



UNIVERSIDAD DE MURCIA

Departamento de Ingeniería y Tecnología de Computadores

# **Diseño, Evaluación y Optimización de la Transformada Wavelet para Codificación de Vídeo Médico en Arquitecturas Monoprocesador**

TESIS PROPUESTA PARA  
LA OBTENCIÓN DEL GRADO DE

DOCTOR EN INFORMÁTICA

Presentada por:  
Gregorio Bernabé García

Supervisada por:  
José Manuel García Carrasco  
José González González

Murcia, Septiembre 2004



A mis padres

A Paqui



## Agradecimientos

---

Durante el desarrollo de esta tesis he recibido el apoyo de muchas personas e instituciones a las que quiero expresar mi gratitud.

En primer lugar, a mis directores de tesis. A José Manuel, por toda la confianza depositada y por todo lo que he podido aprender a su lado. A Pepe, por su incansable esfuerzo y dedicación, el entusiasmo con el que ha realizado este trabajo, la transmisión de unas formas de trabajo, una dirección excelente y su especial amistad.

A José Duato, por su inestimable ayuda en la realización de este trabajo, su disponibilidad para la investigación y su amistad.

Al Hospital Recoletas de Albacete por la donación de secuencias de vídeo médico sin las que hubiera sido imposible realizar este trabajo.

A mis compañeros de departamento Manolo, Juan Luis, Javi, Lorenzo, Juan-Pe, Antonio, Pilar, Pedro, Juan-Pi, Diego, Oscar, etc, con los que he trabajado, discutido y he pasado momentos increíbles e inolvidables. Quiero hacer una mención especial a Manolo, que me ha soportado como compañero de despacho durante seis largos años y se ha convertido en más que un amigo. Una nota destacada merece la disponibilidad y paciencia de Pilar para resolverme cualquier problema relacionado con Linux. Tampoco quiero olvidarme de Paco Alfaro, al que siempre consideraré mi compañero y amigo.

A mi hermano, por los buenos momentos que hemos pasado. A mis abuelos y mis tíos, por el apoyo y confianza mostrada hacia mí.

A Paqui, por saber comprenderme y ayudarme en los momentos difíciles. Su cariño, amor y buen humor me han hecho olvidar cualquier problema y superar todos los obstáculos encontrados.

De una manera muy especial para mí, agradecer el esfuerzo realizado por mis padres durante toda su vida para proporcionarme unos estudios y una forma de vivir. Su perfecta comprensión en todo momento, su inestimable apoyo, su forma de afrontar mi mal humor, sus sabios consejos y su enorme cariño me han permitido llegar hasta aquí.



## Resumen

---

El aumento espectacular del uso de aplicaciones multimedia en las diferentes especialidades médicas y del volumen de información médica (imágenes y vídeos) en los hospitales actuales, ponen de manifiesto la necesidad de las técnicas de compresión para reducir tanto la cantidad de información almacenada como los datos a transmitir. La información médica cuenta con propiedades particulares (seguridad, privacidad, disponibilidad, calidad, etc), y medidas legales que no hacen más que aumentar los espacios de almacenamiento y el ancho de banda necesario para transmitir la información. Los estándares tradicionales de compresión de imágenes y vídeo se han basado en la transformada discreta del coseno. Diversos inconvenientes y la aparición de la transformada discreta de wavelet han originado nuevas vías de investigación.

En este trabajo presentamos un nuevo codificador de vídeo médico con pérdida de información basado en la transformada de wavelet 3D. El codificador obtiene grandes tasas de compresión y una calidad excelente desde el punto de vista médico, ya que no se aprecian diferencias y no aumenta el grado de distorsión en los vídeos reconstruidos.

La ejecución de las aplicaciones multimedia en tiempo real en un ordenador personal es uno de los principales inconvenientes que presentan dichas aplicaciones. En esta tesis se desarrollan varias propuestas para permitir la compresión y transmisión en tiempo real de la transformada de wavelet 3D en arquitecturas monoprocesador. En las diferentes propuestas se hará uso de las extensiones multimedia, técnicas de división en bloques (*blocking*), reuso de operaciones en punto flotante, desenrollado de bucles o técnicas de pre-búsqueda. La integración de las propuestas anteriores supone un primer paso para la automatización de la convergencia entre las aplicaciones multimedia y las extensiones multimedia.

Con la llegada de las nuevas arquitecturas monoprocesador que permiten ejecutar simultáneamente múltiples hilos (*Simultaneous Multithreading* (SMT)), se realizan varias propuestas que explotan de forma eficiente el paralelismo a nivel de hilos. Las distintas propuestas se diferencian en la forma de realizar la descomposición de la aplicación original. La codificación de vídeo

médico basada en la transformada de wavelet 3D obtiene mejores resultados cuando se realiza una descomposición funcional que en el caso de una descomposición basada en el dominio de los datos. Los compiladores actuales no son capaces de llevar a cabo una paralelización automática eficaz basada en una descomposición funcional, ya que se requiere un conocimiento del funcionamiento de la aplicación. Por tanto, las diferentes propuestas que obtienen los mayores beneficios se obtienen mediante la intervención directa del programador o paralelización manual.

En resumen, en esta tesis se propone un codificador de vídeo médico basado en la transformada de wavelet 3D que aumenta la tasa de compresión entre un 40 % y un 70 % para una misma calidad de las secuencias de vídeo reconstruidas. A su vez, el tiempo de ejecución se reduce en un 363 %.



# Índice general

---

Índice general	I
Índice de figuras	V
Índice de figuras	V
Índice de tablas	IX
Índice de tablas	IX
<b>1. Introducción</b>	<b>1</b>
1.1. Aplicaciones multimedia . . . . .	1
1.2. Características de la información médica . . . . .	3
1.3. Necesidades de compresión . . . . .	5
1.4. Técnicas de compresión . . . . .	6
1.5. El entorno de trabajo . . . . .	8
1.6. Motivación y aportaciones . . . . .	11
1.6.1. Motivación . . . . .	11
1.6.2. Aportaciones de la tesis . . . . .	13
1.7. Organización de la tesis . . . . .	14
<b>2. Compresión de imágenes y secuencias de vídeo</b>	<b>15</b>
2.1. La transformada matemática . . . . .	19

2.1.1.	La transformada de Fourier . . . . .	19
2.1.2.	La transformada de wavelet . . . . .	20
2.2.	La cuantificación . . . . .	28
2.2.1.	Cuantificación uniforme . . . . .	28
2.2.2.	Cuantificación no uniforme . . . . .	28
2.2.3.	Cuantificación adaptativa . . . . .	29
2.3.	La codificación entrópica . . . . .	29
2.3.1.	Codificación Huffman . . . . .	30
2.3.2.	Codificación Aritmética . . . . .	32
2.3.3.	Codificación <i>Run-Length</i> . . . . .	34
2.4.	Codificadores basados en una transformada . . . . .	35
2.4.1.	Estándares de compresión basados en una transformada . . . . .	35
2.4.2.	Métodos de compresión basados en la transformada de wavelet 2D para imágenes y 3D para vídeo e imágenes volumétricas . . . . .	43
2.5.	Otras aplicaciones de la transformada de wavelet . . . . .	46
2.6.	Conclusiones . . . . .	49
<b>3.</b>	<b>Codificador de vídeo médico basado en la transformada de wavelet 3D</b>	<b>51</b>
3.1.	Introducción . . . . .	51
3.2.	Trabajo relacionado . . . . .	54
3.2.1.	Codificadores basados en la transformada de wavelet 2D . . . . .	54
3.2.2.	Codificadores basados en la transformada de wavelet 3D . . . . .	60
3.3.	Codificador basado en la transformada de wavelet 3D . . . . .	63
3.3.1.	La transformada de wavelet 3D. Elección de la función wavelet madre . . .	63
3.3.2.	La umbralización . . . . .	64
3.3.3.	La cuantificación y la codificación entrópica . . . . .	66
3.4.	Optimizaciones sobre el cuantificador y el codificador entrópico . . . . .	69
3.4.1.	<i>Run-length</i> binario 3D inteligente . . . . .	70

3.4.2. Codificación hexadecimal . . . . .	72
3.4.3. Codificación aritmética . . . . .	75
3.5. Evaluación de resultados . . . . .	76
3.5.1. Métricas para determinar la calidad del vídeo comprimido . . . . .	76
3.5.2. Entorno de trabajo . . . . .	76
3.5.3. Evaluación del codificador base . . . . .	78
3.5.4. Evaluación del codificador mejorado . . . . .	82
3.5.5. Comparación del codificador base y mejorado con MPEG-2 y EZW . . . . .	86
3.6. Conclusiones . . . . .	88
<b>4. La transformada de wavelet 3D en tiempo real</b>	<b>91</b>
4.1. Introducción . . . . .	91
4.2. Trabajo relacionado . . . . .	94
4.2.1. Técnicas de división en bloques . . . . .	94
4.2.2. Extensiones Multimedia . . . . .	95
4.3. Técnicas de división en bloques . . . . .	98
4.3.1. Técnicas de división en cubos . . . . .	99
4.3.2. Técnicas de división en rectángulos . . . . .	102
4.4. Optimización de la técnica de división en rectángulos . . . . .	103
4.4.1. Reuso de cálculos . . . . .	103
4.4.2. Uso de las Extensiones SIMD (SSE) . . . . .	104
4.4.3. Desenrollado de bucles y pre-búsqueda de datos . . . . .	107
4.4.4. Vectorización por columnas . . . . .	108
4.5. Evaluación de resultados . . . . .	111
4.5.1. Entorno de trabajo . . . . .	111
4.5.2. Evaluación del tiempo de ejecución de las técnicas de división en bloques . . . . .	111
4.5.3. Análisis del resto de propuestas . . . . .	115
4.5.4. Análisis del comportamiento de la memoria cache . . . . .	117

4.6. Conclusiones . . . . .	122
<b>5. La transformada de wavelet 3D en un entorno multi-hilo</b>	<b>125</b>
5.1. Introducción . . . . .	125
5.2. La tecnología <i>Hyper-Threading</i> . . . . .	128
5.3. Trabajo relacionado . . . . .	130
5.3.1. Paralelización de la transformada de wavelet . . . . .	130
5.3.2. La tecnología <i>Hyper-Threading</i> . . . . .	132
5.4. Optimización del codificador basado en wavelet usando la tecnología <i>Hyper-Threading</i>	134
5.4.1. Descomposición multi-hilo basada en el dominio de los datos . . . . .	135
5.4.2. Descomposición multi-hilo basada en una división funcional . . . . .	136
5.5. Evaluación de resultados . . . . .	142
5.5.1. Entorno de trabajo . . . . .	142
5.5.2. Descomposición basada en el dominio de los datos . . . . .	143
5.5.3. Descomposición basada en una división funcional . . . . .	145
5.6. Conclusiones . . . . .	148
<b>6. Conclusiones y trabajo futuro</b>	<b>151</b>
6.1. Conclusiones . . . . .	151
6.2. Resultados de la tesis . . . . .	155
6.3. Financiación disfrutada . . . . .	158
6.4. Trabajo futuro . . . . .	159
<b>Referencias</b>	<b>161</b>

## Índice de figuras

---

2.1. Diagrama de bloques de un codificador basado en una transformada matemática . . .	18
2.2. Funciones básicas de Fourier y wavelet (Daubechies). Ventanas tiempo-frecuencia y representación en el plano tiempo-frecuencia. . . . .	22
2.3. Descomposición en sub-bandas de una imagen tras una iteración (parte izquierda) y tres iteraciones (parte derecha) de la transformada de wavelet 2D . . . . .	26
2.4. Descomposición de las imágenes tras dos iteraciones (mono –parte izquierda–) y tres iteraciones (Lena –parte derecha–) de la transformada de wavelet 2D . . . . .	26
2.5. Descomposición de una secuencia de vídeo tras una iteración de la transformada de wavelet 3D . . . . .	27
2.6. Ejemplo de construcción de un árbol de Huffman . . . . .	32
2.7. Diagrama de bloques del codificador JPEG . . . . .	35
2.8. Tipos de imágenes en un codificador MPEG . . . . .	39
2.9. Diagrama de bloques del codificador de vídeo MPEG . . . . .	40
2.10. Procesos de un codificador basado en la transformada de wavelet 2D (2D-FWT) . .	44
2.11. Procesos de un codificador basado en la transformada de wavelet 3D (3D-FWT) . .	44
3.1. Procesos de un codificador basado en la transformada de wavelet 3D (3D-FWT) . .	52
3.2. Procesos de un decodificador basado en la transformada de wavelet 3D inversa (3D-IFWT) . . . . .	52
3.3. Relaciones entre coeficientes wavelet de diferentes sub-bandas (Izquierda). Orden de búsqueda en el proceso de codificación <i>zerotree</i> (Derecha). . . . .	55
3.4. Asignación de bits por capas para 2 y 3 iteraciones de la transformada de wavelet .	67

3.5. a) Resultado de dos iteraciones de la transformada de wavelet sobre una imagen de 16x16 pixels. b) Representación decimal de la sub-banda LL. c) Representación binaria de la sub-banda LL y algunos ejemplos de cadenas de ceros consecutivos (rectángulos) . . . . .	68
3.6. Asignación de bits por sub-cubos para dos iteraciones de la transformada de wavelet 3D. . . . .	71
3.7. Tasa de compresión para la secuencia de vídeo <i>Corazón</i> . . . . .	83
3.8. Tasa de compresión para la secuencia de vídeo <i>Catéter</i> . . . . .	83
3.9. Tasa de compresión para la secuencia de vídeo <i>Mano</i> . . . . .	84
3.10. Contribución de cada una de las mejoras en la secuencia <i>Corazón</i> . . . . .	85
3.11. Tasas de compresión del codificador base, codificador mejorado, MPEG-2 y EZW para las secuencias <i>Corazón</i> y <i>Mano</i> . . . . .	87
4.1. Técnicas de división en cubos . . . . .	100
4.2. Algoritmo de la Transformada de Wavelet 1D con la función Daub-4 . . . . .	101
4.3. Técnicas de división en rectángulos . . . . .	102
4.4. Reutilización en las técnicas de división en rectángulos . . . . .	104
4.5. Cálculo de los cuatro primeros coeficientes wavelet pasa-baja. . . . .	105
4.6. Pasos para calcular los cuatro primeros coeficientes wavelet pasa-baja usando los registros SIMD . . . . .	107
4.7. Vectorización por columnas para un <i>frame</i> de 6 filas y 12 columnas. . . . .	110
4.8. Tiempo de ejecución de las técnicas de división en bloques para la secuencia de vídeo médico <i>Corazón</i> . . . . .	113
4.9. Contribución de cada una de las propuestas al tiempo de ejecución . . . . .	114
4.10. Tiempo de ejecución de las diferentes optimizaciones para la secuencia de vídeo médico <i>Corazón</i> . . . . .	115
4.11. Líneas de entrada de la Cache de Datos L1 para la secuencia de vídeo <i>Corazón</i> . .	118
4.12. Líneas de entrada de la Cache Unificada L2 para la secuencia de vídeo <i>Corazón</i> . .	118
4.13. Líneas de entrada de la Cache de Datos L1 divididas por dimensiones para la técnica de división en rectángulos . . . . .	119

4.14. Líneas de entrada de la Cache de Datos L1 divididas por dimensiones para la técnica de división en rectángulos . . . . .	119
4.15. Instrucciones en punto flotante ejecutadas para la secuencia de vídeo <i>Corazón</i> . . .	120
5.1. (a) Multiprocesador con dos procesadores sin tecnología <i>Hyper-Threading</i> . (b) Multiprocesador con dos procesadores con tecnología <i>Hyper-Threading</i> . . . . .	129
5.2. Descomposición multi-hilo basada en el dominio de los datos . . . . .	135
5.3. Descomposición multi-hilo basada en un hilo de ayuda . . . . .	138
5.4. Tres métodos para el codificador 3D-FWT (a) Método secuencial; (b) Método funcional multi-hilo; (c) Método funcional balanceado correctamente . . . . .	140
5.5. <i>Speed-ups</i> para la descomposición multi-hilo basada en el dominio de los datos . .	143
5.6. <i>Speed-ups</i> para la descomposición multi-hilo basada en una división funcional . . .	147





## Índice de tablas

---

2.1. Tabla de codificación Huffman . . . . .	33
3.1. Calidad en PSNR de la imagen <i>Lena</i> para distintos codificadores y tasas de compresión . . . . .	59
3.2. Coeficientes para las funciones wavelet madre Daub-4, Bathlet-4, Haar y Daub-8 . . . . .	65
3.3. Posibles valores del primer símbolo de codificación hexadecimal . . . . .	74
3.4. Tasa de compresión y PSNR de la secuencia <i>Corazón</i> para las funciones wavelet madre Haar, Daub-4 y Daub-8 . . . . .	79
3.5. Tasa de compresión y PSNR de la secuencia <i>Catéter</i> para las funciones wavelet madre Haar, Daub-4 y Daub-8 . . . . .	80
3.6. Tasa de compresión y PSNR de la secuencia <i>Mano</i> para las funciones wavelet madre Haar, Daub-4 y Daub-8 . . . . .	81
4.1. Características de la jerarquía de memoria . . . . .	111
5.1. Características de la jerarquía de memoria . . . . .	142



# Capítulo 1

## Introducción

---

En este capítulo se describen las principales motivaciones que nos han llevado a la realización de la presente tesis doctoral, así como las principales aportaciones que se pretenden realizar con el desarrollo de este trabajo.

Para situar al lector y comprender dichas motivaciones, en primer lugar, se comentan brevemente las principales materias relacionadas con los temas a desarrollar. El interés por los sistemas basados en multimedia y el desarrollo de la telemedicina ponen de manifiesto la necesidad de la compresión de la información. Las limitaciones de los principales estándares basados en la transformada discreta del coseno (DCT) han originado el desarrollo e investigación de nuevos métodos basados en la transformada discreta de wavelet (DWT). Los métodos de compresión propuestos deben aprovechar las características de las arquitecturas monoprocesador para facilitar su integración y uso en la vida cotidiana, así como su ejecución en tiempo real.

En la parte final del capítulo, y una vez descritas las materias relacionadas comentadas anteriormente, se describen las principales motivaciones y se enumeran las principales aportaciones de la tesis doctoral. Asimismo, se realiza una descripción de la organización y estructura de los diferentes capítulos de la presente memoria.

### 1.1. Aplicaciones multimedia

En los últimos años se ha producido un aumento espectacular del número de sistemas basados en multimedia o aplicaciones multimedia. Las aplicaciones multimedia integran y procesan simultáneamente varios tipos de medios, como vídeo, audio, imágenes, gráficos, texto, etc. El aumento de la capacidad computacional de los ordenadores ha incorporado la información mul-

timedia en muchas disciplinas de distintos ámbitos como las telecomunicaciones, la formación, la comercialización, el entretenimiento o el aspecto lúdico, la medicina, etc.

En la sociedad actual, las telecomunicaciones desempeñan un papel fundamental en nuestra vida diaria. Algunos ejemplos muy representativos pueden ser la generalización del uso del teléfono móvil o el desarrollo del acceso a la red Internet. La aplicación de la tecnología multimedia está permitiendo que la comunicación a distancia se parezca cada vez más a la natural. En los teléfonos móviles de última generación ya es posible transmitir una fotografía o incluso un vídeo. En el futuro más inmediato podremos ver y oír en tiempo real a la persona con la que queramos mantener una conversación. Por su parte, Internet hace una década se encontraba restringida a la comunidad científica. Aún así, era impensable poder difundir documentos que incluyeran audio, vídeo, etc. Hoy en día, cualquier persona desde el ordenador de su casa, puede distribuir documentos que integren gran cantidad de información en varios formatos distintos. Además, Internet se ha convertido en un foco de operaciones comerciales por parte de todo tipo de empresas. El desarrollo de medios de pago a distancia confiables y el perfeccionamiento de catálogos interactivos como medios para atraer a un mayor número de clientes constituyen un par de ejemplos representativos del gran volumen de operaciones comerciales que se producen diariamente.

La educación es otra de las áreas que más se está beneficiando del desarrollo de las aplicaciones multimedia. Los métodos tradicionales de enseñanza están siendo complementados, y en algunos casos sustituidos, por otros cuya efectividad en el aprendizaje es mucho mayor. Un estudio realizado por la *Computer Technology Research Corporation* [Anh y Tegginmath, 2002] demuestra que los humanos retenemos sólo el 20 % de lo que vemos, el 30 % de lo que oímos, el 50 % de lo que vemos y oímos y, de forma espectacular, el 80 % de lo que oímos, vemos y hacemos simultáneamente. Este hecho provoca que la interactividad se convierta en un aspecto fundamental de cualquier sistema multimedia aplicado a la educación. El desarrollo de enciclopedias, manuales de auto-aprendizaje, materiales de apoyo didáctico, bases de datos digitales de cualquier tipo o paseos virtuales para descubrir determinados temas o lugares (museos, países, personajes, parajes naturales) muestran algunos ejemplos de aplicaciones multimedia orientadas a la educación, representativas e introducidas plenamente en nuestro hacer cotidiano.

No nos podemos olvidar de las aplicaciones lúdicas u orientadas hacia el entretenimiento y la diversión. Los videojuegos constituyen un producto con un fuerte peso económico en el mundo de los negocios, cuyo mercado está ampliamente consolidado y en el que se encuentran inmersas las principales compañías de software. La difusión de la televisión digital, el vídeo bajo demanda y el “*home cinema*” representan productos con un potencial creciente y un volumen de negocios

importante. La videoconferencia, por su parte, constituye una aplicación interactiva que se emplea principalmente en la administración de las empresas, pues ahorra costes de desplazamiento y estancias, haciendo más ágil la toma de decisiones.

