 × 480)

Procesos MPI	Procesos OpenMP	Tasa de bits (Kbps)
1	1	8047,106
4	1	8289,771
4	3	8318,211
4	5	8350,546
4	7	8373,684
4	9	8398,278
4	11	8405,666

(c) TRAFFI (2560 × 1600)

Procesos MPI	Procesos OpenMP	Tasa de bits (Kbps)
1	1	13765,021
10	1	19327,581
10	3	19354,576
10	5	19380,992
10	7	19412,669
10	9	19430,491
10	11	19463,938

(d) PARTSC (832 × 480)

Procesos MPI	Procesos OpenMP	Tasa de bits (Kbps)
1	1	8047,106
10	1	8676,546
10	3	8707,154
10	5	8739,460
10	7	8768,247
10	9	8794,862
10	11	8803,378

complexity and implementation analysis," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1685–1696, 2012.

[5] M. Alvarez-Mesa, C.C. Chi, B. Juurlink, V. George, and T. Schierl, "Parallel video decoding in the emerging HEVC standard," in *International Conference on Acoustics, Speech, and Signal Processing, Kyoto*, March 2012, pp. 1–17.

[6] E.A. Ayele and S.B.Dhok, "Review of proposed high efficiency video coding (HEVC) standard," *International Journal of Computer Applications*, vol. 59, no. 15, pp. 1–9, 2012.

[7] Minhua Zhou, "AHG10: Configurable and CU-group level parallel merge/skip," Tech. Rep., Joint Collaborative Team on Video Coding-H0082, 2012.

[8] Qin Yu, Liang Zhao, and Siwei Ma, "Parallel AMVP candidate list construction for HEVC," in *VCIP'12*, 2012, pp. 1–6.

[9] Jie Jiang, Baolong Guo, Wei Mo, and Kefeng Fan, "Block-based parallel intra prediction scheme for HEVC," *Journal of Multimedia*, vol. 7, no. 4, pp. 289 –294, August 2012.

[10] L. Bolc, R. Tadeusiewicz, L. Chmielewski, and K. Wojciechowski, "Diamond scanning order of image blocks for massively parallel HEVC compression," *Lecture Notes in Computer Science*, vol. 7594, pp. 172 –179, 2012.

[11] C. Yan, Y. Zhang, F. Dai, and L. Liang, "Efficient parallel framework for HEVC deblocking filter on many-core platform," in *Data Compression Conference (DCC)*, 2013.

[12] H. Migallón, P. Piñol, O. López-Granado, and M. P. Malumbres, "Subpicture parallel approaches of HEVC video encoder," in *2014 International Conference on Computational and Mathematical Methods in Science and Engineering*, 2014, vol. 1, pp. 927–938.

[13] P. Piñol, H. Migallón, O. López-Granado, and M. P. Malumbres, "Parallel strategies analysis over the HEVC encoder," *The Journal of Supercomputing*, vol. 70, no. 2, pp. 671–683, 2014.

[14] H. Migallón, V. Galiano, P. Piñol, O. López-Granado,

and M. P. Malumbres, "Distributed memory parallel approaches for HEVC encoder," *The Journal of Supercomputing*, vol. 73, no. 1, pp. 164–175, 2017.

[15] P. Piñol, H. Migallón, O. López-Granado, and M. P. Malumbres, "Slice-based parallel approach for HEVC encoder," *The Journal of Supercomputing*, vol. 71, no. 5, pp. 1882–1892, 2015.

[16] HEVC Reference Software, https://hevc.hhi.fraunhofer.de/svn/svn_-HEVCSoftware/ tags/HM-16.3/, ," .

[17] "OpenMP Application Program Interface, version 3.1," *OpenMP Architecture Review Board.*, 2011.

[18] MPI Forum, "MPI: A Message-Passing Interface Standard. Version 2.2," September 4th 2009.