El rápido desarrollo de las tecnologías de las comunicaciones está acelerando el uso de las aplicaciones de telemedicina en el entorno de la sanidad. La disminución de costes, tiempo y desplazamientos, la formación de nuevos especialistas, la mejora en la precisión diagnóstica o el acercamiento de los recursos sanitarios a zonas despobladas, áreas rurales alejadas de los grandes núcleos urbanos o zonas en situación de conflicto son algunas de las ventajas de la telemedicina. Por todo ello, en la actualidad existe una ingente creación de aplicaciones multimedia interactivas y proyectos de investigación en todo tipo de especialidades médicas, especialmente en aquellas en las que el diagnóstico y la planificación del tratamiento están basadas en el estudio y análisis de imágenes o vídeos médicos procedentes de cualquiera de las diferentes modalidades existentes (radiografía computacional, tomografía axial computacional, resonancia magnética, telediagnóstico, ultrasonidos, etc).

## 1.2. Características de la información médica

La información médica (imágenes, secuencias de vídeo e imágenes volumétricas) consta de propiedades totalmente diferentes a las de cualquier otro tipo de información multimedia [Group, 1997][Marsh, 1998], ya que está en juego la salud o la vida de una persona. Algunas de las principales características se resumen a continuación:

- Privacidad y confidencialidad. El acceso a la información médica debe estar controlada y autorizada. Se suele implantar un acceso restringido por categorías, cuyo objetivo primordial es impedir que los datos sean usados por personal no autorizado.
- Disponibilidad. La información relativa a una persona debe estar disponible en el menor espacio de tiempo posible y en el lugar requerido, ya que puede estar en juego su vida.
- Integridad. Se debe garantizar la integridad de la información almacenada independientemente del medio de almacenamiento y de transferencia utilizado.
- Confiabilidad. El mecanismo de captura de la información de un sistema de telemedicina debe garantizar que la calidad de la información sea equivalente a la original, es decir, la calidad obtenida debe ser comparable a la que se obtiene en condiciones de atención sanitaria convencional.

- Seguridad. Se deben proporcionar una serie de técnicas para salvaguardar los sistemas de información telemáticos. La seguridad debe proteger ambos extremos del sistema de información de alteraciones accidentales o incorrectas durante el intercambio y almacenamiento de la información.
- Duplicación. La creación de copias de seguridad de toda la información recogida se tiene que realizar de forma periódica. Para evitar la pérdida de información en caso de una catástrofe, la conservación y el mantenimiento de las copias se lleva a cabo en un lugar diferente al que ocupan los equipos informáticos. Dicha información deberá ser guardada durante varios años (normalmente las diferentes legislaciones establecen un período entre 5 y 10 años), antes de proceder a su eliminación definitiva.
- Compresión. La información médica recuperada debe corresponderse fielmente a la original. Por tanto, los parámetros de compresión se establecen para asegurar un diagnóstico correcto y sin errores.

Como ya hemos mencionado, la salud o la vida de una persona puede depender del correcto tratamiento de la información médica. Por tanto, existen una serie de leyes y regulaciones propias de cada país para la manipulación de dicha información. En nuestro país, la seguridad de la información médica está resguardada por la Ley Orgánica 15/1999 de Protección de Datos, que abarca a los sistemas de seguridad que tienen que poseer los historiales médicos de los centros de salud. El Real Decreto 994/99, que amplía a la Ley Orgánica, reconoce a la información médica como de máxima sensibilidad. Dicha norma establece que la información concerniente a la salud debe estar especialmente protegida, lo que implica un conocimiento expreso de la creación del archivo por parte de los pacientes y un especial énfasis en la seguridad, privacidad e integridad del mismo. Según este mismo Real Decreto, los centros sanitarios deben poseer medidas de seguridad de nivel alto para los datos de sus pacientes. Esto supone el cifrado de la información de los soportes que salgan del hospital correspondiente y de las redes externas del mismo, para evitar que la información sea inteligible o manipulada por cualquier persona no autorizada. El acceso a la información también se encuentra limitado, pudiéndose organizar Infraestructuras de Clave Pública (PKI), es decir, sistemas de identificación –como tarjetas o claves– tanto dentro como fuera de un hospital o centro de salud.

### 1.3. Necesidades de compresión

En los últimos años, las actividades relacionadas con la administración y la manipulación de la información médica han experimentado un crecimiento espectacular. Dentro de la industria de la salud, dicha actividad constituye uno de los segmentos de mercado más importante y con un mayor volumen de negocio. De hecho, el gasto actual en asistencia sanitaria supera los tres trillones de dólares [PricewaterhouseCoopers, 2003]. La combinación del desarrollo de las redes de telecomunicaciones y de Internet ha provocado el verdadero crecimiento y desarrollo de la telemedicina. Su uso en las pasadas guerras del Golfo Pérsico o Afganistán ha permitido que intervenciones médico-quirúrgicas delicadas hayan podido ser dirigidas por especialistas desde puntos tan alejados como Alemania o Estados Unidos con resultados plenamente satisfactorios. La medicina militar de campaña fue pionera en el uso de la telemedicina, y hoy en día, los ejércitos modernos disponen de estos sistemas como parte básica de su infraestructura militar y sanitaria.

Por todo ello, las predicciones establecen que la manipulación de la información médica que generan los hospitales se convertirá en el segmento de mercado más importante en todo el mundo en el año 2010 [PricewaterhouseCoopers, 2003], en el que el gasto destinado a la telemedicina representará en USA el 15 % del total destinado a la sanidad. Un ejemplo de la importancia de este sector es que en USA se están construyendo una docena de hospitales digitales, en los que todo el trabajo se realizará sin la creación de papel alguno, es decir, toda la extensa tramitación burocrática para la admisión y baja de pacientes, así como el resultado de los diferentes tratamientos a los enfermos se llevará a cabo y se almacenará mediante soporte digital. Sin embargo, la mayoría de hospitales actuales no se pueden convertir en un hospital digital ya que no tienen ni siquiera espacio para el cableado necesario para las redes digitales. La renovación y ampliación de todos estos hospitales se está basando en una estructuración más digital en la que se facilita el acceso a los ordenadores, la creación de redes inalámbricas, la reducción del número de empleados y el proceso burocrático, la ampliación de la flexibilidad, el aumento del ancho de banda de las redes y el incremento del tamaño y la capacidad de los encaminadores y de los soportes de almacenamiento.

La incesante y masiva creación de archivos digitales, imágenes y vídeo médico en los hospitales (con las características peculiares de este tipo de información descritas en la sección 1.2), así como la necesidad de que la transmisión y el almacenamiento de la información multimedia en general, y de la médica en particular, requiera un elevado ancho de banda y un considerable espacio de almacenamiento para cumplir todos los requerimientos legales, ponen de manifiesto la necesidad de las técnicas de compresión para reducir tanto la cantidad de información almacenada como los

datos a transmitir. Por ejemplo, un video en color de una duración de un minuto con una resolución de  $640 \times 480$  pixels y una frecuencia de 24 imágenes/segundo ocupa 1,23 Gbytes sin la aplicación de ninguna técnica de compresión y necesita un ancho de banda de 168,75 Mbits/segundo para poder ser transmitido en tiempo real. Por tanto, la compresión de imágenes, audio y video constituye el pilar básico del desarrollo de las aplicaciones multimedia.

## 1.4. Técnicas de compresión

La compresión de datos se define como un conjunto de técnicas que permiten reducir el número de bits necesario para almacenar o transmitir la información [Nelson y Gailly, 1996]. Por lo tanto, la compresión reduce el ancho de banda necesario para transmitir la información y el espacio necesario para almacenar la información.

Las técnicas de compresión se pueden clasificar desde el punto de vista de la pérdida o no de información. La pérdida de información requiere la introducción de métricas para determinar la calidad visual de la señal reconstruida (error cuadrático medio (MSE)), pico de la relación señal/ruido (PSNR)). Las técnicas de compresión con pérdida de información consiguen unas tasas elevadas de compresión que podrían facilitar el almacenamiento y manipulación de la información en los hospitales. Sin embargo, los especialistas médicos se muestran muy reacios a utilizar dichas técnicas, ya que pueden degradar la calidad de la información reconstruida y complicar la emisión de un diagnóstico final. No debemos olvidar que cuando se realiza un diagnóstico médico puede estar en juego la vida de una persona.

Por tanto, los médicos prefieren utilizar técnicas de compresión en las que no se produce una pérdida de información y se preserva la calidad de los datos iniciales. Sin embargo, estos últimos métodos obtienen tasas de compresión ridículas, comparándolas con las técnicas con pérdida de información. Como ya hemos comentado, los requerimientos legales que imponen los datos médicos, con respecto a su almacenamiento y duplicado de la información, hacen prohibitivo el almacenamiento de la información sin técnicas de compresión eficaces, ya que los soportes de almacenamiento requeridos para cualquier hospital serían muy extensos. Además, el uso de la telediagnosís se complica, si no se dispone de la suficiente compresión, ya que el tráfico en las redes de comunicaciones no cesa de aumentar de forma constante.

Todas estas razones motivan la investigación en técnicas de compresión con pérdida de información y orientadas a la explotación del vídeo médico. Además de las particularidades comentadas en la sección 1.2 sobre la información médica en general, el vídeo médico suele estar codificado en escala de grises, es decir, utiliza un byte para cada pixel, y la cantidad de movimiento entre



*frames* consecutivos suele ser escasa. Por tanto, se pueden aprovechar dichas características para construir técnicas de compresión eficientes.

A lo largo del tiempo se han propuesto muchos y diferentes esquemas de codificación de vídeo. Los más extendidos y que han dado lugar a diversos estándares son los propuestos por el *Grupo de Expertos de Imágenes en Movimiento* (MPEG). La primera generación de estándares para la compresión de imágenes y vídeo (JPEG, JPEG-LS, MPEG-1 y MPEG-2) se construyeron basados en la transformada discreta del coseno. En particular, el estándar MPEG-2 se ocupa de la codificación de vídeo. La aplicación del estándar MPEG-2 a la codificación de vídeo médico no explota de forma adecuada las principales características de este tipo de información. Los principales inconvenientes que presenta son los siguientes:

- El estándar centra sus esfuerzos en la codificación de vídeo en color, representando los colores mediante las componentes YUV, las cuales no son adecuadas para el vídeo codificado en escala de grises.
- La aplicación de la transformada discreta del coseno divide las distintas imágenes que componen la secuencia de vídeo en bloques de  $8 \times 8$  pixels. La división en bloques causa la aparición de artefactos en los bordes, produciendo degradaciones significativas en la reconstrucción del vídeo médico.
- El denominado efecto de Gibbs [Basith y Done, 1996] aparece alrededor de la forma que delimita objetos artificiales (planos de color, textos y figuras geométricas) como una mancha o una nube de polvo. Este efecto viene determinado por el uso de la DCT al digitalizar la información de crominancia y luminancia. Dicho fenómeno también se hace visible en formas naturales como la silueta humana. En el área del fondo que se sitúa alrededor del sujeto aparece un brillo o resplandor que muestra un pequeño movimiento del sujeto. Este brillo o resplandor es lo que se conoce como la aparición de mosquitos en la imágenes o secuencias de vídeo reconstruidas.

Debido a todos estos inconvenientes, en los últimos años, las técnicas basadas en la transformada de wavelet han despertado mucho interés y se han utilizado en diferentes áreas de aplicación como la compresión de huellas dactilares del FBI [Brislaw, 1994], la eliminación de ruido en las señales [Donoho, 1993][Chang *et al.*, 1997][Chang y Vetterli, 1997], la detección de comportamientos auto-similares en series de tiempo [Scargle *et al.*, 1993], la recuperación de imágenes basadas en el contexto [Wang *et al.*, 1997][Brambilla *et al.*, 1999], el diseño de un sistema de seguridad basado en un reconocimiento facial [Martínez, 2002], etc. Por tanto, la DWT surge como alternativa a la DCT para el desarrollo de nuevas técnicas de compresión de imágenes y vídeo. De hecho,

se han propuesto diferentes esquemas de codificación basados en la transformada de wavelet 2D [Lewis y Knowles, 1992][Antonini *et al.*, 1992][Shapiro, 1993] y 3D [Chen y Pearlman, 1996][Kim y Pearlman, 1997][Kim y Pearlman, 2000] para la codificación de imágenes y vídeo. El aumento progresivo del uso de la DWT ha dado lugar a que los nuevos estándares para la codificación de imágenes y vídeo (JPEG-2000, MPEG-4) contengan a la mencionada transformada como opción para realizar la codificación de imágenes y vídeo, respectivamente.

Dado que las técnicas de compresión basadas en la transformada de wavelet 3D ofrecen mejores resultados que las técnicas basadas en 2D para la codificación de vídeo médico, encontramos la necesidad de investigar en técnicas de compresión basadas en la transformada de wavelet 3D que reduzcan drásticamente la cantidad de información a enviar y almacenar, con una calidad suficiente para poder realizar un diagnóstico correcto, una vez que se reconstruya la información.

## 1.5. El entorno de trabajo

Un aspecto crucial en el desarrollo y avance de la telemedicina en general, y de la telediagnos<sup>1</sup> en particular, es la disponibilidad de datos, imágenes y vídeos médicos en tiempo real. En un momento determinado, un médico puede pedir un consejo o un diagnóstico a un colega que se encuentra en otro hospital, el cual, puede estar a miles de kilómetros en cualquier lugar del mundo. Tanto la solicitud de información como la respuesta tienen que producirse de la forma más rápida posible, ya que puede estar en juego la vida de una persona. Por tanto y como ya hemos mencionado, es necesario disponer de la información en el mínimo tiempo posible mediante la codificación y transmisión de la información médica en tiempo real.

Las diferentes especialidades médicas basadas en la telemedicina tienden a utilizar como plataforma de trabajo un ordenador personal (PC de sobremesa o un portátil), ya que suponen una inversión muy baja y permite reducir los costes. Sin embargo, el principal inconveniente que presentan las arquitecturas monoprocesador de propósito general es la posibilidad de ejecutar las diversas y numerosas aplicaciones multimedia mencionadas en la sección 1.1 en tiempo real. En general, la codificación de vídeo médico se presenta como un ejemplo claro de dicha imposibilidad. El doble objetivo de obtener la máxima compresión posible unido a la mayor calidad, provoca que el conjunto de trabajo sea demasiado grande y la aplicación se encuentre limitada por la cantidad de recursos disponibles y, principalmente, por la memoria. Recordemos que una de las limitaciones con mayor importancia en los procesadores actuales viene determinada por el hecho de que la latencia de acceso a la memoria principal es elevada comparada con la velocidad

---

<sup>1</sup>Técnica con una gran popularidad para intercambiar información entre hospitales.

de ejecución del procesador. En un caso particular, las técnicas de codificación de vídeo médico basadas en la transformada de wavelet 3D tienen como principal inconveniente el excesivo tiempo de ejecución en los procesadores actuales al utilizar las tres dimensiones (espacio (x,y) y tiempo). Por tanto, a la hora de realizar la compresión y la descompresión con la citada transformada, el conjunto de trabajo va a ser demasiado grande y el algoritmo wavelet se va encontrar totalmente limitado por la cantidad de memoria disponible. Dicho problema es fácilmente generalizable a otras aplicaciones multimedia, por lo que debe abordarse para mostrar una forma sencilla para permitir la ejecución en tiempo real de aplicaciones multimedia de similares características.

De forma adicional, otro problema común de las aplicaciones multimedia es que el paralelismo existente es difícil de explotar automáticamente con el objetivo de mejorar el rendimiento de las aplicaciones. Por ejemplo, la introducción de las extensiones multimedia en la mayoría de los procesadores actuales de propósito general tiene como objetivo explotar el paralelismo a nivel de datos y ayudar a solventar el problema descrito. La forma más sencilla de incorporar el uso de las extensiones multimedia a las aplicaciones se produce tras un análisis del código fuente por parte del compilador y su incorporación automática al código. En este proceso denominado vectorización automática, el compilador asume toda la responsabilidad e impone ciertas restricciones en el código para detectar y vectorizar las instrucciones correspondientes. También hay que tener en cuenta las dificultades que puede imponer la propia naturaleza del algoritmo a paralelizar. Por tanto, la realización de dicha tarea de una forma eficaz no es sencilla.

En la actualidad, existen compiladores capaces de llevar a cabo una vectorización automática. Sin embargo y debido a las dificultades descritas, es posible que no se obtengan los beneficios esperados. En ese caso, la alternativa que se plantea consiste en llevar a cabo una vectorización manual en la que el programador realiza un estudio del código fuente y del conocimiento del funcionamiento de la aplicación para incorporar de forma manual el uso de las extensiones multimedia y extraer el paralelismo a nivel de datos existente.

Los diseñadores de procesadores están buscando formas para mejorar el rendimiento sin aumentar la complejidad, el consumo de potencia y la disipación de calor. Una de las tendencias actuales es la creación de arquitecturas paralelas de un sólo chip, en las que la disponibilidad de un mayor número de transistores permite una explotación eficiente del paralelismo a nivel de hilos sin la necesidad de disponer de varios chips. Actualmente, existen dos tendencias cuya importancia es cada vez mayor. Por una lado, la posibilidad de incluir varios procesadores en un solo chip (*Chip Multiprocessor* (CMP)) [Hammond *et al.*, 1997][Hammond *et al.*, 2000] permite crear sistemas multiprocesadores de menor coste y disponer de una mayor velocidad de comunicación entre los distintos procesadores. Por otro lado, un único procesador puede realizar la ejecución

simultánea de varios hilos (*Simultaneous Multithreading* (SMT)) [Tullsen *et al.*, 1995][Eggers *et al.*, 1997][Marcuello y González, 1999] añadiendo relativamente poca complejidad al procesador, compartiendo las distintas unidades funcionales y la mayoría de recursos, y duplicando sólo los registros necesarios para mantener el estado de los hilos.

La opción SMT combina los fundamentos de los procesadores superescalares y multi-hilo. Un procesador SMT es capaz de obtener varias instrucciones cada ciclo y mantener simultáneamente el estado de varios hilos. Por tanto, dicho procesador es capaz de emitir varias instrucciones de distintos hilos en cada ciclo, compartiendo los recursos del procesador (unidades de ejecución, caches, predictores de saltos, etc) entre los hilos de forma dinámica y duplicando aquellos recursos necesarios para almacenar el estado de los procesos (registros, contador de programa, pila de direcciones de retorno, etc). De esta forma, se consigue ocultar la latencia de determinados eventos (fallo de cache, fallo de predicción de un salto, interrupciones, etc) de gran duración, sin estar limitada la velocidad por el paralelismo a nivel de instrucción disponible. Por tanto, un procesador SMT mejora la productividad total del sistema cuando ejecuta varios procesos independientes, aunque el tiempo de ejecución de una aplicación secuencial puede aumentar. Dicha tecnología se está utilizando cada vez más en arquitecturas monoprocesador de propósito general como el PC. La compañía Intel ha introducido una implementación de SMT denominada *Hyper-Threading* (HT) [Marr *et al.*, 2002] en sus nuevos procesadores *Pentium IV Xeon* obteniendo mejoras en el tiempo de ejecución de cerca del 30 % en algunos casos [Chen *et al.*, 2002][Magro *et al.*, 2002].

El aprovechamiento eficaz de las arquitecturas descritas supone el uso de esquemas basados en varios hilos o procesos. Por tanto, hay que replantearse la implementación de muchos algoritmos para poder aprovechar las prestaciones de los nuevos procesadores y construir programas paralelos o paralelizar programas secuenciales ya realizados. La forma más sencilla de obtener programas paralelos a partir de secuenciales se consigue a través de la paralelización automática, en la que se produce la intervención del compilador para examinar el código fuente y extraer el paralelismo existente en la aplicación correspondiente. Dicho proceso es atractivo y prometedor para algunas aplicaciones, pero es incapaz de obtener buenos resultados en muchas otras. En ese momento se plantea una segunda opción, que consiste en la paralelización manual, mediante el análisis del código fuente por parte del programador para paralelizar la aplicación directamente o transformarla de forma que se pueda aplicar la paralelización automática.

Los métodos de paralelización automática no suelen obtener buenos resultados para ciertas aplicaciones multimedia, como es el caso de la codificación de vídeo médico basada en la transformada de wavelet 3D. Por tanto, es necesario llevar a cabo una paralelización manual para obtener buenos resultados y aprovechar eficazmente la tecnología SMT. La aparición de los procesado-

res SMT [Marr *et al.*, 2002] permite el desarrollo de propuestas basadas en una implementación multi-hilo en el que se aproveche las ventajas de dichos procesadores. Existen varias formas de descomponer la aplicación original. En una descomposición basada en el dominio de los datos, cada hilo realiza de forma simultánea las mismas tareas sobre una parte de los datos. Por el contrario, en una descomposición funcional cada hilo procesa una tarea diferente sobre el mismo conjunto de datos simultáneamente. La descomposición funcional, que requiere un conocimiento del funcionamiento de la aplicación, va a obtener mejores resultados que una descomposición por datos. Sin embargo, los compiladores encuentran mayores dificultades al aplicar una descomposición funcional que en el caso de una descomposición basada en el dominio de los datos para realizar una paralelización automática eficaz.

## 1.6. Motivación y aportaciones

A partir de los antecedentes descritos, en esta sección presentaremos las principales motivaciones que nos han llevado a desarrollar la presente tesis doctoral y describiremos las principales aportaciones conseguidas.

### 1.6.1. Motivación

Como ya se ha indicado anteriormente, en los últimos años se ha producido un aumento espectacular del número de aplicaciones multimedia que integran vídeo, audio, imágenes, texto, etc. Este hecho, unido al aumento del volumen de imágenes y vídeo en los hospitales y al desarrollo de todo tipo de especialidades médicas y proyectos de investigación basados en la telemedicina, especialmente aquellas que se basan en el estudio y análisis de imágenes o vídeos médicos, así como las características particulares y peculiares de la información médica para su almacenamiento y transmisión, ponen de manifiesto la necesidad de las técnicas de compresión de la información médica.

El desarrollo de técnicas de compresión para el tratamiento de la información médica debe realizarse para que se obtenga la máxima compresión posible sin que existan degradaciones en la calidad de las imágenes o vídeos reconstruidos, ya que la emisión de un diagnóstico o tratamiento erróneo puede acabar con la vida de una persona. Los codificadores basados en una transformada matemática han dado lugar a los principales estándares de compresión de imágenes y vídeo (JPEG, JPEG-LS, MPEG-1, MPEG-2) basados en la transformada discreta del coseno. Sin embargo, en el caso de la codificación de vídeo médico, dicha transformada no aprovecha

las particularidades propias de este tipo de información (codificación en escala de grises, poco movimiento) y presenta graves inconvenientes que dificultan la labor médica. La transformada de wavelet surge como alternativa a la DCT, dando lugar a que los nuevos estándares (JPEG-2000, MPEG-4) de codificación de imágenes y vídeo utilicen dicha transformada.

Por tanto, el desarrollo de técnicas de compresión basadas en la transformada de wavelet 3D para la codificación de vídeo médico pretende reducir de forma drástica la cantidad de información a transmitir y almacenar, y disponer de una calidad suficiente y con garantías para realizar un diagnóstico correcto por parte de la comunidad médica.

Como ha quedado reflejado, una de las propiedades más importantes para el desarrollo de la telemedicina consiste en la disponibilidad de la información en tiempo real. De hecho es una de las limitaciones más importantes para la ejecución de aplicaciones multimedia en arquitecturas monoprocesador de propósito general. En el caso de la codificación de vídeo médico basada en la transformada de wavelet 3D, se pueden desarrollar propuestas que aprovechen las características de las arquitecturas de propósito general con un único procesador y permitan la codificación de vídeo médico en tiempo real. La realización de un primer paso hacia la convergencia entre las aplicaciones multimedia y las extensiones multimedia, disponibles en la mayoría de los procesadores de propósito general, para explotar el paralelismo a nivel de datos y reducir el tiempo de ejecución de las aplicaciones es un campo de investigación totalmente abierto.

El uso de la vectorización automática o manual para extraer el paralelismo de las aplicaciones, las técnicas de división en bloques, el desenrollado de bucles, la pre-búsqueda de datos o el reuso de operaciones en punto flotante entre otras, constituyen un abanico de posibles técnicas para llevar a cabo en las aplicaciones y conseguir la ejecución en tiempo real de las mismas.

Las nuevas técnicas en el diseño de procesadores para mejorar el rendimiento sin aumentar el número de transistores, el consumo de potencia y la disipación de calor ha originado la aparición de nuevas técnicas hardware para explotar el paralelismo como la ejecución de varios hilos simultáneamente (SMT) en un sólo procesador. Dicha técnica requiere replantearse la implementación de muchos algoritmos para aprovechar eficazmente dichas arquitecturas mediante una paralelización automática o manual. Si analizamos una aplicación concreta como el codificador 3D-FWT, se pueden realizar varias propuestas para una ejecución multi-hilo que mejoren el rendimiento del procesador y que se puedan generalizar a aplicaciones multimedia similares.

### 1.6.2. Aportaciones de la tesis

Las principales aportaciones de la presente tesis doctoral van a estar relacionadas con la propuesta, evaluación y optimización de un codificador de vídeo médico basado en el uso de la transformada de wavelet 3D, y ejecutado en arquitecturas monoprocesador. Todas las propuestas desarrolladas para tal fin deben poseer una sencilla generalización a aplicaciones multimedia de características similares. Las aportaciones más destacadas son las siguientes:

1. Se realiza una descripción detallada de los fundamentos y las distintas opciones para las principales fases o procesos que componen un codificador basado en una transformada matemática. Haremos especial hincapié en el funcionamiento de la transformada de wavelet, en las características principales de los estándares de compresión basados en la DCT y en la DWT, así como en las aplicaciones más habituales de la transformada de wavelet.
2. Se propone un nuevo codificador de vídeo médico con pérdidas basado en la transformada de wavelet 3D. Sus principales objetivos son: obtener una tasa de compresión alta, conseguir una calidad excelente desde el punto de vista médico, y mejorar las técnicas existentes. Para tal fin se realizan varias propuestas en cada una de las fases del codificador.
3. Se realizan varias optimizaciones sobre el codificador de vídeo médico original para superar sus limitaciones y aumentar la tasa de compresión sin alterar la calidad de las secuencias de vídeo reconstruidas.
4. Se reduce el tiempo de ejecución del codificador haciendo visibles al algoritmo algunas de las principales características de la arquitectura donde se ejecuta: jerarquía de memoria y extensiones multimedia. Asimismo, se implementará el reuso de operaciones en punto flotante. Todo esto supone un primer paso hacia la automatización del proceso de convergencia entre las aplicaciones multimedia y los procesadores de propósito general que disponen de extensiones multimedia.
5. Se llevan a cabo varias propuestas basadas en un esquema multi-hilo para el codificador de vídeo médico 3D-FWT y el procesador *Pentium IV Xeon* HT (SMT), que se diferenciarán en la forma de descomponer la aplicación original, con el objetivo de reducir el tiempo de ejecución y, por tanto, aprovechar eficazmente las características de las nuevas arquitecturas paralelas en un único chip. Las propuestas basadas en una descomposición funcional de las tareas a realizar por una aplicación obtendrán los mejores resultados aunque requerirán un mayor conocimiento del comportamiento de dicha aplicación.

## 1.7. Organización de la tesis

Para llevar a cabo las aportaciones descritas en la sección anterior, la presente memoria se ha estructurado en los siguientes capítulos:

- Capítulo 1. Se introducen, de forma resumida, las principales materias relacionadas con la presente tesis doctoral para motivar y definir las diferentes aportaciones a llevar a cabo.
- Capítulo 2. Se presenta una descripción detallada de los fundamentos y las distintas opciones para las principales fases o procesos de un codificador basado en una transformada matemática. Los procesos descritos son: la transformada matemática de los datos, la cuantificación y la codificación entrópica. Además, se introducen los principales estándares de compresión de imágenes y vídeo basados en la DCT y en la DWT, así como un resumen de las principales aplicaciones de la transformada de wavelet.
- Capítulo 3. Se propone un nuevo codificador de vídeo médico con pérdida de información, basado en la transformada de wavelet 3D (3D-FWT). Se llevan a cabo varias propuestas y se analizan y evalúan distintas opciones en cada una de las fases del codificador – transformada de wavelet 3D-FWT, umbralización, cuantificación y codificación entrópica – con el objetivo de obtener una gran tasa de compresión, conseguir una calidad excelente desde el punto de vista médico y mejorar las técnicas existentes.
- Capítulo 4: Se proponen varias optimizaciones aplicadas al cálculo de la transformada de wavelet 3D: técnicas de división en bloques (*blocking*), reuso de operaciones en punto flotante, uso de extensiones multimedia, desenrollado de bucles, pre-búsqueda de datos y vectorización por columnas, para permitir su ejecución en tiempo real en arquitecturas monoprocesador.
- Capítulo 5: Se realiza un análisis e implementación de una aplicación multimedia (codificador 3D-FWT, desarrollado en los dos capítulos anteriores), en un procesador con tecnología SMT. Se realizan varias propuestas para la ejecución multi-hilo del codificador 3D-FWT, que se diferencian en la forma de descomponer la aplicación original. Por tanto, se propone una descomposición basada en el dominio de los datos o una descomposición basada en una división funcional de las tareas de la aplicación multimedia.
- Capítulo 6: Se resumen las principales conclusiones, a partir de las diferentes propuestas realizadas en cada uno de los capítulos, y se describen las principales orientaciones para continuar con el trabajo desarrollado en la presente tesis doctoral.



## Capítulo 2

### Compresión de imágenes y secuencias de vídeo

---

La compresión de datos se define como un conjunto de técnicas que permiten reducir el número de bits necesario para almacenar o transmitir la información [Nelson y Gailly, 1996]. Por lo tanto, la compresión reduce el ancho de banda necesario para transmitir la información y el espacio necesario para almacenar la información.

Para medir cuánto comprime un algoritmo de compresión, se define *la tasa, el ratio, la proporción o la relación de compresión* como la proporción del tamaño (número de bytes) del conjunto original y el conjunto de datos comprimido, como se puede observar en la ecuación 2.1.

$$\text{Tasa de compresión} = \frac{\text{Tamaño original}}{\text{Tamaño comprimido}} \quad (2.1)$$

La tasa de compresión puede expresarse por un número o por dos. Por ejemplo, 30 : 1 significa que 30 bytes de los datos originales se representan por un solo byte en los datos comprimidos. Un mayor ratio de compresión determina una mayor cantidad de datos comprimida.

Otra forma de medir la compresión alcanzada por un determinado algoritmo es mediante *la tasa o factor de bit, especificado en bits/pixel o “bpp”* que se define como el número medio de bits requeridos para representar el valor de los datos para un pixel de una imagen o una secuencia de vídeo. Por ejemplo, en una imagen de  $n \times m$  pixels en la que se usan  $B$  bits para almacenar la imagen comprimida, la *tasa de bit* es:

$$\frac{\text{Bits}}{\text{pixel}} = \frac{B}{n \times m} \quad (2.2)$$

La tasa de bits se especifica en bits por pixel (bpp). Por tanto, cuanto menor sea la tasa de

bits, mayor será la compresión alcanzada por una técnica de compresión. Por ejemplo, si tenemos una imagen original de 8 bits por pixel y el algoritmo de compresión tiene una tasa de 1 bpp, entonces el factor de compresión es 8 : 1.

Las técnicas de compresión se pueden clasificar desde diferentes puntos de vista. Con respecto a la pérdida o no de información tenemos:

- **Compresión sin pérdida de información.** La imagen o el video reconstruido, después de la compresión, es idéntico al original, por lo que no se produce pérdida de información. Estas técnicas suelen alcanzar tasas de compresión pequeñas. Algunos ejemplos de codificadores sin pérdida son los codificadores de Huffman, aritméticos, Shannon-Fano, etc.
- **Compresión con pérdida de información.** La imagen o el video reconstruido contiene distorsión o degradación con respecto al original. Se pueden conseguir grandes tasas de compresión dependiendo de la mayor o menor pérdida de información. Con estas técnicas se trata de encontrar un compromiso entre la tasa de compresión y una calidad visual percibida en la que no exista una pérdida significativa. Algunos codificadores con pérdida de información representativos son el método de modulación por código diferencial de pulsos (DPCM), codificadores basados en una transformada como la transformada discreta del coseno o la transformada de wavelet, etc.

En las técnicas de compresión con pérdida de información es necesario medir la calidad visual de la imagen o la secuencia de vídeo reconstruida. Esta medida indica el nivel de similitud de la señal reconstruida con respecto a la imagen o el vídeo original. Los índices de calidad visual se pueden clasificar en objetivos y subjetivos.

La evaluación subjetiva se refiere a la percepción por parte del sistema visual humano de la buena o mala reconstrucción de una imagen o una secuencia de vídeo. Esta evaluación suele estar basada en la realización de una serie de tests visuales en los que se valora con un índice numérico la calidad de la señal. Dichos cuestionarios deben ser llevados a cabo por expertos en el tipo de imágenes o secuencias de vídeo. Por ejemplo, un médico es una persona cualificada para la evaluación de imágenes o secuencias de vídeo médicas.

El error cuadrático medio (MSE) y el pico de la relación señal/ruido (PSNR) son las medidas más utilizadas para medir de forma objetiva la calidad visual de las imágenes o secuencias de vídeo reconstruidas. Dada una imagen original,  $f(x, y)$ , y la correspondiente reconstruida,  $g(x, y)$ , ambas de  $n \times m$  pixels, el MSE se define en la ecuación 2.3 y se interpreta como la media del cuadrado de la diferencia entre los pixels de las dos imágenes.

$$MSE = \frac{1}{n \cdot m} \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} (g(x, y) - f(x, y))^2 \quad (2.3)$$

En cuanto al PSNR, su cálculo se puede observar en la ecuación 2.4.

$$PSNR = 10 \log_{10} \frac{f_{max}^2}{MSE} \quad (2.4)$$

dónde  $f_{max}$  es el valor máximo de un pixel (255 para imágenes de 8 bits). El PSNR se mide en decibelios, dB, de forma que un mayor valor indica una mejor calidad de la señal reconstruida. Normalmente, se considera que un valor superior a 30 dB para una imagen indica una calidad muy buena.

Para una secuencia de vídeo se pueden calcular el MSE y el PSNR como la media aritmética del MSE o del PSNR de cada una de las imágenes reconstruidas con respecto a la correspondiente imagen original.

Por otro lado, las técnicas de compresión también se pueden clasificar con respecto a la transformación de los datos para obtener la compresión final:

- **Codificación basada en predicción.** Este esquema intenta predecir el siguiente valor en la cadena de datos, almacenando la diferencia entre la predicción y el valor actual, por lo que se obtienen valores más pequeños en el dominio del espacio. Se trata de técnicas fáciles de implementar y adaptables a las características locales de la imagen. El método de modulación por código diferencial de pulsos es el ejemplo más representativo de codificación predictiva.
- **Codificación basada en transformada.** Los métodos basados en transformada, transforman la imagen o el video del dominio del espacio a otra forma de representación utilizando una transformada como la de Fourier, coseno, wavelet, etc. La transformación tiene como objetivo decorrelar la información para aplicar eficazmente otras técnicas de compresión. Obtienen mayores tasas de compresión que las técnicas basadas en predicción aunque tienen una mayor complejidad de computación.

Si nos centramos en la codificación basada en transformada, normalmente, además de la transformación lineal de los datos, es necesario combinar la transformación de los datos con otros métodos de compresión para obtener los resultados esperados. La figura 2.1 presenta el esquema típico de un codificador de imágenes basado en una transformada matemática.

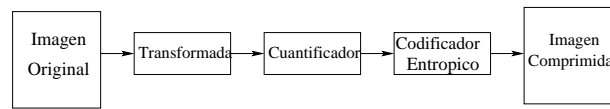


Figura 2.1: Diagrama de bloques de un codificador basado en una transformada matemática

Los trabajos desarrollados por los grupos de expertos de imágenes estáticas JPEG (*Joint Photographic Experts Group*) e imágenes en movimiento MPEG (*Moving Picture Experts Group*) constituyen, respectivamente, los ejemplos más representativos del desarrollo de algoritmos para la compresión de imágenes, audio y vídeo. Estos grupos de trabajo han dado lugar a la creación de una serie de estándares basados en una transformada matemática para la compresión de imágenes (JPEG, JPEG-LS, JPEG-2000) y vídeo (MPEG-1, MPEG-2 y MPEG-4) que describiremos posteriormente. La mayoría de dichos estándares se basan en la aplicación de la transformada discreta del coseno (DCT). Sin embargo, la transformada de wavelet (DWT) [Mallat, 1989] aparece como una alternativa a la DCT para la compresión de imágenes y vídeo. La transformada DWT está recibiendo mucha atención en el campo del procesamiento de imágenes debido a su flexibilidad en la representación de señales no estacionarias y a su adaptación a las características del sistema de visión humano. Su aplicación realiza un análisis de multiresolución y una descomposición sub-banda, por lo cual su utilización en los últimos diez años está siendo fundamental en el procesamiento digital de imágenes y vídeo. De hecho, los estándares JPEG-2000 y MPEG-4 de compresión de imágenes y vídeo se basan o posibilitan, respectivamente, la utilización de la DWT.

En este capítulo, describiremos las principales fases – transformada matemática, cuantificación y codificación entrópica – que componen un codificador basado en una transformada matemática. En concreto, dentro del proceso de transformación definiremos la DCT a partir de la transformada de Fourier, y la transformada de wavelet como una alternativa a la de Fourier. Se introducen los algoritmos para el cálculo de la transformada de wavelet unidimensional, bidimensional y tridimensional. La descripción del proceso de cuantificación viene acompañada de un análisis de las principales formas de cuantificación. En cuanto a la codificación entrópica, se define la codificación Huffman a partir de la codificación de Shanon-Fano y el funcionamiento de la codificación aritmética y la codificación *Run-Length*. A continuación se realiza una descripción del funcionamiento de los principales estándares de codificación basados en una transformada matemática, así como una enumeración de los principales métodos de compresión basados en la transformada de wavelet bidimensional y tridimensional para la compresión de imágenes y vídeo, respectivamente. El capítulo concluye con una exposición de las principales aplicaciones de la transformada

de wavelet.

## 2.1. La transformada matemática

La aplicación de una transformada matemática supone el cambio de una cantidad (un número, un vector, una función, etc) a una forma menos conocida en la que se pueden mostrar propiedades particularmente útiles. El principal uso de la transformada consiste en la resolución de un problema o cálculo, a cuyo resultado se le aplica la transformada inversa correspondiente para obtener la forma original.

Un ejemplo sencillo e ilustrativo lo constituye la multiplicación de números Romanos. Cuando se quieren multiplicar dos números Romanos, normalmente, transformamos ambos números en notación decimal, multiplicamos, y realizamos la transformada inversa para obtener el producto final en notación Romana, como se muestra en la ecuación 2.5.

$$XCVI \times XII = 96 \times 12 = 1152 = MCLII \quad (2.5)$$

Las funciones suelen tener al tiempo como parámetro. Se dice que una función  $g(t)$  se encuentra representada en el dominio del tiempo. Dado que una función típica oscila, se puede representar mediante una onda o una combinación de ondas. Una vez realizado este paso, la función resultante  $G(t)$  se encuentra representada en el dominio de la frecuencia, en el cual se pueden llevar a cabo una serie de operaciones sobre las funciones de forma sencilla.

En esta sección vamos a describir tres transformadas matemáticas: la transformada de Fourier, la transformada discreta del coseno y la transformada de wavelet. Las dos últimas se han utilizado ampliamente en la compresión de imágenes y secuencias de vídeo.

### 2.1.1. La transformada de Fourier

La transformada de Fourier de una señal proporciona una descripción de la distribución de la energía de la señal con respecto a la frecuencia, es decir, su espectro. La representación de la señal en el dominio de Fourier es única, y consiste en una suma continua de sinusoides de diferentes amplitudes, frecuencias y fases.

La transformada de Fourier transforma una función en el dominio del tiempo al dominio de la frecuencia, para analizar el espectro de frecuencias de la señal, ya que los coeficientes de Fourier representan la contribución de cada una de las funciones seno y coseno a cada una de las

frecuencias. La transformada inversa de Fourier, como era de esperar, transforma una función en el dominio de la frecuencia al dominio del tiempo.

La transformada discreta de Fourier (DFT) realiza una estimación de la transformada de Fourier mediante el muestreo de un número determinado de puntos de la función. Las distintas muestras caracterizan la señal en el dominio del tiempo, representando los valores transformados la media, sobre todo el tiempo, del contenido de energía de la señal a una cierta frecuencia. La DFT cumple la propiedad simétrica, de la misma forma que la transformada continua, por lo que se puede obtener, fácilmente, la transformada inversa discreta de Fourier.

Sin embargo, si la señal a representar no es estacionaria, la potencia y/o el espectro de la señal pueden variar con el tiempo. Por ejemplo, en el caso de señales transitorias con una duración finita, el proceso de promediado puede dar lugar a que se genere un espectro complicado en el que se oscurezca el hecho de que la señal sólo ha estado presente durante un intervalo de tiempo finito. Por lo tanto, la transformada de Fourier no es adecuada para el análisis de señales no estacionarias.

La transformada de Fourier localizada (WFT, *Windowed Fourier Transform*) puede ser una solución para la representación de señales no estacionarias, ya que realiza un análisis simultáneo de la señal en el dominio del tiempo y de la frecuencia. Para ello, se divide la señal en distintas secciones o ventanas y se realiza el análisis de Fourier sobre un intervalo temporal pequeño centrado en un instante de tiempo determinado. La WFT también puede ser considerada como la implementación de un banco de filtros. A una cierta frecuencia, se realiza el filtrado de la señal a lo largo de todo el tiempo.

#### 2.1.1.1. La transformada discreta del coseno

La transformada del coseno unidimensional, puede expresarse como la aplicación de la transformada discreta de Fourier a una secuencia finita de  $n$  elementos. Por tanto, se trata de una variación de la transformada de la DFT donde la señal se descompone en sumas de cosenos (y no de senos y cosenos como en la de Fourier). La DCT produce una compactación muy buena de la energía de los datos que se encuentran altamente correlacionados.

#### 2.1.2. La transformada de wavelet

La transformada de wavelet continua (*Continuous Wavelet Transform* (CWT)) es una alternativa a la transformada de Fourier localizada (WFT). La CWT combina la aplicabilidad al

análisis tiempo-frecuencia propia de la WFT, con las propiedades matemáticas más interesantes de la transformada de Fourier, como la ortogonalidad y la unicidad. La transformada de wavelet proyecta una función del tiempo (la señal) en una función de  $a$  y  $b$ . El parámetro  $a$  se denomina escala, mientras que el parámetro  $b$  es la translación. Suponemos que la señal  $x(t)$  es tal que  $x(t) \in L^2(R)$ , y por lo tanto

$$X_{CWT}(a, b) = \frac{1}{\sqrt{a}} \int x(t) \psi\left(\frac{t-b}{a}\right) dt \quad (2.6)$$

donde  $\psi(t)$  es la función wavelet madre y las funciones base  $\psi(\frac{t-b}{a})/\sqrt{a}$ , resultado de las dilataciones y translaciones que se realizan sobre  $\psi(t)$ , constituyen la familia de wavelet.

La transformación puede interpretarse también como la implementación de un banco de filtros no uniforme (y continuo en el tiempo), en el que se mantiene una relación constante entre la resolución en frecuencia en el tiempo dependiendo de  $a$ . Valores grandes de  $a$  dan lugar a una mayor resolución espectral, mientras que los valores pequeños proporcionan una mejor resolución temporal.

Puede demostrarse que la CWT, al contrario que la WFT, permite una reconstrucción perfecta de la señal, sin ninguna ambigüedad, a partir de los coeficientes de la transformada [Daubechies, 1992].

En la ecuación 2.6, tanto  $a$  como  $b$  son variables continuas, con lo que la representación de  $x(t)$  con la CWT resulta redundante. Además, en la práctica, para la CWT sólo se puede utilizar un número finito de valores de  $(a, b)$ . Un algoritmo discreto para aproximar la CWT se obtiene muestreando  $a$  y  $b$ , de tal forma que  $a = a_0^j > 0$  y  $b = a_0^j k b_0$ , donde  $a_0$  y  $b_0$  son los intervalos de muestreo y  $j, k \in \mathbb{Z}$ . Normalmente, se utilizan los valores 2 y 1 para  $a$  y  $b$ , respectivamente. Sustituyendo estos valores de  $a$  y  $b$  en la ecuación 2.6 se obtiene la transformada de wavelet discreta (DWT):

$$X_{DWT}(j, k) = \int x(t) \psi_{j,k}(t) dt \quad (2.7)$$

donde  $\psi_{j,k} = 2^{-j/2} \psi(2^{-j}t - kb_0)$ ,  $\psi_{0,0}(t) = \psi(t)$ . Obsérvese que tanto  $x(t)$  como  $\psi(t)$  siguen siendo funciones continuas en el tiempo. Los valores de  $a$  y  $b$  se eligen de tal forma que el conjunto de funciones wavelet forman un conjunto ortogonal, es decir, el producto interno entre las diferentes funciones wavelet  $\psi_{j,k}$  es igual a cero.

El análisis de los datos a diferentes escalas o resoluciones es la principal diferencia entre la DWT y la WFT, y constituye el análisis de multi-resolución (MRA), introducido por Mallat

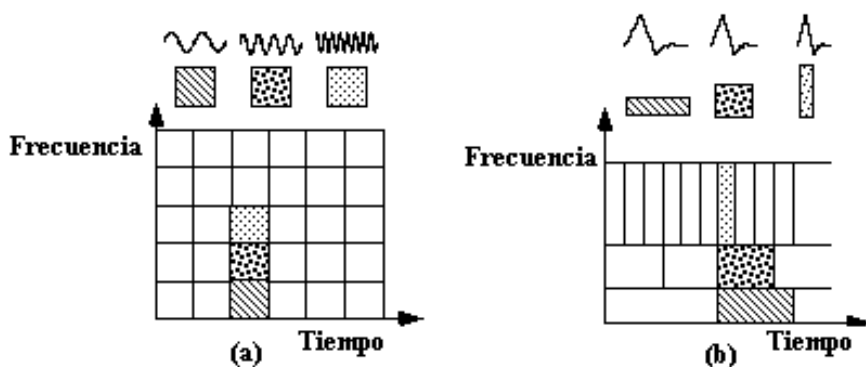


Figura 2.2: a) Funciones básicas de Fourier, ventanas tiempo-frecuencia y representación en el plano tiempo-frecuencia. b) Funciones básicas wavelet (Daubechies), ventanas tiempo-frecuencia y representación en el plano tiempo-frecuencia

[1989]. En la WFT, se produce la imposibilidad de mejorar simultáneamente las resoluciones temporal y en frecuencia. Si se quiere alcanzar una cierta resolución en el tiempo o la frecuencia, hay que utilizar diferentes anchos de la ventana. Por otro lado, la mayor anchura posible de la ventana viene muy a menudo impuesta por el mayor intervalo durante el cual puede suponerse que la señal es estacionaria.

Una forma de establecer las diferencias entre la DWT y la WFT es observar cómo la función base se representa en el plano tiempo-frecuencia. La figura 2.2 a) representa la WFT en el plano tiempo-frecuencia, en la que la ventana es simplemente una onda cuadrada. Dicha ventana necesita truncar la función seno o coseno correspondiente para poder representarla en una ventana de una determinada anchura, ya que la WFT utiliza una sola ventana para todas las frecuencias y la resolución del análisis es la misma para todas las posiciones en el plano tiempo-frecuencia.

En cuanto a la transformada de wavelet, la ventana puede variar de tamaño, por lo que la representación de señales no estacionarias, se puede llevar a cabo mediante varias funciones de corta duración para altas frecuencias (mejor resolución en el tiempo), y funciones de larga duración para bajas frecuencias (mejor resolución en la frecuencia), es decir, la transformada wavelet. La figura 2.2 b) representa la zona cubierta por una función wavelet (Daubechies) en el plano tiempo-frecuencia.

Debemos recordar que la transformada de wavelet tiene un conjunto infinito de funciones básicas posibles, al contrario que la transformada de Fourier que solo puede utilizar las funciones seno y coseno. Por todo ello, el análisis de la transformada de wavelet proporciona un acceso inmediato a la información que puede quedar oculta por otros métodos tiempo-frecuencia como



la transformada de Fourier.

El MRA analiza la señal en distintas frecuencias y con diferentes (tiempo) resoluciones. Se diseña para conseguir una adecuada resolución temporal para frecuencias muy altas y una buena resolución espectral para frecuencias muy pequeñas. Mallat explica como obtener la información a través de varias iteraciones y define la transformada de wavelet como una transformación lineal de una señal a una base que descansa entre los dominios del espacio y de la frecuencia, donde no existe información redundante ya que las funciones wavelet son ortogonales. De esta forma, en un MRA existen dos funciones: la función wavelet madre  $\psi$  y la función de escalado asociada  $\phi$ . Por lo tanto, la transformada de wavelet se puede implementar mediante filtros de reflejo cuadráticos (QMF):  $G = g(n)$  y  $H = h(n)$   $n \in \mathbb{Z}$ , donde  $h(n) = \frac{1}{2} < \phi(\frac{x}{2}), \phi(x - n) >$  y  $g(n) = (-1)^n h(1 - n)$  ( $< >$  especifica el espacio de todas las funciones integrables al cuadrado  $L^2$ ).  $H$  se corresponde con un filtro pasa-baja, mientras que  $G$  es un filtro pasa-alta. Los filtros de reconstrucción se basan en una respuesta a impulsos finitos, donde  $H^* = h^*(n) = h(1 - n)$ , y  $G^* = g^*(n) = g(1 - n)$ . Para un mayor análisis de la relación entre las funciones wavelet y los filtros QMF, se puede analizar el artículo de Mallat [1989].

Una vez introducida la transformada discreta de wavelet, vamos a ver como se aplica la transformada sobre una señal unidimensional (1D), una imagen bidimensional (2D) y un vídeo tridimensional (3D) [Chui, 1992][Meyer, 1993][Vidakovic y Müller, 1994] [Graps, 1995][Stollnitz *et al.*, 1995a][Stollnitz *et al.*, 1995b].

### 2.1.2.1. La transformada de wavelet 1D

La aplicación de los filtros QMF, mencionados en el apartado anterior se corresponden con una iteración de la transformada de wavelet. Supongamos una señal discreta ( $s$ ) de una longitud de  $2^n$ . La aplicación de la transformada de wavelet mediante dos filtros determinados,  $H$  y  $G$ , divide la señal original en dos mitades: la sub-banda pasa-baja o señal de referencia y la sub-banda pasa-alta o detalle de la señal. El proceso se puede repetir, aplicando sobre la sub-banda pasa-baja los filtros  $H$  y  $G$ , para obtener dos nuevas sub-bandas pasa-baja y pasa-alta, es decir, el siguiente nivel de descomposición de la transformada, y así sucesivamente para el resto de iteraciones. Este proceso se denomina transformada de wavelet 1D (1D-WT).

Es importante señalar que el proceso de descomposición de la transformada de wavelet genera un conjunto de muestras discretas, cuyo número es exactamente el mismo que el de la señal original, gracias a la ortogonalidad de las funciones wavelet.

Dado un vector de elementos  $\{c_n^J\}_{n=0,1,\dots,N-1}$ , hemos implementado la 1D-WT mediante el

cálculo de la transformada de wavelet rápida 1D (1D-FWT), que obtiene las señales de referencia  $c_n^{j-1}$  y detalle  $d_n^{j-1}$  mediante las siguientes expresiones:

$$c_n^{j-1} = \sum_{l=0}^{D-1} a_l c_{<l+2n>_{2^j}}^j \quad (2.8)$$

$$d_n^{j-1} = \sum_{l=0}^{D-1} b_l c_{<l+2n>_{2^j}}^j \quad (2.9)$$

donde  $n = 0, 1, \dots, 2^{j-1} - 1$  y  $j = J, J-1, \dots, J-\lambda+1$ . La expresión  $<l+2n>_{2^j}$  representa una función módulo de tal forma que  $<l+2n>_{2^j} \in [0, 2^j - 1]$  para todos los valores de  $l \in \mathbb{Z}$ .  $\lambda$  es un entero entre 0 y  $J$  que determina el número de iteraciones de la transformada, mientras que los números  $a_l$  y  $b_l$  para  $l = 0, 1, \dots, D-1$  son los denominados coeficientes de filtro wavelet. Por tanto, y como se ha mencionado anteriormente, cada iteración de la transformada de wavelet divide el vector de entrada en dos partes (referencia y detalle), tal y como se ilustra en la ecuación 2.10. Dicha ecuación muestra el resultado de cuatro iteraciones de la 1D-FWT sobre una secuencia de 16 elementos de entrada en la que  $\lambda = J = 4$ . Denotar que el cálculo de cada uno de los elementos  $d_n^{j-1}$  que forman parte del detalle de la señal no se vuelve a calcular, mientras que los elementos  $c_n^{j-1}$  que componen la señal de referencia se calculan en cada una de las iteraciones a partir del resultado de la iteración anterior.

$$\begin{aligned}
 & c_0^4 c_1^4 c_2^4 c_3^4 c_4^4 c_5^4 c_6^4 c_7^4 c_8^4 c_9^4 c_{10}^4 c_{11}^4 c_{12}^4 c_{13}^4 c_{14}^4 c_{15}^4 \\
 & \quad \downarrow \\
 & c_0^3 c_1^3 c_2^3 c_3^3 c_4^3 c_5^3 c_6^3 c_7^3 d_0^3 d_1^3 d_2^3 d_3^3 d_4^3 d_5^3 d_6^3 d_7^3 \\
 & \quad \downarrow \\
 & c_0^2 c_1^2 c_2^2 c_3^2 d_0^2 d_1^2 d_2^2 d_3^2 d_0^3 d_1^3 d_2^3 d_3^3 d_4^3 d_5^3 d_6^3 d_7^3 \\
 & \quad \downarrow \\
 & c_0^1 c_1^1 d_0^1 d_1^1 d_0^2 d_1^2 d_2^2 d_3^2 d_0^3 d_1^3 d_2^3 d_3^3 d_4^3 d_5^3 d_6^3 d_7^3 \\
 & \quad \downarrow \\
 & c_0^0 d_0^0 d_0^1 d_1^1 d_0^2 d_1^2 d_2^2 d_3^2 d_0^3 d_1^3 d_2^3 d_3^3 d_4^3 d_5^3 d_6^3 d_7^3
 \end{aligned} \quad (2.10)$$

La transformada de wavelet inversa para una señal unidimensional se puede obtener de la misma forma que la transformada directa, aplicando el proceso anterior de forma inversa. Para

ello, dada el último nivel de descomposición de la transformada de wavelet de una señal unidimensional, se aplican los filtros  $H^*$  y  $G^*$  para obtener el penúltimo nivel de descomposición, y así sucesivamente hasta conseguir la señal original.

### 2.1.2.2. La transformada de wavelet 2D

La generalización de la 1D-FWT puede realizarse con sencillez a dos o más dimensiones para su aplicación a datos multidimensionales [Mallat, 1989]. Para el caso de dos dimensiones, la transformada de wavelet se puede obtener aplicando la 1D-FWT sobre cada una de las dimensiones. Por ejemplo, dada una imagen  $f(x, y)$ , la transformada de wavelet se obtiene aplicando la 1D-FWT sobre cada una de las filas que componen la imagen, y posteriormente, sobre cada una de las columnas de la misma, o lo que es lo mismo, se aplican los filtros  $H$  y  $G$  sobre cada una de las filas y sobre cada una de las columnas de la imagen como en el caso unidimensional. Este proceso se denominada transformada de wavelet 2D (2D-FWT), y se puede aplicar de forma repetida como en la 1D-FWT.

Dada una imagen,  $f(x, y)$ , una aplicación de la 2D-FWT descompone la imagen en cuatro sub-bandas:

- Sub-banda pasa baja-baja ( $HH$ ). Contiene una imagen que es una referencia de la original.
- Sub-banda pasa alta-baja ( $GH$ ). Contiene los detalles de la orientación horizontal de la imagen.
- Sub-banda pasa baja-alta ( $HG$ ). Contiene los detalles de la orientación vertical de la imagen.
- Sub-banda pasa alta-alta ( $GG$ ). Contiene los detalles de la orientación diagonal de la imagen.

El tamaño de cada una de las sub-bandas es un cuarto del tamaño de la imagen original. Para obtener una nueva iteración de la transformada, al igual que en el caso unidimensional, se aplica la 2D-FWT sobre la sub-banda  $HH$  o imagen de referencia, y se vuelven a generar cuatro sub-bandas, de la misma forma que en la primera iteración. El proceso se puede repetir en el mismo sentido para el resto de iteraciones. En la figura 2.3, se puede observar el resultado de la primera iteración de la 2D-FWT sobre una imagen (parte izquierda), y la descomposición que se produce tras la aplicación sucesiva de tres iteraciones de la 2D-FWT sobre la misma imagen (parte derecha). Como se puede observar, la aplicación sucesiva de la 2D-FWT genera una descomposición de la imagen jerárquica o piramidal. En la figura 2.4 se puede observar la descomposición de la imagen de un *mono* tras dos iteraciones y de la imagen *Lena* tras tres iteraciones de la transformada de

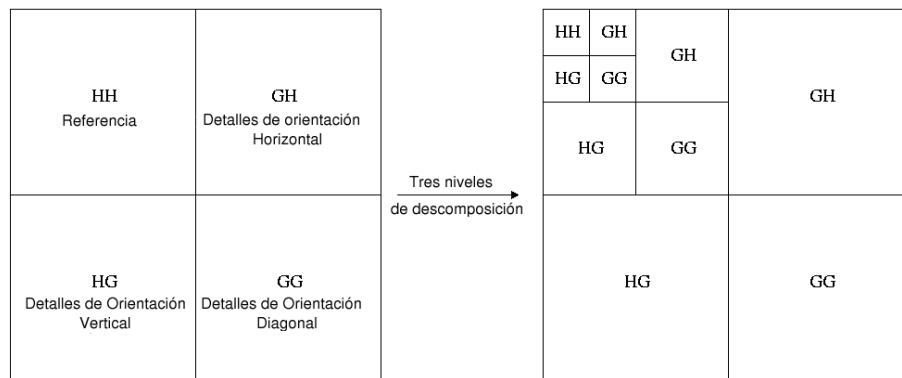


Figura 2.3: Descomposición en sub-bandas de una imagen tras una iteración (parte izquierda) y tres iteraciones (parte derecha) de la transformada de wavelet 2D

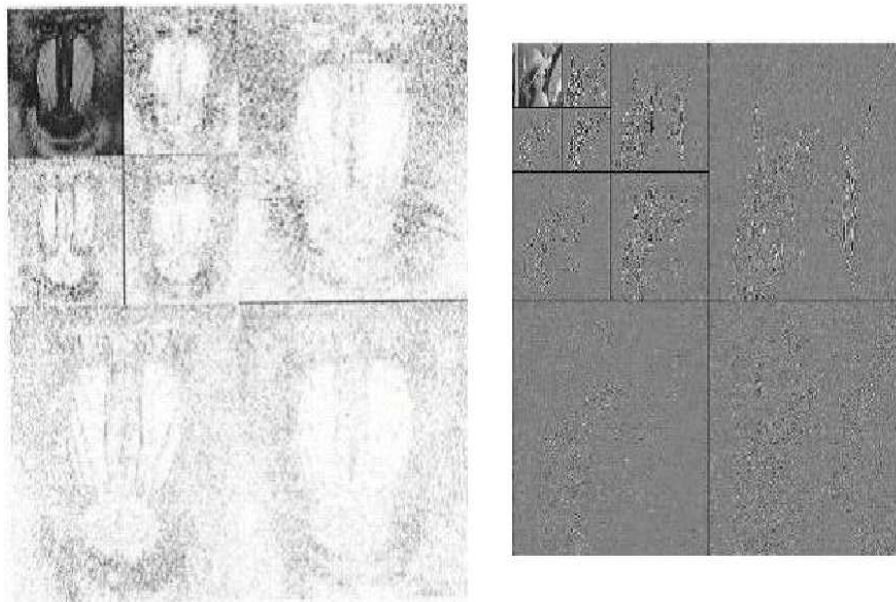


Figura 2.4: Descomposición de las imágenes tras dos iteraciones (mono –parte izquierda–) y tres iteraciones (Lena –parte derecha–) de la transformada de wavelet 2D

wavelet 2D. Se puede observar como la sub-banda HH contiene la información principal de cada una de las imágenes.

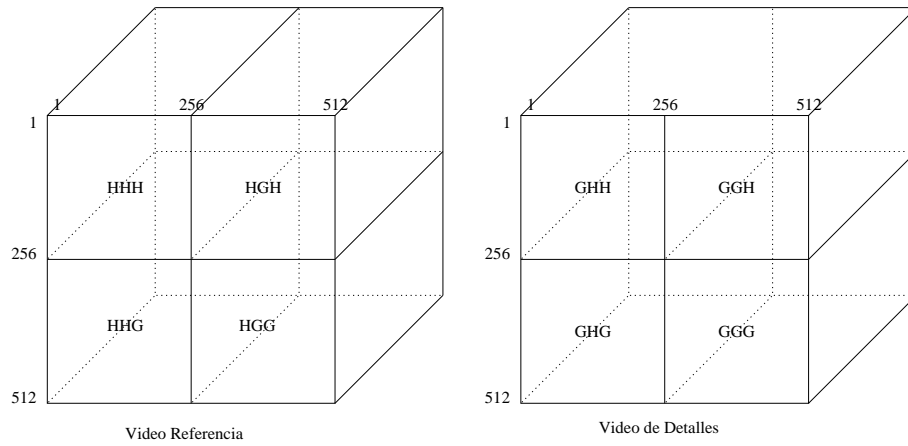


Figura 2.5: Descomposición de una secuencia de vídeo tras una iteración de la transformada de wavelet 3D

### 2.1.2.3. La transformada de wavelet 3D

De la misma forma que para la 2D-FWT, se puede generalizar la transformada de wavelet 1D para datos tridimensionales, por lo que se aplica la transformada de wavelet 1D sobre las tres dimensiones. Por ejemplo, dada una secuencia de video  $s(x, y, t)$ , tenemos una secuencia de imágenes de dos dimensiones en vez de una única imagen como en el caso de la 2D-FWT, ya que aparece una nueva dimensión que es el tiempo. La transformada de wavelet 3D se puede obtener aplicando la 1D-FWT sobre cada una de las dimensiones, de forma consecutiva.

La primera iteración de la transformada de wavelet 3D consiste en aplicar la 1D-FWT sobre la dimensión en el tiempo para obtener dos cubos: el video de referencia o parte baja y el video de detalles o parte alta. A continuación, se aplica la 2D-FWT sobre cada uno de las imágenes que componen la secuencia de video para obtener la descomposición presentada en la figura 2.3, por lo que se obtienen ocho sub-cubos, como podemos observar en la figura 2.5. Para obtener la segunda iteración de la transformada 3D, se aplica la misma sobre el sub-cubo de referencia ( $HHH$ ) para que se generen, de nuevo, ocho sub-cubos. Este proceso se denomina transformada de wavelet 3D (3D-FWT) y se puede repetir de forma consecutiva como en el caso de la 1D-FWT y la 2D-FWT.

## 2.2. La cuantificación

Una vez realizada la transformación de los valores de la señal de entrada, la cuantificación es un proceso que convierte los datos de entrada (que suelen ser continuos en magnitud), en un conjunto de valores finitos y diferentes. Por tanto, la cuantificación es realmente una discretización de los diferentes valores. Dicho proceso tiene un gran impacto sobre la tasa de compresión y la distorsión de la imagen o el vídeo reconstruido, ya que la cuantificación es un proceso irreversible en el que es imposible obtener el valor original a partir del valor cuantificado y en el que se produce una pérdida de información. A continuación se describen las principales formas de cuantificación existentes [Shi y Sun, 2000].

### 2.2.1. Cuantificación uniforme

La cuantificación uniforme es la forma más sencilla y popular dentro de las técnicas de cuantificación. La relación entre la entrada y la salida caracteriza al cuantificador. La señal de entrada se distribuye en una serie de intervalos uniformemente espaciados, donde los diferentes extremos de los intervalos son los denominados niveles de decisión ( $d_i$ ). Para cada intervalo se establece un valor de salida, es decir el correspondiente valor cuantificado, que puede venir expresado por la siguiente función matemática:

$$y_i = Q(x) \text{ si } x \in (d_i, d_{i+1}) \quad (2.11)$$

Los diferentes valores cuantificados se encuentran uniformemente espaciados y se corresponden con la media aritmética de los dos niveles de decisión del correspondiente intervalo al que pertenece el valor de entrada.

### 2.2.2. Cuantificación no uniforme

Al contrario de la cuantificación uniforme, en este caso la señal de entrada se distribuye en una serie de intervalos no uniformes. El número de valores cuantificados es finito pudiendo encontrar una serie de valores más cercanos a una determinada magnitud o ampliamente distribuidos a lo largo del rango de valores de salida. Por tanto, la cuantificación no uniforme distribuye los niveles de salida a partir de los valores estadísticos<sup>1</sup> de la señal de entrada.

---

<sup>1</sup>Se incluyen como valores estadísticos la media, la varianza y el tipo de la señal de entrada.

### 2.2.3. Cuantificación adaptativa

La cuantificación uniforme y no uniforme se caracterizan por ser invariantes en el tiempo, por tanto no son apropiadas para señales de entrada no estacionarias. La cuantificación adaptativa intenta diseñar el cuantificador para que se adapte a los valores estadísticos de la señal de entrada que van a ser variables. El precio de la cuantificación adaptativa viene determinado por un retraso en el proceso de cuantificación así como la necesidad de un espacio de almacenamiento adicional. Básicamente, existen los siguientes tipos de cuantificación adaptativa:

#### Cuantificación adaptativa directa

La entrada al cuantificador se divide en bloques de una determinada longitud. A cada uno de los bloques se le realiza un análisis estadístico, a partir del cual, se determinan los diferentes intervalos y los correspondientes valores cuantificados. La información del análisis estadístico se envía conjuntamente con la información cuantificada para realizar el proceso de decodificación. La elección del tamaño del bloque es crucial para la eficiencia de la cuantificación.

#### Cuantificación adaptativa inversa

En este caso, el análisis estadístico se realiza a partir de la salida del cuantificador, por lo que no es necesario transferir la información al decodificador. Se produce una degradación en la adaptación al cambio de los diferentes valores estadísticos, ya que se tiene en cuenta la salida del cuantificador en vez de la señal de entrada, para llevar a cabo el análisis estadístico.

#### Cuantificación adaptativa por selección

Se dispone de una serie de cuantificadores con una amplia variedad de señales de entrada y sus correspondientes salidas. Se realiza un análisis estadístico de las entradas o salidas recientes y se elige uno de los cuantificadores con el objetivo de realizar la mejor cuantificación posible.

## 2.3. La codificación entrópica

Una vez vista la cuantificación, el siguiente paso consiste en la codificación de los valores obtenidos. La codificación entrópica se define como un método de codificación sin pérdida de información, que reduce la redundancia explotando las características de los métodos de longitud

variable. De esta forma se aumenta la tasa de compresión sin que se produzca una degradación de la imagen o el video reconstruido. La idea básica consiste en conocer la probabilidad de aparición de los diferentes símbolos en un mensaje para codificar dicho mensaje con una menor cantidad de espacio.

El primer método que realiza una codificación eficiente de símbolos es el de Shannon-Fano. A partir de la probabilidad de aparición de los distintos símbolos que pueden aparecer en un mensaje, se construye una tabla de códigos que tiene las siguientes propiedades:

- A cada uno de los códigos se le asigna un número diferente de bits.
- El número de bits que se asigna a cada código depende de la probabilidad de aparición, estableciéndose un mayor número de bits para aquellos códigos que tienen una menor probabilidad de aparición y un menor número de bits para aquellos códigos que tienen una mayor probabilidad de aparición.
- Dado que cada uno de los códigos tiene un número de bits diferente, se pueden decodificar de forma unívoca y sin pérdida de información.

El algoritmo de Shanon-Fano ordena las probabilidades de aparición de cada símbolo de mayor a menor. A continuación, se agrupan los distintos símbolos en dos mitades, las cuales, deben de sumar la misma o aproximada cantidad de apariciones entre todos los símbolos que la componen. A una mitad se le asigna un cero, mientras que a la otra se le asigna un uno. Por tanto, los códigos de la primera mitad empezarán por cero y los de la segunda mitad por uno. El proceso de dividir en dos mitades y asignar un cero y un uno a cada una de las partes, se repite hasta que cada uno de los símbolos posibles tenga un código asignado. Por tanto, la evolución del algoritmo representa la creación de un árbol binario en el que cada uno de los códigos va a ser una de las hojas del árbol.

Partiendo de las ideas del algoritmo de Shanon-Fano se han desarrollado otros sistemas de codificación entrópica más eficientes. A continuación se describen los más representativos.

### 2.3.1. Codificación Huffman

De la misma forma que el algoritmo de Shanon-Fano, la codificación Huffman [1952] crea una tabla de códigos de longitudes diferentes. Dicha codificación presenta las siguientes características:

- Asigna a los símbolos con mayor frecuencia de aparición un código de menor longitud a costa de asignar códigos de mayor longitud a los símbolos con menor frecuencia de aparición.



- El número de bits de cada uno de los códigos es un número entero.
- Aunque los códigos asociados a símbolos diferentes tienen diferente longitud, éstos se pueden decodificar de forma única. Dicha característica recibe el nombre de propiedad del prefijo único y se basa en que ningún código de compresión es prefijo (forma parte inicial) de cualquier otro código.

Para construir un código de Huffman normalmente se utiliza un árbol binario que se construye a partir de las probabilidades de los símbolos que deseamos codificar. El diseño de dicho árbol se realiza de la siguiente forma:

1. Crear tantos nodos (que serán las hojas del árbol) como símbolos se vayan a codificar. En cada nodo almacenaremos el símbolo codificado y su probabilidad. Estos nodos forman la lista de nodos sin procesar.
2. Seleccionar dos nodos de la lista de nodos sin procesar que tengan la menor probabilidad o frecuencia de aparición.
3. Crear un nodo que es el padre de los dos nodos seleccionados. Este nodo no tiene asociado ningún símbolo y su probabilidad será la suma de las probabilidades de los nodos hijo.
4. Repetir los pasos 2 y 3 hasta que únicamente tengamos un nodo en la lista de nodos sin procesar, nodo que debe tener una probabilidad igual a 1 y que se llama nodo raíz del árbol de Huffman.

Una vez construido el árbol, el código asociado a un símbolo se calcula recorriendo el árbol desde la raíz hasta el símbolo codificado (situado siempre en una hoja). En la figura 2.6 se describe la construcción del árbol de Huffman para cinco símbolos distintos A, B, C, D y E con frecuencias de aparición de 15, 7, 6, 6 y 5 respectivamente. Por comodidad trabajaremos con pesos enteros y no con probabilidades reales.

El primer paso, figura 2.6(a), consiste en crear una lista de nodos sin procesar formada por los cinco símbolos. Gráficamente esta lista estará formada por los nodos que queden en el nivel superior. A continuación se extraen de la lista los nodos con símbolos D y E por ser los de menor peso. También se podrían seleccionar el nodo con símbolo C. El árbol resultante sería diferente (los códigos pueden variar tanto de valor como de longitud) pero equivalente desde el punto de vista de la compresión de datos. Los pesos de los nodos D y E son 6 y 5. Por tanto, insertamos como nodo padre de ambos símbolos un nuevo nodo en la lista, sin símbolo asociado y con un peso igual a 11

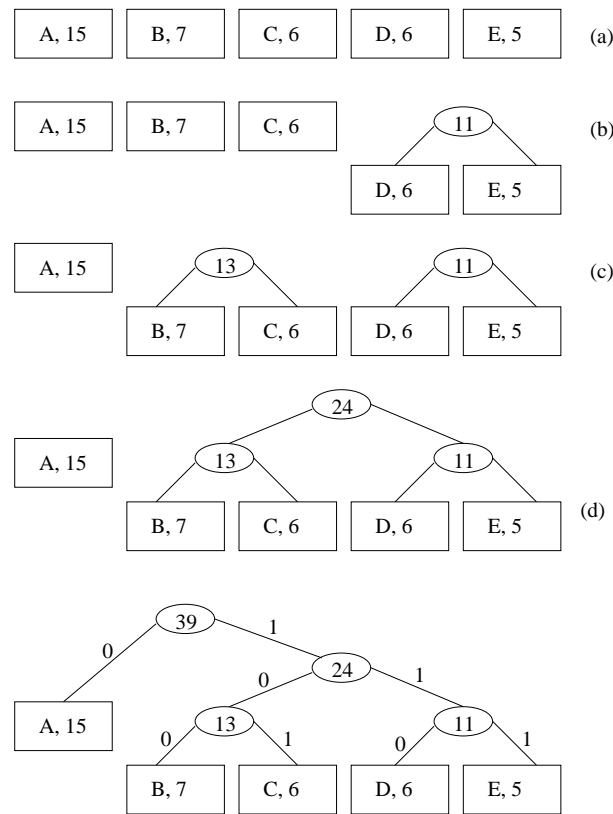


Figura 2.6: Ejemplo de construcción de un árbol de Huffman

(figura 2.6(b)). Los dos siguientes nodos utilizados para construir un nuevo subárbol son el B y el C (figura 2.6(c)). A continuación los dos subárboles forman un árbol mayor (figura 2.6(d)) y por último, el nodo A se procesa quedando el árbol de Huffman completo (figura 2.6(e)). Por tanto, se crea la tabla 2.1 de codificación Huffman, en la que cada posible símbolo tiene asignado un código determinado de bits. Dada una secuencia de símbolos, se codifica cada símbolo con el código de bits correspondiente. La decodificación realiza el proceso inverso, asignando a cada código de bits el símbolo correspondiente por lo que no se produce ninguna pérdida en la información codificada.

### 2.3.2. Codificación Aritmética

La codificación aritmética [Langdon y Rissanen, 1981][Rissanen y Langdon, 1979] mejora la idea de codificar cada símbolo de entrada con un código específico, tal y como establece Huffman, para aumentar la compresión de la información. Para ello se va a codificar una secuencia de símbolos mediante un número real de simple precisión.

Símbolos	Código
A	0
B	100
C	101
D	110
E	111

Tabla 2.1: Tabla de codificación Huffman

A cada posible secuencia de símbolos se le asigna un subintervalo dentro del intervalo  $[0, 1)$  cuyo tamaño es igual al producto de todas las probabilidades de los símbolos que forman la secuencia. La posición del subintervalo dentro del intervalo real  $[0, 1)$  está determinado por la secuencia concreta de símbolos. El algoritmo básico de codificación consiste en emitir un número cualquiera perteneciente al subintervalo final. El número de bits de código necesarios para representar el código aritmético, es igual a la entropía de la secuencia de símbolos y por esta razón, la codificación aritmética se considera 100 % eficiente.

El algoritmo básico de codificación es el siguiente:

1. Sea  $[L, H) \leftarrow [0, 1)$  el intervalo inicial.
2. Mientras existan símbolos de entrada:
  - Dividir el intervalo  $[L, H)$  en tantos subintervalos como símbolos diferentes existen en el alfabeto. El tamaño de cada subintervalo es proporcional a la probabilidad del símbolo asociado.
  - Seleccionar de entre todos los subintervalos, el que corresponde al símbolo codificado en la iteración actual. Sea el intervalo elegido  $[L', H')$ .
  - Hacer  $[L, H) \leftarrow [L', H')$ .
3. Emitir un número  $x \in [L, H)$  como código aritmético. El número de cifras deberá permitir distinguir el intervalo final  $[L, H)$  de cualquier otro.

Supongamos la secuencia de entrada *aab* de la que se desprende una probabilidad de  $2/3$  para el símbolo *a* y de  $1/3$  para el símbolo *b*. Inicialmente  $L = 0$  y  $H = 1$ . Dividimos en dos subintervalos: el  $[0, 2/3)$  asociado al símbolo *a* y el  $[2/3, 1)$  asociado al símbolo *b*. En la primera iteración codificamos una *a* por lo que hacemos  $L = 0$  y  $H = 2/3$ . En la segunda iteración

dividimos el intervalo actual  $[0, 2/3)$  en los subintervalos  $[0, 4/9)$  y  $[4/9, 2/3)$ . Ya que codificamos otra  $a$ , hacemos  $L = 0$  y  $H = 4/9$ . En la tercera y última iteración dividimos el intervalo  $[0, 4/9)$  en  $[0, 8/27)$  y  $[8/27, 4/9)$ . Al codificar una  $b$  hacemos  $L = 8/27$  y  $H = 4/9$ . Ahora buscamos un real  $x \in [8/27, 4/9)$  de forma que tenga un número de cifras mínimo. Si utilizamos la base decimal,  $x = 0,3\epsilon[0,2962929\dots, 0,444\dots)$ . De igual forma  $x$  podría haber sido 0,4. En código binario, un real perteneciente al intervalo y con un número de cifras mínimo es  $x = 0,011$ . Al tratarse siempre de un número menor que la unidad, la parte entera de la representación puede omitirse, lo que implica que la secuencia  $aab$  puede codificarse en decimal por 3 y en binario por 011.

De forma análoga, el algoritmo básico de decodificación es el siguiente:

1.  $[L, H) \leftarrow [0, 1)$ .
2. Mientras sea posible decodificar símbolos:
  - Dividir el intervalo  $[L, H)$  en subintervalos de tamaño proporcional a las probabilidades de los símbolos.
  - Seleccionar el subintervalo  $[L', H')$  al que el código aritmético  $x$  pertenece.
  - Emitir el símbolo asociado al intervalo  $[L', H')$ .
  - $[L, H) \leftarrow [L', H')$ .

A partir del código aritmético 3 (0,3 en realidad), que hemos codificado anteriormente, y las probabilidades  $2/3$  y  $1/3$  para los símbolos  $a$  y  $b$  respectivamente, comenzamos la decodificación. Inicialmente  $L = 0$  y  $H = 1$ . Dividimos este intervalo en dos subintervalos  $[0, 2/3)$  y  $[2/3, 1)$ . Seleccionamos el primer intervalo pues  $0,3\epsilon[0, 2/3)$ . Así, el primer símbolo decodificado es una  $a$ . Tomamos como intervalo actual dicho subintervalo que es dividido en  $[0, 4/9)$  y  $[4/9, 2/3)$ . Como  $0,3\epsilon[0, 4/9)$ , el segundo símbolo decodificado es una  $a$ . De nuevo dividimos este intervalo en  $[0, 8/27)$  y  $[8/27, 4/9)$ . Como  $0,3\epsilon[8/27, 4/9)$ , el último símbolo codificado es una  $b$ . Por tanto, la secuencia decodificada es  $aab$  demostrando la ausencia de pérdida de información de la codificación aritmética.

### 2.3.3. Codificación *Run-Length*

La codificación *Run-Length* se basa en el siguiente principio: Si un símbolo  $d$  se repite  $n$  veces de forma consecutiva en una secuencia de entrada, dicha secuencia se reemplaza o se codifica mediante  $nd$ . Por ejemplo, la codificación 10 17 representa que el símbolo 17 se ha repetido 10

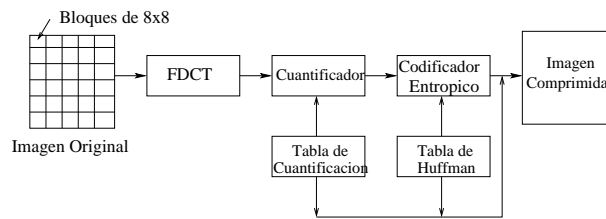


Figura 2.7: Diagrama de bloques del codificador JPEG

veces de forma consecutiva. La decodificación realiza el proceso inverso sin pérdida alguna, ya que repite el valor codificado tantas veces como indique el valor de repetición.

## 2.4. Codificadores basados en una transformada

En este apartado se describen las características principales de los codificadores basados en una transformada más utilizados para la compresión de imágenes y secuencias de vídeo.

### 2.4.1. Estándares de compresión basados en una transformada

En primer lugar, analizaremos los principales métodos que se han convertido en un estándar para la compresión de imágenes o secuencias de vídeo.

#### 2.4.1.1. JPEG (Joint Photographic Experts Group)

En 1991, JPEG [Wallace, 1991] se convierte en el primer estándar para la compresión de imágenes en color y escala de grises. Se especifican tres modos de operación (secuencial, progresivo y jerárquico) basados en la transformada del coseno para compresión con pérdida de información y un modo para compresión sin pérdida basada en codificación predictiva. Sin embargo, el esquema JPEG por defecto es el denominado codificador JPEG que podemos observar en la figura 2.7.

La imagen original se divide en bloques de  $8 \times 8$  pixels. A cada bloque de datos se le aplican tres procesos para obtener la imagen comprimida. En primer lugar, se lleva a cabo la transformada discreta del coseno. A continuación, se realiza una cuantificación para reducir la precisión y, finalmente, se realiza una codificación entrópica utilizando *Run-Length* y un codificador aritmético o basado en Huffman.

La FDCT decorrelaciona los datos, concentrando la mayor parte de la señal en las frecuencias espaciales bajas, por lo que la mayoría de los coeficientes obtenidos serán cero o estarán cercanos a

cero. La cuantificación se lleva a cabo de forma uniforme para cada grupo de 64 coeficientes DCT, aplicando la misma tabla de cuantificación que se utilizará posteriormente en el decodificador para obtener los valores originales.

A continuación, los valores cuantificados son codificados entrópicamente utilizando un codificador *Run-Length* en “zig-zag”. Este esquema beneficia la transformación en el dominio de la frecuencia del coseno, según la cual, los valores en la esquina superior izquierda del bloque representan las frecuencias de orden menor (mayor magnitud o referencia de la imagen), mientras que los valores de la esquina inferior derecha representan las frecuencias de orden mayor (menor magnitud o detalles de la imagen). Por lo tanto, el orden en “zig-zag” favorece la aparición de ceros consecutivos, sobre todo a medida que nos acercamos a la esquina inferior derecha, por lo que el *Run-Length* proporciona una mayor tasa de compresión sin afectar a la calidad de la imagen reconstruida. Cada término codificado contiene tres valores: longitud de serie, tamaño y el valor. La longitud representa el número de términos con valor cero precediendo al término actual. El tamaño representa el número de bits usados para representar el valor del dato, mientras que el valor, representa al valor propiamente dicho del término. Finalmente, se lleva a cabo un codificador basado en Huffman o aritmético, para aumentar aún más la tasa de compresión y mantener la calidad de la imagen reconstruida.

#### 2.4.1.2. JPEG-LS (Lossless and near-lossless coding of continuous tone still images)

JPEG-LS [(JPEG/JBIG), 1992] es el estándar de compresión sin pérdidas para imágenes en escala de grises. El estándar está basado en el algoritmo LOCO-I [Weinberger *et al.*, 1996] (Compresión sin pérdidas para imágenes con un nivel de complejidad bajo). El algoritmo combina las mejores características de la simplicidad de la codificación Huffman y la potencialidad de compresión de los modelos de contexto. Utiliza un modelo de contexto simple y fijo, que alcanza la capacidad de las técnicas complejas de modelado de contexto universal, para la captura de dependencias de orden superior. LOCO-I obtiene tasas de compresión iguales o superiores a los algoritmos basados en codificación aritmética, pero con una menor complejidad.

#### 2.4.1.3. JPEG-2000

El uso de la transformada discreta del coseno sobre bloques de  $8 \times 8$  pixels en el estándar JPEG provoca la reconstrucción de imágenes (sobre todo cuando hay una gran pérdida de información) en la que aparecen artefactos de bloque. El estándar JPEG-2000 [Bolić, 2000][Marcellin *et al.*, 2000][Santa-Cruz y Ebrahimi, 2000] para la compresión y codificación digital de imágenes estáticas

se diseña para resolver las limitaciones de JPEG e incorporar nuevas funcionalidades. Por ello, la principal característica del nuevo estándar es el uso de la transformada de wavelet para mejorar la eficacia del codificador en la compresión con o sin pérdida de información. Como características adicionales de JPEG-2000 podemos destacar las siguientes:

- Tasa de compresión elevada (estado del arte).
- Proporcionar una mejor calidad de imagen, tanto desde el punto de vista objetivo (tasa de compresión) como subjetivo (sistema visual humano), especialmente para tasas de compresión muy altas.
- Permite la transmisión progresiva de imágenes por calidad, resolución, componente de color o región de interés.
- Acceso directo (espacial) fácil y rápido a la secuencia comprimida.
- Posibilidad de realizar *pan* y *zoom* en el proceso de decodificación sobre un subconjunto de la imagen comprimida.
- El decodificador puede rotar y cortar la imagen durante la decodificación.
- Implementación limitada de memoria y sin excesiva complejidad computacional.
- Codificación progresiva de regiones de interés.
- Mayor tolerancia a fallos. Se pueden incluir códigos de corrección de errores en la codificación para mejorar la transmisión en entornos con posibilidad de ruido.

JPEG-2000 procesa imágenes en color y en escala de grises. En el primer caso las imágenes se dividen en las tres componentes de color para realizar el proceso de codificación. Cada una de las mencionadas componentes de color o la imagen en escala de grises se divide en varias regiones rectangulares no solapadas, denominadas *tiles*, con tamaños diferentes o incluso el tamaño total de la imagen, las cuales se codifican de forma independiente mediante la aplicación consecutiva de los siguientes procesos:

1. Cálculo de la transformada de wavelet. Se puede aplicar la transformada basada en enteros o en reales. El número de iteraciones de la transformada es uno de los parámetros a especificar para el proceso de codificación.

2. Cuantificación de los coeficientes wavelet. Cada sub-banda wavelet puede disponer de un factor de cuantificación diferente. Por tanto, cada coeficiente wavelet se divide por el factor de cuantificación y se realiza un truncamiento del resultado. Si la compresión es sin pérdida el factor de cuantificación es 1.
3. Utilización del codificador MQ para codificar aritméticamente los coeficientes wavelet por planos de bits. Se utiliza el algoritmo EBCOT [Taubman, 2000] que se describirá en la sección 3.2.1.

#### 2.4.1.4. MPEG (Moving Picture Expert Group)

MPEG constituye un grupo de trabajo de la Organización Internacional de Estándares (ISO), creado en 1988, para el desarrollo de estándares para la codificación audio-visual de la información (películas, secuencias de vídeo, música) en formato digital.

MPEG-1 [MPEG, 1993] fue el primer estándar desarrollado por MPEG. El objetivo principal consistía en la codificación de imágenes en movimiento y el audio asociado, proporcionando un ancho de banda de 1,5 Mbits/s. En términos prácticos, se trata de un estándar para el almacenamiento y recuperación eficiente de audio y video de un compact disc (CD). MP3 o Video CD son algunas de las aplicaciones que están basadas en MPEG-1.

El desarrollo de extensiones a MPEG-1, para permitir una mayor complejidad del formato de entrada, tasas de compresión mayores y una mejor tolerancia a fallos, fueron los pilares básicos para el desarrollo de MPEG-2 [MPEG, 1994]. MPEG-2 se denomina codificación genérica de imágenes en movimiento y el audio asociado. MPEG-2 proporciona soporte para la transmisión eficiente a través de sistemas propensos a errores, codificación eficiente de imágenes entrelazadas con diferente resolución espacial, codificación de audio multicanal, protocolos a nivel de sesión para la comunicación a través de redes diferentes y control remoto de un servidor MPEG-2, codificación de video sobre muestras representadas con una precisión superior a 8 bits y una interface en tiempo real entre la capa de transporte de MPEG-2 y un decodificador. La televisión digital de alta definición (HDTV) o el DVD constituyen las principales aplicaciones que están basadas en MPEG-2.

Todos los estándares MPEG son genéricos, es decir, independientes de la aplicación. Los algoritmos de compresión de vídeo de MPEG-1 y MPEG-2 están basados en las ideas introducidas por JPEG, por lo que nos referiremos como MPEG en general. Se trata de un esquema híbrido de compresión que utiliza dos técnicas: la compensación de movimiento basada en bloques para reducir la redundancia temporal (compresión *inter-frame*), y la codificación basada en la



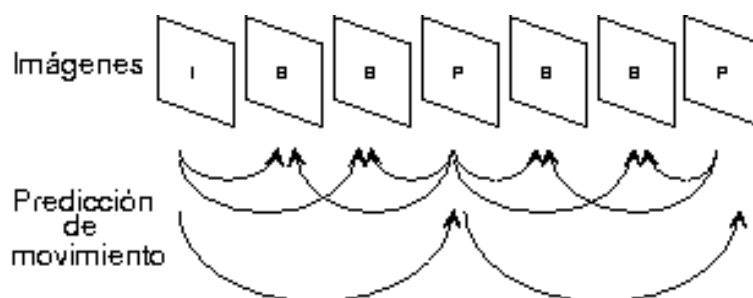


Figura 2.8: Tipos de imágenes en un codificador MPEG

transformada discreta del coseno para reducir la redundancia espacial (compresión *intra-frame*). Inicialmente, la secuencia de vídeo se divide en grupos de imágenes (GOP). Cada GOP incluye tres tipos de imágenes posibles, atendiendo al tipo de compresión empleado para codificar cada una de éstas imágenes. Como primer tipo tenemos las *imágenes intra-codificadas* (I), las cuales, se codifican como una imagen aislada utilizando el esquema descrito para JPEG con pequeñas variaciones. La distancia entre dos imágenes I viene determinada por el tamaño del GOP. En segundo lugar y dado que la mayoría de los píxeles de una imagen no cambian con respecto a la siguiente, las *imágenes predichas* (P) codifican los píxeles que han cambiado con respecto a la imagen de referencia. Finalmente, las *imágenes interpoladas bi-direccionalmente* (B) se codifican con respecto a dos imágenes, una anterior y otra posterior, aunque no se utilizan en ningún caso como referencia, en la cual se codifican las regiones de la imagen que cambian de forma gradual a lo largo de la mayoría de las imágenes que componen la secuencia de vídeo. En la figura 2.8 se observa la relación entre los tres tipos de imágenes.

Los distintos tipos de imágenes (I, P y B) se dividen en bloques de  $16 \times 16$  píxeles denominados macrobloques. Un macrobloque contiene cuatro bloques de luminancia y dos de crominancia (formato 4:2:0) de  $8 \times 8$  píxeles, siendo dicho macrobloque la unidad básica para calcular la estimación de movimiento, aplicar la DCT y realizar la cuantificación. Para explotar la redundancia temporal, se estiman los vectores de movimiento para cada macrobloque a partir de dos imágenes originales de luminancia mediante un algoritmo de búsqueda basado en bloques. Una vez calculado los vectores de movimiento, los valores de los píxeles para el macrobloque a codificar se pueden predecir a partir de la imagen previamente decodificada. Todos los macrobloques de una imagen I se codifican sin compensación de movimiento, mientras que los macrobloques de las imágenes P y B se pueden codificar utilizando la compensación de movimiento. Una vez que se ha elegido el modo de codificación, la predicción con compensación de movimiento del contenido del bloque se hace a partir de la imagen de referencia (I o P) pasada (caso de las imágenes P) y eventualmente

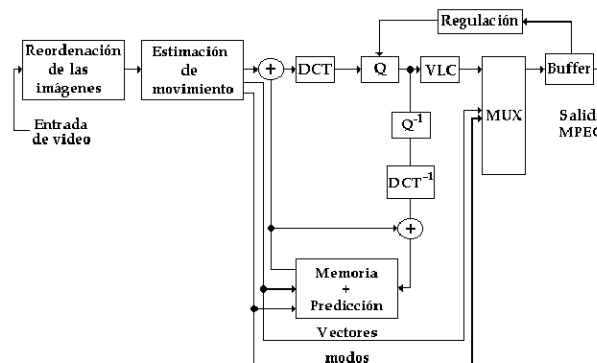


Figura 2.9: Diagrama de bloques del codificador de vídeo MPEG

futura (caso de las imágenes B). La predicción se elimina de los datos reales del macrobloque, lo que da la señal de error de predicción.

Con respecto a la redundancia espacial, el estándar MPEG divide en bloques de  $8 \times 8$  pixels y aplica la transformada discreta del coseno (2D-DCT) sobre los distintos tipos de imágenes. A continuación, se lleva a cabo un proceso de cuantificación uniforme para reducir el número de bits para representar los coeficientes y aumentar el número de coeficientes con valor cero. La cuantificación suprime los coeficientes con mayores frecuencias y representa con un número adecuado de bits las frecuencias menores, sin afectar a la distorsión y teniendo en cuenta la subjetividad del sistema visual humano. De hecho, el cuantificador dispone de matrices diferentes para los bloques intra-codificados e inter-codificados, ya que los estadísticos de las imágenes I son totalmente diferentes a los de las imágenes P o B. Las imágenes I distribuyen la energía a lo largo del dominio de la frecuencia, por lo que se pueden producir artefactos de bloque si la cuantificación se realiza con coeficientes de cuantificación de elevado peso, mientras que las imágenes codificadas tras la predicción del movimiento contienen una mayoría de coeficientes con frecuencias altas por lo que se pueden cuantificar con coeficientes de cuantificación de elevado peso. Finalmente, el codificador entrópico se compone de un codificador *Run-Length* en “zig-zag”, de igual forma y por las mismas razones que JPEG, y de un codificador de Huffman o aritmético para aumentar la tasa de compresión sin afectar a la distorsión final de la secuencia de vídeo. La figura 2.9 muestra el diagrama de bloques del codificador de vídeo MPEG.

La decodificación es más sencilla que la codificación, ya que no tiene que efectuar el proceso

de estimación de movimiento, que es una de las partes más complejas del codificador. Para las imágenes I, la decodificación consiste en aplicar a cada bloque la decodificación entrópica (Huffman y *Run-Length*), el proceso de cuantificación inversa de los coeficientes y la transformada DCT inversa. Para las imágenes P o B, este proceso consiste en construir la predicción de cada macrobloque a partir de su tipo, de los vectores de movimiento y de las imágenes de referencia memorizadas. El decodificador lee, decodifica y realiza el proceso de cuantificación inversa de los coeficientes DCT para cada bloque de 8x8 pixels, y, después de la transformada DCT inversa, añade el resultado a la predicción. La reconstrucción de la imagen se efectúa cuando todos los macrobloques han sido tratados. Por último, las imágenes se ordenan según el orden inicial de visualización.

El estándar MPEG-4 [Battista *et al.*, 1999][Battista *et al.*, 2000] proporciona una forma para crear y difundir contenidos multimedia e interactivos en entornos múltiples. Proporciona un amplio rango de aplicaciones (64Kbps–4Mbps) independientemente del medio de transmisión, aunque se ha centrado principalmente en tres ámbitos o sectores de aplicabilidad: televisión digital, aplicaciones gráficas interactivas y multimedia interactiva. Se requiere la comunicación de información audiovisual natural o sintética y la posibilidad de realizarla en tiempo real. El estándar proporciona vídeo y audio con cualquier tipo de interactividad con el usuario. Por lo tanto, el objetivo principal de MPEG-4 reside en la codificación audiovisual de aplicaciones multimedia que permitan interactividad, elevada compresión, escalabilidad del vídeo y del audio, y soporte para vídeo y audio con contenidos reales y sintéticos en 2D y 3D. Además, los algoritmos de codificación de vídeo de MPEG-4 contienen toda la funcionalidad de MPEG-1 y MPEG-2.

El estándar MPEG-4 define “objetos media o audiovisuales” para permitir la creación de elementos interactivos e incluye herramientas para manejar la distribución en una amplia variedad de entornos y el ajuste a la velocidad de conexión. Las escenas audiovisuales se componen de varios objetos media organizados jerárquicamente:

- Objetos de audio natural: procedentes de grabaciones.
- Objetos de audio sintéticos: generador de sonidos.
- Objetos de vídeo natural: representa imágenes naturales.
- Objetos *Still Texture*: codificará objetos de sólo textura, como pueden ser fondos de aplicaciones, textura de regiones etc.
- Objetos 2D/3D *Mesh*: representan objetos de 2D o 3D sintéticos.

- Objetos FBA (*Facial and Body Animation Object*): son representaciones sintéticas tanto de caras como de cuerpos humanos.

Es importante destacar que cada objeto se puede tratar (visualizar, editar, escalar, ...) por separado e independiente de los demás. La estructura jerárquica permite a los creadores construir escenas complejas y a los usuarios manipular de forma consistente objetos y grupos de objetos. Para construir una escena se dispone de la posibilidad de ubicar objetos audiovisuales en un sistema de coordenadas, aplicar transformaciones para cambiar la apariencia geométrica acústica de un objeto, agrupar objetos para formar objetos nuevos, modificar los atributos de los objetos según un flujo de datos y cambiar interactivamente la posición de los objetos y el punto de vista.

Se define un VOP como cada uno de los muestreos que se hacen del estado de un objeto de vídeo en un instante determinado de tiempo. Cada VOP puede ser codificado de manera independiente, o en relación con otros VOPs (I, P o B) mediante el uso de compensación de movimiento. A su vez, el VOP se divide en macrobloques. Un macrobloque contiene toda la información codificada sobre la forma, textura y movimiento de la parte del objeto que codifica. El movimiento y la textura siguen la misma filosofía que MPEG-2, aunque introducen algoritmos más eficientes. Para la codificación de la forma se emplean dos métodos: codificación binaria y codificación en escala de grises. En la primera se utiliza un etiquetado para definir que conjunto de pixels pertenecen al objeto de vídeo en cada instante de tiempo, mientras que la segunda es una generalización del concepto de codificación binaria, que hace posible la representación de objetos transparentes. Entre las diferentes opciones de codificación, el estándar MPEG-4 permite la codificación de texturas mediante un algoritmo basado en la transformada de wavelet (*zerotree* [Shapiro, 1993]) que proporciona una excelente calidad para grandes tasas de compresión.

MPEG-7 [Martínez, 2003] es una interfaz de descripción de contenidos multimedia creado para describir la inconmensurable cantidad de información audiovisual existente y el más que seguro aumento de la misma. El valor de la información depende de lo fácil que pueda ser encontrada, recuperada, accedida, filtrada y manejada. Las bases de datos de información multimedia permiten la búsqueda de imágenes utilizando características como su color, textura o información de su forma o contorno de los objetos. La descripción estará asociada con el contenido en sí mismo, para permitir una rápida y eficiente búsqueda del material en el que un usuario puede estar interesado. Existen muchas aplicaciones y dominios de aplicación que se beneficiarán del estándar MPEG-7, como por ejemplo librerías digitales, servicios de directorio multimedia, selección de medios de difusión, edición multimedia, etc.

#### 2.4.1.5. H.261

A finales de 1984, la CCITT (actualmente la ITU-T) creó un grupo de trabajo de expertos (también participaron en el desarrollo de MPEG), para desarrollar un estándar para videotelefonía sobre las líneas telefónicas existentes RDSI, las cuales tenían un ancho de banda estrecho. Fruto de este trabajo surgió el estándar H.261 aplicable tanto a videotelefonía como a videoconferencia para caudales de  $px64$  kbps, donde  $p$  se sitúa entre 1 y 30. La característica más importante de este codificador es que debe estar diseñado para trabajar en tiempo real tanto en la codificación como en la decodificación. Al igual que MPEG, el codificador utiliza el concepto de macrobloque, codificación inter-*frame* basada en la compensación de movimiento, transformada discreta del coseno sobre bloques de  $8 \times 8$  pixels, cuantificación, *Run-Length* en “zig-zag” y codificación Huffman. Como principales diferencias con respecto a MPEG podemos encontrar la cuantificación de todos los coeficientes DCT por igual, mediante un coeficiente de cuantificación, la imposibilidad de codificar imágenes interpoladas bi-direccionalmente o de tipo B y la limitación de un área de búsqueda para la estimación de movimiento a  $\pm 15$  pixels como máximo, ya que el estándar está diseñado para aplicaciones de videotelefonía y videoconferencia, donde las imágenes presentan poco movimiento (normalmente, representan imágenes de una persona).

#### 2.4.1.6. H.263

H.263 [ITU-T, 1996] se basa en H.261 y se diseñó para comunicaciones con pequeños requisitos de ancho de banda, normalmente, se sitúan por debajo de 64 Kbits/s. Al igual que H.261, el codificador contiene una estructura híbrida entre predicción inter-*frame* (DPCM) para explotar la redundancia temporal basado en la estimación del movimiento, y codificación basada en la transformada DCT por bloques de  $8 \times 8$  pixels para explotar la redundancia espacial. El estándar incluye una serie de opciones para mejorar el rendimiento que le permite alcanzar en una videoconferencia sobre RDSI anchos de banda entre 64 y 256 Kbit/s, con imágenes en formato CIF ( $352 \times 288$  pixels) y QCIF ( $176 \times 144$  pixels).

### 2.4.2. Métodos de compresión basados en la transformada de wavelet 2D para imágenes y 3D para vídeo e imágenes volumétricas

Sin duda alguna, la principal aplicación de la transformada de wavelet es la compresión de imágenes y secuencias de vídeo. El principal objetivo de la compresión de información radica en el almacenamiento eficiente de la misma. Según Mallat [1989], los métodos de descomposición de

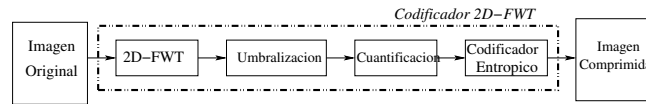


Figura 2.10: Procesos de un codificador basado en la transformada de wavelet 2D (2D-FWT)

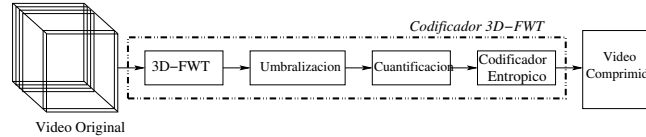


Figura 2.11: Procesos de un codificador basado en la transformada de wavelet 3D (3D-FWT)

la señal en distintas frecuencias son adecuados para la compresión de imágenes y secuencias de vídeo, gracias a las propiedades estadísticas de las imágenes y al hecho de que la representación de éstas se aproximan a la sensibilidad del sistema visual humano. Un codificador basado en la aplicación de la transformada de wavelet 2D o 3D consta de las siguientes fases o procesos comunes que se pueden observar en las figuras 2.10 y 2.11.

Como se ha definido anteriormente, la idea general de la compresión por transformada se basa en la obtención de una representación alternativa de la imagen o la secuencia de vídeo, de tal manera que la energía de la misma queda concentrada de forma selectiva en unos cuantos coeficientes. Dada una imagen o una secuencia de vídeo se aplica la transformada 2D-FWT o 3D-FWT para decorrelacionar la imagen o la secuencia de video original, respectivamente. Los coeficientes de las sub-bandas o sub-cubos de detalle tienen una baja entropía y generalmente contienen muchos coeficientes cercanos a cero, con pocos coeficientes de alta magnitud, que corresponden a los bordes de la imagen o la secuencia de vídeo original en la correspondiente localización espacial y de la frecuencia [Anderson, 1995]. Por tanto, la mayor parte de los coeficientes van a estar cercanos a cero, por lo que se elige un determinado umbral en el proceso de umbralización, para eliminar una serie de coeficientes y conseguir una determinada tasa de compresión. El resto de coeficientes se guardan después de una cuantificación para generar posteriormente la imagen o la secuencia de vídeo reconstruida. Por tanto, las técnicas de codificación de imágenes y secuencias de vídeo basadas en la transformada de wavelet están asociadas a la codificación y cuantificación efectiva de los coeficientes de alta magnitud que representan la mayoría de la energía de la imagen o la secuencia de vídeo [Anderson, 1995]. Tras la fase de cuantificación, la codificación entrópica se encarga de aumentar la tasa de compresión sin modificar la calidad de la imagen o el vídeo reconstruido.

Diferentes opciones en cada una de las fases descritas han dado lugar a la creación de una serie

de codificadores basados en la transformada de wavelet 2D y 3D para la compresión de imágenes [Antonini *et al.*, 1992][Lewis y Knowles, 1992][Shapiro, 1993] [Xing, 1994][Said y Pearlman, 1996b][Said y Pearlman, 1996a] [Buccigrossi y Simoncelli, 1997][Rabinovitch, 1997][Chen *et al.*, 1997][Wang y Kuo, 1997] [Taubman, 2000][Saha y Vemuri, 1999][Saha y Vemuri, 2000], secuencias de vídeo [Hilton *et al.*, 1994] [Taubman y Zakhor, 1994][Chen y Pearlman, 1996][Kim y Pearlman, 1997] [Kim y Pearlman, 1998][Wang *et al.*, 1999][Levy y Wilson, 1999], e imágenes volumétricas [Muraki, 1992][Muraki, 1995][Ihm y Park, 1998][Ihm *et al.*, 1998] [Bilgin *et al.*, 1998][Xiong *et al.*, 1998][Kim y Pearlman, 1999], cuyas principales características se desarrollaran en la sección 3.2. De forma adicional, se han desarrollado codificadores basados en la transformada de wavelet 2D para la compresión de imágenes particulares como huellas dactilares [Brislawn, 1994] o imágenes del METEOSAT [Acheroy y Grandjean, 1994] que pasamos a describir brevemente.

### Compresión de huellas dactilares (FBI)

Entre 1924 y 1995, el FBI almacenó 30 millones de huellas dactilares. Hopper, Bradley y Brislawn [1994] desarrollaron un codificador de imágenes de huellas dactilares basado en la transformada de wavelet 2D con funciones wavelet madre biortogonales y simétricas [Daubechies, 1992], que se aplicaban cinco veces de forma consecutiva sobre cada una de las dimensiones de la imagen. Además, utilizaron un cuantificador escalar (WSQ) uniforme y un codificador de Huffman como codificador entrópico. El codificador conseguía una tasa de compresión media de 0,75 bits/pixels (alrededor de 15:1), sobre imágenes de 500x500 pixels. El FBI tenía una base de datos de 200 millones de registros, cuyo almacenamiento, unos 2000 terabytes, disminuyó con el desarrollo del codificador anterior.

### Compresión de imágenes del METEOSAT

El satélite METEOSAT produce una imagen (37,5 Mbytes) cada media hora. La capacidad de almacenamiento necesaria para un día es de 1,8 Gbytes. Además, las imágenes producidas por el METEOSAT deben ser almacenadas con una calidad excelente desde el punto de vista del sistema visual humano, ya que deben ser observadas para detectar correctamente los fenómenos meteorológicos. Teniendo en cuenta las características anteriores, Acheroy y Grandjean [1994] desarrollaron un esquema basado en la transformada wavelet 2D y orientado a la compresión de imágenes del METEOSAT, en el que se utiliza una cuantificación uniforme que asigna un número de bits suficiente y dependiente de la posición de cada uno de los coeficientes, para que no se produzcan pérdidas en la reconstrucción de las imágenes. En cuanto a la umbralización,

se selecciona un valor entre 0 y 5 para eliminar (poner a cero) todos los coeficientes que son menores que dicho umbral, de forma que el error cuadrático medio de la imagen reconstruida no es significativo. Finalmente, en la codificación entrópica se utiliza un codificador *Run-Length* y un codificador de Huffman. El codificador obtenía tasas de compresión superiores a 8 : 1, y conseguía un rendimiento mejor que el estándar de compresión de imágenes JPEG.

## 2.5. Otras aplicaciones de la transformada de wavelet

Aunque la compresión de imágenes y secuencias de vídeo constituye la principal fuente de aplicación de la transformada de wavelet, en este apartado se describen otras aplicaciones de la citada transformada como la eliminación de ruido [Donoho, 1993][Chang *et al.*, 1997][Chang y Vetterli, 1997], la detección de comportamientos auto-similares en series de tiempo [Scargle *et al.*, 1993], la recuperación de imágenes basadas en el contexto [Wang *et al.*, 1997][Brambilla *et al.*, 1999] y el diseño de un sistema de seguridad basado en un reconocimiento facial [Martínez, 2002]. Esta variedad de usos denota la importancia que ha adquirido la transformada de wavelet en diferentes campos de aplicación.

### Eliminación de ruido

Donoho [1993] desarrolló el método de umbralización tras la aplicación de la transformada de wavelet, para seleccionar un umbral determinado, de tal forma que los valores de los coeficientes transformados menores que dicho umbral se igualan a cero. Este método supuso un gran avance en la eliminación de ruido en los datos, ya que el filtrado del ruido se realiza sin llevar a cabo un proceso de suavizado en la estructura que delimita los bordes, pone de manifiesto los detalles importantes y muestra una señal clara y limpia. Esta misma técnica se ha aplicado en numerosos campos como la eliminación de ruido en imágenes mediante un umbral determinado [Chang y Vetterli, 1997] o un umbral adaptativo a la zona de la imagen [Chang *et al.*, 1997].

### Detección de comportamientos autosimilares en series de tiempo

En 1993, Scargle *et al.* [1993] y la NASA investigaron las oscilaciones periódicas y el ruido a bajas frecuencias de rayos X astronómicos, ya que el sistema estelar Sco X-1 estaba determinado por el mismo fenómeno físico. Los investigadores detectaron que la luminosidad de Sco X-1 se comportaba de forma autosimilar, de forma que las funciones wavelet pueden representar esta información eficientemente a diferentes escalas o resoluciones. Scargle diseñó un escalegrama de



series de tiempo como la media de los cuadrados de los coeficientes wavelet a una resolución determinada para investigar las series de tiempo. Las funciones wavelet mostraban la mayor parte de la información que ofrecía la transformada de Fourier y además, presentaban la función en el plano de la frecuencia.

### Recuperación de imágenes basadas en el contexto

La incesante creación y desarrollo de colecciones de imágenes *on-line* implican la necesidad de la creación de métodos que puedan buscar y recuperar imágenes a partir de una imagen determinada. Wang *et al.* [1997] diseñaron un método basado en la transformada de wavelet 2D, el cual aplicaba la transformada, usando la función wavelet madre de Daubechies de ocho coeficientes, sobre cada una de las componentes de color de las imágenes a guardar en la base de datos. A continuación, se almacenaba un vector de características de cada una de las imágenes con los coeficientes de la sub-banda más baja o imagen de referencia y la varianza de dicha sub-banda, para cada una de las componentes de color. El proceso de recuperación de las imágenes se dividía en dos partes: la primera, calculaba la varianza de la imagen a consultar y realizaba una selección entre las imágenes almacenadas en la base de datos, mientras que la segunda, aplicaba la transformada de wavelet 2D sobre la imagen a consultar para comparar con los vectores de características de las imágenes seleccionadas en la primera fase y obtener una selección de las imágenes que tengan una mayor similitud con respecto a la original. El método propuesto obtiene mejores resultados que los algoritmos de recuperación tradicionales basados en la distribución del color de la imagen. Dicha técnica es capaz de recuperar de una base de datos de propósito general compuesta por 10000 imágenes, las 100 imágenes con mayor parecido a una imagen determinada en menos de 3,3 segundos en una estación de trabajo SUN SPARC-20.

De forma similar, Brambilla *et al.* [1999] propusieron una estrategia eficiente para que un usuario pudiera realizar consultas visuales para recuperar un conjunto de imágenes de una base de datos, que cumplieran una serie de criterios sin la necesidad de expresarlos semánticamente. Para ello, aplicaban la transformada de wavelet 2D, con la función wavelet madre Haar, a cada una de las componentes de color de una imagen (RGB). A continuación, los 128 coeficientes mayores en magnitud se cuantifican con +1 o -1 dependiendo de si son positivos o negativos respectivamente, y el resto de coeficientes se eliminan. Por lo tanto, una imagen se identifica mediante una signatura compuesta de  $3 \times 128$  coeficientes cuantificados, la media del color y el ratio del aspecto de la imagen (ratio entre las dimensiones de la imagen). Las imágenes se recuperan mediante un proceso de comparación de la signatura de la imagen a consultar con las signaturas de las imágenes almacenadas en la base de datos. Las imágenes recuperadas o extraídas de la base

de datos se clasifican de mayor a menor similitud según una función de precisión. El sistema tiene un comportamiento mejor que un sistema de recuperación basado en la distribución del color sobre una colección de imágenes de textiles antiguos.

### **Diseño de un sistema de seguridad basado en un reconocimiento facial**

Martínez [2002] llevo a cabo la simulación de un sistema de seguridad para controlar el acceso de personas a una urbanización. La identificación y obtención de los rasgos invariables o características básicas de los rostros de cada una de las personas se realiza mediante la transformada de wavelet. Dadas las imágenes de los rostros de una serie de personas que pueden acceder a la urbanización, se les aplica la transformada de wavelet 2D y se almacenan los coeficientes resultantes. Estos coeficientes conforman los rasgos característicos de cada una de las personas que pueden acceder a la urbanización y componen el sistema de almacenamiento. Cuando una persona quiera acceder a la urbanización, el sistema de identificación captura la imagen del rostro de la persona, le aplica la transformada de wavelet 2D y realiza una comparación con los coeficientes de las distintas personas que tiene en el sistema de almacenamiento. Se calcula el PSNR para cada comparación posible, por lo que se obtienen tantos PSNR como individuos tengamos almacenados, siendo identificado el individuo correspondiente con el mayor PSNR. Si la diferencia entre el PSNR de la persona identificada y la media de todos los PSNR's calculados es mayor que 1 (valor escogido tras exhaustivos análisis de diversas configuraciones) se llega a la conclusión de no permitir el acceso a la persona en cuestión. Para la validación del sistema, se utilizaron diecisiete individuos, quince de ellos con diferentes fotos y en distintos estados registrados en el sistema. Las dos personas restantes no estaban registradas y determinan si el sistema es capaz de rechazarlas o no. Las pruebas se llevaron a cabo con las funciones wavelet madre Haar, Daub-4, Daub-8, Daub 9/7 y una función biortogonal, y para dos, tres y cuatro iteraciones de la transformada de wavelet 2D. La configuración ideal se obtiene con tres iteraciones de la función wavelet Haar. Bajo estas condiciones, el sistema se comporta de forma ideal, ya que no se rechaza a ninguna persona que conserva el mismo aspecto con el que fue registrada en el sistema, ni se permite el acceso a ninguna persona no identificada por el sistema. Por el contrario, en el 29,09 % de los casos el sistema no reconoce correctamente a individuos que se encuentran registrados pero que han cambiado su aspecto de forma significativa, por ejemplo incluyen en su rostro gafas de sol o un gorro, con respecto a las imágenes que tienen almacenadas en el sistema. Sin embargo, este último hecho no es preocupante, ya que todos estos individuos pueden volver a registrarse en la base de datos del sistema.

## 2.6. Conclusiones

En este capítulo se ha realizado una descripción del funcionamiento de las principales técnicas de compresión de imágenes y secuencias de vídeo que vamos a utilizar en los siguientes capítulos. El desarrollo de la información multimedia viene determinado por la eficacia de los métodos de compresión que permiten reducir el espacio de almacenamiento y el tiempo de transmisión de la información.

En lo que se refiere a las técnicas de compresión, nos hemos centrado en la descripción pormenorizada del funcionamiento de un codificador basado en una transformada matemática. Los tres procesos básicos que componen un codificador de este tipo son:

1. Transformación de los datos.
2. Cuantificación de los coeficientes transformados.
3. Codificación entrópica de los coeficientes cuantificados.

Con respecto a la transformación de los datos, hemos descrito la transformada de Fourier continua y discreta para determinar, a partir de la misma, la transformada discreta del coseno. A continuación, se ha presentado la transformada de wavelet continua y discreta como una alternativa para representar de forma eficiente las señales no estacionarias. A partir de la transformada discreta de wavelet, se presentan los algoritmos para aplicar la transformada de wavelet unidimensional (1D-FWT), bidimensional (2D-FWT) y tridimensional (3D-FWT). Hemos hecho especial hincapié en el uso de dicha transformada y en sus fundamentos matemáticos, ya que será uno de los pilares básicos para el desarrollo de un codificador de vídeo en los capítulos siguientes.

En cuanto al proceso de cuantificación, en el cual se realiza una discretización de los coeficientes que se obtienen como resultado de la aplicación de la transformada matemática, hemos descrito las principales formas de cuantificación existentes: uniforme, no uniforme y adaptativa. Dentro de esta última se ha descrito la cuantificación directa, inversa y por selección.

La codificación entrópica permite aumentar la tasa de compresión sin afectar a la distorsión de la señal reconstruida. A partir de la codificación de Shanon-Fano, hemos definido la codificación Huffman, la cual se basa en la sustitución de cada uno de los símbolos por un código de bits específico, que se obtiene a partir de la probabilidad de aparición de cada uno de los símbolos. La codificación aritmética mejora la codificación Huffman, representando una secuencia de símbolos mediante un número real de simple precisión. Por último, se ha introducido el funcionamiento de

la codificación *Run-Length* basada en la representación eficiente de símbolos repetidos de forma consecutiva.

Los codificadores basados en una transformada han dado lugar a una serie de estándares para la compresión de imágenes (JPEG, JPEG-LS y JPEG-2000) y secuencias de vídeo (MPEG-1, MPEG-2, MPEG-4, H.261 y H.263). Las principales características de los mencionados estándares y la descripción de las distintas fases se han llevado a cabo en la sección 2.4.1. Al mismo tiempo se han desarrollado otra serie de métodos basados en la transformada de wavelet, que se han introducido en la sección 2.4.2.

Por último, y para denotar la importancia de la transformada de wavelet en los últimos años, se han descrito otros usos de dicha transformada (aparte de la compresión de imágenes y secuencias de vídeo), entre los que se encuentran la eliminación de ruido, la detección de comportamientos autosimilares en series de tiempo, la recuperación de imágenes basadas en el contexto y el diseño de un sistema de seguridad basado en un reconocimiento facial.

## Capítulo 3

### Codificador de vídeo médico basado en la transformada de wavelet 3D

---

#### 3.1. Introducción

El gran aumento del volumen de imágenes y vídeo médico en los hospitales, así como las características peculiares de este tipo de información ha provocado que los hospitales actuales requieran unos grandes espacios de almacenamiento. Por otro lado, el desarrollo de la telemedicina viene determinado por la disponibilidad de la información en tiempo real, ya que puede estar en juego la vida de una persona. Por ejemplo, en la telediagnosís un doctor necesita consultar a otro colega, que se encuentra en otro hospital de cualquier lugar del mundo, sobre un determinado diagnóstico, por lo que es necesario poder transmitir la información en tiempo real. Por todo ello, las técnicas de compresión son necesarias para reducir tanto la cantidad de información almacenada como los datos a transmitir. Como hemos visto en el capítulo anterior, en los últimos años se ha desarrollado el conocimiento de la transformada de wavelet y se ha utilizado en diferentes ámbitos, entre los que se encuentra la compresión de imágenes y vídeo.

En este capítulo, presentamos y desarrollamos un nuevo codificador de vídeo con pérdida de información basado en la transformada de wavelet 3D. El objetivo principal es doble y trata de conseguir, por un lado, una gran tasa de compresión y, por otro lado, una calidad excelente en el vídeo reconstruido. El término calidad excelente, desde un punto de vista médico, se define como el proceso de compresión en el cual el vídeo reconstruido es de tal calidad que los médicos no pueden apreciar diferencias con el original.

En las figuras 3.1 y 3.2, podemos observar las diferentes fases o procesos en los que se descompone un codificador basado en la transformada de wavelet 3D y el correspondiente decodificador.

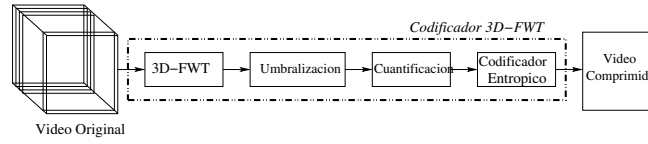


Figura 3.1: Procesos de un codificador basado en la transformada de wavelet 3D (3D-FWT)

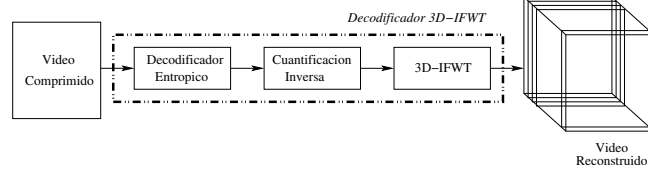


Figura 3.2: Procesos de un decodificador basado en la transformada de wavelet 3D inversa (3D-IFWT)

Con la construcción de un codificador de vídeo propio, desarrollaremos cada una de las partes del codificador y del decodificador: transformada de wavelet 3D directa e inversa, cuantificación y descuantificación, umbralización, codificación y decodificación entrópica. De esta forma, analizaremos y comprenderemos cada una de las partes del codificador, y podremos realizar propuestas que mejoren las técnicas existentes.

Comenzando con la parte de la transformada de wavelet 3D, analizaremos varios factores que influyen tanto en la tasa de compresión final como en la calidad del vídeo reconstruido. En este caso, evaluaremos el comportamiento de diferentes funciones wavelet, el número de iteraciones que se aplica la transformada en cada una de las dimensiones o el orden a seguir para aplicar la transformada sobre las diferentes dimensiones. A partir de dicho estudio, estableceremos una combinación de los factores anteriores con el objetivo de obtener la mejor relación entre la tasa de compresión y la calidad del vídeo reconstruido.

En el proceso de umbralización, propondremos el uso de dos técnicas. La primera utiliza la técnica del percentil y consiste en eliminar (poner a cero) todos los coeficientes wavelet que sean menores que un percentil determinado, mientras que la segunda descarta de cada uno de los coeficientes wavelet un número determinado de bits menos significativos.

En cuanto al proceso de cuantificación, desarrollaremos un cuantificador que asigna un número de bits a cada coeficiente wavelet, dependiendo de la capa a la que pertenece dicho coeficiente.

En el codificador entrópico, propondremos la utilización de un *Run-Length* seguido de un codificador de Huffman, con el objetivo de aumentar la tasa de compresión y mantener la calidad del vídeo reconstruido. El *Run-Length* se aplicará sobre la representación binaria de los coeficientes

wavelet cuantificados, mientras que el codificador de Huffman intervendrá sobre el resultado generado por el *Run-Length*. Ambos codificadores actuarán de forma independiente sobre cada una de las imágenes que componen la secuencia de vídeo.

Una vez desarrollado el codificador basado en la transformada de wavelet 3D, estudiaremos y analizaremos sus limitaciones para realizar varias propuestas centradas en el cuantificador y en el codificador entrópico con el objeto de aumentar la tasa de compresión y mantener la calidad del vídeo reconstruido del codificador original. Por todo ello, desarrollaremos un *Run-Length* binario 3D inteligente que aproveche de forma eficiente las ventajas de la aplicación de la transformada 3D para conseguir una mayor tasa de compresión. Dicha propuesta vendrá acompañada de la adecuación del cuantificador a la utilización de la transformada de wavelet 3D, ya que la secuencia de vídeo original se descompone en varios sub-cubos, a los que se le asignará un número diferente de bits en función de la magnitud de los coeficientes que contengan.

Relacionado con el desarrollo del *Run-Length* binario 3D inteligente, utilizaremos la codificación hexadecimal para representar la longitud de las cadenas de ceros y unos consecutivos sin ninguna limitación para el *Run-Length*. Además, y basándonos en el estudio de la codificación hexadecimal, propondremos dos técnicas para mejorar la representación de las cadenas desde uno hasta siete ceros consecutivos y las cadenas de unos seguidos.

Para mejorar la codificación entrópica, reemplazaremos el codificador de Huffman por un codificador aritmético de 16 símbolos. El número de símbolos viene determinado por el uso previo de la codificación hexadecimal. En este caso, en vez de sustituir cada símbolo de entrada por un código específico, se sustituye una secuencia de símbolos de entrada por un número en punto flotante.

Una vez desarrolladas las distintas propuestas, se evalúan los resultados del codificador base, analizando las diferentes técnicas propuestas en cada una de las partes del codificador para determinar la mejor relación entre la tasa de compresión y la calidad del vídeo reconstruido. A continuación, se analiza el rendimiento del codificador integrado con las mejoras propuestas para el cuantificador y el codificador entrópico y se realiza una comparación con respecto al estándar de compresión de vídeo MPEG2 y al codificador *zerotree* (EZW) de Shapiro [1993].

## 3.2. Trabajo relacionado

### 3.2.1. Codificadores basados en la transformada de wavelet 2D

En los últimos años se han desarrollado varios codificadores basados en la transformada de wavelet 2D para la compresión de imágenes y vídeo, desde diferentes perspectivas.

Antonini *et al.* [1992] proponen un esquema basado en la transformada de wavelet 2D, que tiene en cuenta algunas características del sistema de visión humana para la compresión de imágenes. Se desarrolla un algoritmo piramidal basado en la transformada, donde la imagen se descompone en diferentes escalas, a las que se aplica un cuantificador basado en vectores que utiliza un libro de códigos (codebook) de varias resoluciones. Finalmente, para codificar los coeficientes wavelet se propone un procedimiento de asignación de bits que depende de la importancia del citado coeficiente. Por ejemplo, el cuantificador asume que los detalles con frecuencias altas son menos visibles para el sistema de visión humano, por lo que le asigna un menor número de bits. Se utiliza un esquema de transmisión progresivo, que permite recibir las imágenes a la resolución deseada por el receptor con el objetivo de reconocer la imagen en el mínimo tiempo y con el menor coste. Además, se proponen varias funciones wavelet madre: Spline 9 – 3, Spline 9 – 7 (denominada Daubechies 9/7 (Daub-9/7)), y Coiflet 5 – 7.

Al mismo tiempo y de la misma forma, Lewis y Knowles [1992] utilizan la transformada de wavelet 2D con la función wavelet madre Daub-4, para descomponer una imagen en diferentes resoluciones. El algoritmo de compresión consiste en enviar todos los coeficientes de la sub-banda más baja y determinar cuales se envían del resto de sub-bandas, según un algoritmo basado en la restricción de la localidad espacial de Marr. Dicho algoritmo establece que si un determinado coeficiente con una situación espacial es importante, es de esperar que los coeficientes con la misma situación espacial en las distintas sub-bandas puedan superar un umbral determinado y ser transmitidos. A continuación, los coeficientes wavelet se cuantifican de forma jerárquica e individual de acuerdo a una estimación de la sensibilidad del sistema de visión humano. Finalmente, se utiliza el codificador de Huffman para obtener una mayor tasa de compresión manteniendo el nivel de calidad. Además, el algoritmo se puede implementar mediante hardware para obtener un rendimiento en tiempo real.

Posteriormente, Shapiro [1993] desarrolla un codificador empaquetado (EZW), basado en la transformada de wavelet y en la construcción de árboles de ceros (*zerotrees*). El algoritmo establece dos propiedades importantes para los coeficientes wavelet:

1. La energía en las diferentes sub-bandas disminuye a medida que disminuye la escala o la



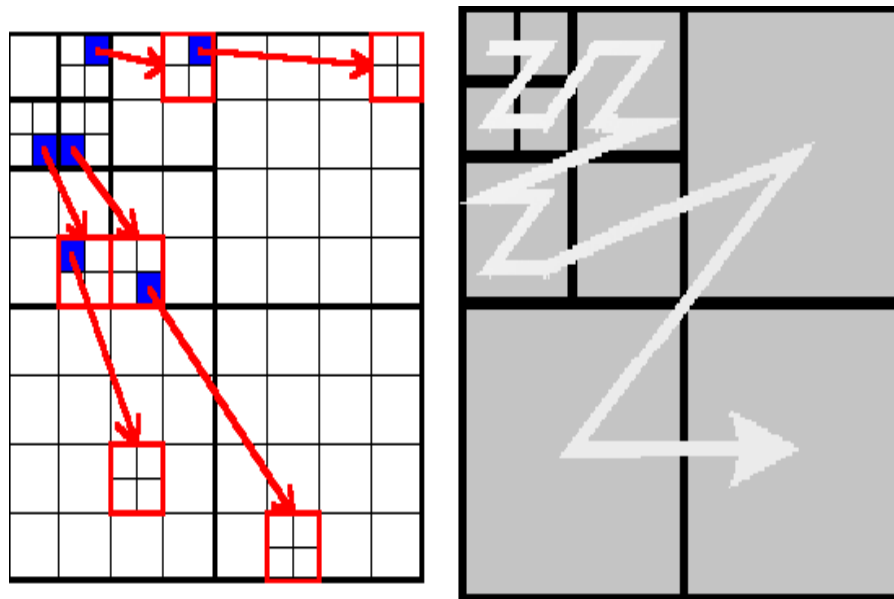


Figura 3.3: Relaciones entre coeficientes wavelet de diferentes sub-bandas (Izquierda). Orden de búsqueda en el proceso de codificación *zerotree* (Derecha).

resolución, por lo que los coeficientes wavelet serán menores en las sub-bandas pasa-alta que en las pasa-baja. Por lo tanto, el modelo wavelet se adapta perfectamente a la codificación progresiva, tal y como establecieron anteriormente Antonini *et al.* [1992].

2. El orden de importancia de los coeficientes wavelet viene determinado por su mayor o menor magnitud en valor absoluto. Por lo tanto, los coeficientes wavelet negativos y con valores absolutos grandes son más importantes que los coeficientes positivos y pequeños en magnitud.

En el proceso de codificación se aplican las propiedades anteriores, de tal forma que en cada una de las iteraciones se elige un umbral que se compara con cada uno de los coeficientes wavelet para determinar si el coeficiente es mayor o menor que el umbral. Si el coeficiente es mayor se codifica y se elimina de la imagen, mientras que si es menor se conserva para la siguiente iteración. Una vez que se han comparado todos los coeficientes con el umbral establecido, se disminuye el umbral y se analiza de nuevo la imagen para añadir nuevos detalles. El proceso se repite hasta que se codifiquen todos los coeficientes o se cumpla un determinado criterio relacionado con la tasa de compresión. Además, el algoritmo aprovecha la situación espacial de un determinado coeficiente, de tal forma que cada coeficiente en una sub-banda pasa-baja tiene cuatro descendientes en la siguiente sub-banda pasa-alta como podemos observar en la figura 3.3 (parte izquierda). Por

lo tanto, se define un árbol de ceros (*zerotree*) como un árbol con cuatro hijos por nodo en el que los valores de todos los nodos son iguales o menores que la raíz. El codificador explota el concepto anterior, basándose en que existe una gran probabilidad de que todos los coeficientes que forman un árbol sea menores que un determinado umbral. Si la raíz de un árbol es menor que el umbral, se codifica el árbol entero mediante un solo símbolo denominado *zerotree* (T). Además, los coeficientes se codifican siguiendo un orden pre-establecido para codificar su posición, como podemos observar en la figura 3.3 (parte derecha). Este hecho permite la codificación de muchas posiciones de forma implícita, lo que influye notablemente en la tasa de compresión final. Tras el proceso de cuantificación, se lleva a cabo un codificador aritmético implementado con los siguientes cuatro símbolos:

- Positivo (P), si el coeficiente es mayor que el umbral.
- Negativo (N), si el coeficiente es menor que el umbral.
- Zerotree (T), si el coeficiente es la raíz de un *zerotree*.
- Cero aislado (Z), si el coeficiente es menor que el umbral pero no es la raíz de un *zerotree*.

Posteriormente, Hilton *et al.* [1994] proponen usar la transformada de wavelet 2D para la compresión de vídeo. Para ello, aprovecha las redundancias temporales existentes entre las diferentes imágenes consecutivas que compone una secuencia. El codificador calcula la diferencia entre dos pixels que tienen la misma situación espacial en dos imágenes consecutivas, aplica la transformada, umbraliza y analiza aquellas regiones que necesitan realizar la transformada inversa para reconstruir la imagen original, y obtener de esta forma una transformada inversa muy rápida. Los resultados obtenidos sobre el vídeo de *Miss América* compuesto de 30 imágenes, alcanzan una tasa de compresión de 18 : 1 y un valor medio del PSNR de 35,06 dB.

Basándose en las propiedades del codificador EZW, Said y Pearlman [1996b] desarrollaron un algoritmo basado en la división de conjuntos en árboles jerárquicos (SPIHT). Se trata de un esquema de transmisión progresivo que incorpora el concepto de ordenación de los coeficientes wavelet por magnitud, ya que los coeficientes mayores contienen una mayor cantidad de información que los menores, y la transmisión de los coeficientes por planos de bits se realiza según un orden de mayor a menor bits significativos. Dicha transmisión por planos de bits determina que dado un conjunto de  $n$  coeficientes de  $m$  bits, se transmiten primero los  $n$  bits más significativos de los  $m$  coeficientes; a continuación se transmiten los  $n$  bits que se encuentran en la posición  $m - 1$  (segundos bits mas significativos), y así sucesivamente hasta la transmisión de los bits menos significativos de cada uno de los coeficientes. El algoritmo de ordenación divide el conjunto

de pixels original en conjuntos de árboles jerárquicos que dependen de la situación espacial de los coeficientes, tal y como se hace en el codificador EZW. El proceso de umbralización utiliza las distintas potencias de dos, de mayor a menor, como umbral para determinar si un conjunto es significativo o no. Si el conjunto es significativo, se divide en nuevos subconjuntos y se aplica de nuevo el test de umbralización, mientras que si el conjunto no es significativo se descartan todos los coeficientes que pertenecen al citado conjunto. El proceso continua hasta que se alcanza una tasa de compresión o una calidad determinada. Se propone un cuantificador escalar uniforme que obtiene un mayor rendimiento del esperado a priori, debido a la ordenación previa de la información. En la codificación entrópica se utiliza un codificador aritmético adaptativo.

Estos mismos autores, Said y Pearlman [1996a] propusieron la transformada S+P para la compresión de imágenes sin pérdida de información y con un comportamiento similar a la wavelet. Dicha transformada se basa en el cálculo de operaciones mediante sumas de enteros y desplazamiento de bits. En la codificación entrópica, los coeficientes enteros se transmiten divididos en tres partes: el número del conjunto al que pertenece la magnitud, el signo y la diferencia de magnitud entre la actual y la menor magnitud de un conjunto de pixels transformados, basándose en el estándar JPEG. Se utiliza un codificador de Huffman o aritmético, dependiendo de los recursos hardware disponibles y de la velocidad de ejecución deseada. El rendimiento, en cuanto a la tasa de compresión, es totalmente competitivo con los resultados obtenidos por las técnicas más eficientes de compresión de imágenes sin pérdida de información.

Posteriormente, Buccigrossi y Simoncelli [1997] desarrollaron un codificador basado en la transformada de wavelet 2D y en un modelo de probabilidad condicional entre los coeficientes de las diferentes sub-bandas (EPWIC). En el codificador se establece que los coeficientes con mayor magnitud se producen en las mismas situaciones espaciales de las diferentes sub-bandas. Cuando la magnitud del coeficiente padre es grande, el valor esperado del coeficiente hijo se corresponde linealmente con la magnitud del padre, mientras que cuando la magnitud del coeficiente padre es pequeña, la magnitud del hijo es totalmente independiente de la del padre. Se utilizan funciones wavelet simétricas de 9 coeficientes. El cuantificador asigna diez bits a cada uno de los coeficientes que se envían al receptor, mediante planos de bits. En el codificador entrópico, se utiliza un codificador aritmético no adaptativo y un *Run-Length* por planos de bits. EPWIC obtiene mejores resultados que EZW sobre seis imágenes (*Baboon*, *Boats*, *Goldhill*, *Lena*, *Peppers* y una Tomografía Computacional), para tasas de compresión grandes. Para tasas de compresión pequeñas se comporta de forma ineficiente, al igual que el codificador EZW.

Al mismo tiempo, Rabinovitch [1997] presenta un esquema basado en la transformada de wavelet 2D y en características del sistema de visión humano como la sensibilidad o la importancia

de los bordes para la compresión de imágenes sin pérdidas. Dada una imagen, se selecciona la función wavelet madre que obtenga un número menor de coeficientes distintos de cero de entre un grupo de funciones wavelet. En la cuantificación, se codifica primero la sub-banda pasa-baja o imagen de referencia, y a continuación el resto de coeficientes. Se utiliza un método de modulación (DPCM) que obtiene la diferencia entre los pixels adyacentes. El codificador entrópico utiliza un *Run-Length* basado en ceros y en la localidad espacial de los coeficientes de las diferentes sub-bandas y un codificador de Huffman.

Simultáneamente Chen *et al.* [1997] presentan un esquema no empaquetado basado en la transformada de wavelet 2D. Se propone un cuantificador uniforme para todos los coeficientes, que se descomponen en el bit más significativo y en el resto de bits o bits residuales para realizar la codificación entrópica. En esta última fase, se utiliza un codificador aritmético adaptativo que se realiza en distinta dirección según la sub-banda, determinando en cada momento si el coeficiente es significativo o no.

Por otro lado, Wang y Kuo [1997] desarrollaron un codificador wavelet empaquetado con umbrales diferentes para cada una de las sub-bandas (MTWC), obteniendo para *Lena* con la función Daub-9/7 [Antonini *et al.*, 1992] mejores resultados que los codificadores LZC<sup>1</sup> [Taubman y Zakhor, 1994] y SPIHT.

Calderbank *et al.* [1998] proponen funciones wavelet representadas mediante enteros para conseguir una reconstrucción exacta de las imágenes. Se presentan dos esquemas diferentes para la construcción de funciones wavelet usando enteros. Los resultados demuestran que los filtros desarrollados permiten la codificación progresiva y sin pérdidas de las imágenes.

Taubman [2000] desarrolló un algoritmo basado en el concepto de compresión escalable que tiene como predecesores a varios codificadores ya comentados como EZW, SPIHT y LZC [Taubman y Zakhor, 1994]. El esquema propone la codificación de bloques empaquetados e independientes y una umbralización optimizada de los flujos de bits empaquetados (EBCOT). La compresión escalable consiste en la creación de flujos de bits que contienen subconjuntos empaquetados. Cada subconjunto representa la imagen original con una tasa de compresión eficiente, una resolución inferior o un grado de distorsión superior a la imagen original. Se utiliza un cuantificador mediante planos de bits, desde los bits más significativos a los menos significativos y un codificador aritmético como codificador entrópico. El algoritmo EBCOT fue elegido para el estándar de compresión de imágenes JPEG-2000 [Bolie, 2000][Marcellin *et al.*, 2000], ver sección 2.4.1.3, con algunas modificaciones en el codificador entrópico, donde se introduce un codificador aritmético denominado codificador MQ [Ueno *et al.*, 1998] que evita el cálculo de multiplicaciones y divisio-

---

<sup>1</sup>El codificador LZC se basa en la transformada de wavelet 3D y se explicará en la sección 3.2.2.

Lena (512x512 pixels - 8 bpp)			
Codificadores	0.25 bpp	0.5 bpp	1 bpp
JPEG [Wallace, 1991]	30.40	34.70	37.84
Antonini <i>et al.</i> [1992]	29.11 (0.21)	30.85 (0.37)	
Lewis y Knowles [1992]		34.03	
Shapiro [1993] (EZW)	33.17	36.28	39.55
Said y Pearlman [1996b] (SPIHT)	34.15	37.25	40.46
Rabinovitch [1997]			37.83 (0.94)
Chen <i>et al.</i> [1997]	34.49	37.53	40.64
JPEG-2000 [Marcellin <i>et al.</i> , 2000]	34.03	37.33	40.43

Tabla 3.1: Calidad en PSNR de la imagen *Lena* para distintos codificadores y tasas de compresión. Los datos entre paréntesis indican la tasa de compresión real de algunos resultados que se han aproximado a las tasas de compresión de la segunda fila de la tabla.

nes. De esta forma, se obtiene un codificador entrópico que es un 40 % más rápido en tiempo de ejecución que el original, y se consigue una ganancia a nivel de calidad de 0,15 dB con respecto a EBCOT.

Saha y Vemuri [1999][2000] propusieron una selección dinámica de la función wavelet madre para cada imagen, en función de la tasa de compresión más elevada. Los resultados ponen de manifiesto diferencias, a nivel de calidad, de hasta 6dB en función de la función wavelet elegida y de la naturaleza de la imagen.

En la tabla 3.1, podemos observar una comparativa de algunos de los codificadores descritos anteriormente, a nivel de calidad (PSNR) de la imagen *Lena* (512x512 pixels y 8 bits por pixels (8 bpp)), y para diferentes tasas de compresión (0,25 bpp, 0,5 bpp y 1 bpp).

En primer lugar, podemos observar que los resultados obtenidos por el codificador de Antonini *et al.* [1992] son inferiores, aunque muy cercanos, al estándar de compresión de imágenes JPEG [Wallace, 1991], ver sección 2.4.1.1. El codificador propuesto por Lewis y Knowles [1992] supera al de Antonini *et al.* [1992] y obtiene resultados muy similares a JPEG. Dichos resultados verifican la validez de los métodos de compresión basados en la transformada de wavelet 2D y promueven la creación de nuevos métodos.

El codificador EZW propuesto por Shapiro [1993] utiliza filtros piramidales QMF de 9 coeficientes, obtiene una ganancia de entre 2 y 3 dB para diferentes tasas de compresión, con respecto

al estándar JPEG. A partir del EZW, el codificador SPIHT [Said y Pearlman, 1996b] aplica cinco veces la transformada de wavelet con la función madre Daub-9/7 [Antonini *et al.*, 1992], obtiene mejores resultados que su predecesor, y por tanto, supera al estándar JPEG. De la misma forma, el codificador propuesto por Chen *et al.* [1997] mejora los resultados de JPEG, EZW y SPIHT, utilizando la función wavelet madre Daub-9/7.

Por otro lado, el esquema presentado por Rabinovitch [1997] también obtiene mejores resultados que el estándar JPEG. En este caso, la calidad se mide mediante el PSNR y la evaluación subjetiva de quince observadores.

Finalmente, el nuevo estándar de compresión de imágenes JPGE-2000, que utiliza el algoritmo EBCOT, mejora los resultados del codificador SPIHT, realizando cinco iteraciones de la transformada de wavelet con la función madre Daub-9/7 [Antonini *et al.*, 1992]. En cuanto a la calidad, las imágenes reconstruidas con EBCOT presentan un menor ruido en los bordes, las texturas presentan una mejor definición y se mantienen algunos detalles eliminados por SPIHT.

### 3.2.2. Codificadores basados en la transformada de wavelet 3D

Basándose en los codificadores desarrollados utilizando la transformada de wavelet 2D, Muraki [1992][1995] introdujo la idea de utilizar la transformada de wavelet 3D para aproximar eficientemente imágenes volumétricas médicas 3D como Tomografía Computacional (CT). Esta idea la desarrollaron Ihm y Park [1998] e Ihm *et al.* [1998] con el objetivo de conseguir un compromiso entre una tasa de compresión alta, una rápida decodificación, un acceso rápido y una mínima degradación de la calidad de la imagen. Para ello, la imagen volumétrica se divide en bloques de  $16 \times 16 \times 16$  pixels, a los que se le aplica la transformada de wavelet 3D dos veces con la función wavelet madre Haar. El cuantificador divide los coeficientes y utiliza una estructura de datos que tiene en cuenta la coherencia espacial de los coeficientes. El codificador entrópico utiliza un codificador de Huffman o aritmético. La técnica se puede aplicar a imágenes volumétricas en color de 24 bits (RGB), es decir, 8 bits para cada uno de los colores primarios. La técnica desarrollada consigue ratios de compresión de hasta 28 : 1 con una buena calidad para CT.

Considerando el tiempo como una de las tres dimensiones anteriores, se pueden transferir las ideas anteriores a la investigación en la compresión de vídeo. Chen y Pearlman [1996] desarrollaron un codificador de vídeo basado en la transformada de wavelet 3D. La transformada se aplica cuatro veces en el tiempo y en el espacio (x,y). Se utiliza como función wavelet madre la Daub-9/7 [Antonini *et al.*, 1992]. Los coeficientes transformados se codifican mediante un codificador [Said y Pearlman, 1993] basado en EZW [Shapiro, 1993] y desarrollado por los mismos autores. El

codificador EZW para dos dimensiones se generaliza para tres dimensiones y se utiliza un codificador aritmético como codificador entrópico. Los resultados de codificación para dos secuencias de vídeo, tenis y fútbol, se comparan con el estándar MPEG2, mostrando una mejor calidad media (PSNR) para 0,3 bpp y 1,0 bpp, aunque con algunas manchas en ciertas regiones del vídeo reconstruido. Los vídeos reconstruidos con el estándar MPEG2 presentan efectos de bloque que desde el punto de vista del sistema visual humano producen una mayor degradación de la calidad de las imágenes que las manchas. Además, el estándar MPEG2 introduce una mayor complejidad debido a la complejidad de la compensación de movimiento (*motion compensation*).

Posteriormente, Kim y Pearlman [1997] mejoraron los resultados anteriores aplicando el algoritmo SPIHT [Said y Pearlman, 1996b] a la compresión de vídeo, es decir, utilizando la transformada de wavelet 3D y las mejoras introducidas por el algoritmo SPIHT para tres dimensiones. Los resultados obtenidos sobre las secuencias de vídeo, tenis y fútbol, superan al codificador anterior [Chen y Pearlman, 1996] y al estándar MPEG2, a nivel de calidad media (PSNR) y tasas de compresión de 0,3 bpp y 1,0 bpp. Basándose en este algoritmo, Kim y Pearlman [1998] realizaron modificaciones sobre el 3D-SPIHT, para permitir una mayor flexibilidad en la elección del número de veces que se aplica la transformada de wavelet en cada una de las dimensiones, mediante la introducción de árboles desbalanceados, y la posibilidad de codificar vídeos en color. La primera modificación permite seleccionar un número distinto de iteraciones de la transformada wavelet en el tiempo y en el espacio, decidiendo en cada momento cual es la mejor relación entre rendimiento, retraso de la codificación y requerimientos de memoria. Con respecto a la segunda modificación, un esquema sencillo hubiera consistido en aplicar el 3D-SPIHT a cada una de las componentes del color. Sin embargo, dicha opción supone una reserva de bits entre los diferentes componentes de color, una pérdida en el control de la tasa de compresión, y que el decodificador tenga que esperar hasta que lleguen completamente los distintos flujos de bits para reconstruir el vídeo. Por todo ello, se propone generar un flujo de bits mixto de tal forma que podamos detenernos en cualquier punto del flujo de bits para reconstruir el vídeo con la mejor calidad y con una tasa de compresión determinada. Las secuencias de vídeo en color, tenis y fútbol, se codifican con diferentes tamaños de grupos de imágenes (GOF), 4, 8 y 16 imágenes, y se comparan con los estándares MPEG2 y H.263. En general, un mayor tamaño del GOF implica una mejor adaptación a la reconstrucción de la escena. Para un GOF de 16 imágenes, el 3D-SPIHT en color se comporta mejor que el estándar MPEG2, mientras que supera a H.263 en las componentes U y V, y se encuentra ligeramente por debajo en la componente Y.

Posteriormente, Bilgin *et al.* [1998] propusieron un algoritmo para la compresión de imágenes volumétricas médicas sin pérdidas, basado en la transformada de wavelet 3D con enteros y el

codificador EZW. El algoritmo explota las redundancias en las tres dimensiones y generaliza el codificador EZW para tres dimensiones como en [Chen y Pearlman, 1996]. Además, se utiliza un codificador aritmético adaptativo y basado en el contexto que explota las dependencias espaciales y jerárquicas entre los diferentes símbolos para mejorar el rendimiento del codificador EZW. Los resultados se presentan para dos iteraciones de la transformada de wavelet 3D en cada una de las dimensiones. Se utiliza como función wavelet madre una función entera diádica  $(2, 4)$ . Las imágenes volumétricas de Tomografía Computacional (CT) y Resonancia Magnética (MR) se procesan mediante grupos consecutivos de 16 imágenes. El codificador propuesto supera los resultados obtenidos por las mejores técnicas de compresión basadas en dos dimensiones, como el estándar de compresión de imágenes sin pérdidas en escala de grises LOCO-I [Weinberger *et al.*, 1996] y el codificador 2D-SPIHT [Said y Pearlman, 1996b].

En el mismo tema, Xiong *et al.* [1998], y posteriormente, Kim y Pearlman [1999], aplican el algoritmo 3D-SPIHT a la compresión de imágenes volumétricas médicas con pérdidas y sin pérdidas. Se trata de una analogía 3D del sistema progresivo con pérdidas y sin pérdidas introducido en [Said y Pearlman, 1996a]. Establecen que los resultados con pérdidas, determinados anteriormente por Bilgin *et al.* [1998] no son correctos totalmente, debido a que no se utiliza una transformada unitaria, por lo que el error de cuantificación en el dominio wavelet no es igual al error cuadrático medio en el dominio del tiempo o del espacio. Se utiliza la transformada de wavelet 3D basada en enteros, donde los coeficientes de las funciones wavelet madre son las potencias de dos. Además, la unidad de código es menor que en Bilgin *et al.* [1998], por lo que se produce un consumo menor de memoria. Los tests se realizan sobre las mismas imágenes volumétricas utilizadas por Bilgin *et al.* [1998], obteniendo el nuevo 3D-SPIHT sin pérdidas un mejor rendimiento que el codificador de Bilgin *et al.* [1998], y por tanto que las técnicas basadas en dos dimensiones.

Por otro lado, Wang *et al.* [1999] introdujeron el concepto de compensación de movimiento global para codificadores basados en la transformada de wavelet 3D. Varios autores [Kronander, 1989][Ohm, 1994][Choi, 1996] intentaron introducir este concepto con métodos diferentes que no consiguieron la suficiente ganancia con respecto a otros codificadores. En contraposición, Taubman y Zakhor [1994] desarrollaron el codificador LZC, en el cual se utilizaba una compensación panorámica simple para obtener un intervalo de ganancia entre 0,56 y 1,29 dB con respecto a su propio codificador sin compensación de movimiento, para tres secuencias y diferentes tasas de compresión. El método de Wang *et al.* [1999] se basaba en trasladar cada imagen del vídeo a un sistema de coordenadas común para componer un volumen 3D, al que se le aplica la transformada de wavelet 3D. Los parámetros necesarios para realizar la compensación global de movimiento, junto con los coeficientes, se cuantifican y se envían al decodificador. La técnica obtiene una ganancia



media de  $0,56 - 1,02$  dB y  $1,93 - 2,42$  dB con respecto al codificador 3D-SPHIT sin compensación de movimiento, para las secuencias *Guarda Costas* y *Stefan*, respectivamente. Además, el rendimiento obtenido por las dos secuencias es similar al estándar H.263 (desde  $-0,57$  dB hasta  $+0,50$  dB). Otros esquemas, sobre el mismo tema, introducidos pueden encontrarse en [Waldemar *et al.*, 1995][Asbun *et al.*, 1998][Shen y Delp, 1999] y [Yang y Hemami, 1999].

Levy y Wilson [1999] se basaron en la utilización de la transformada de wavelet 3D tres veces en cada una de las dimensiones, para decorrelacionar la secuencia de vídeo original. A continuación, aplicaban un cuantificador basado en vectores que utiliza un *codebook* para expresar la simetría de la señal y un codificador aritmético. Además, las sub-bandas espaciales pasa-alta con las frecuencias temporales más altas no se codificaban, ya que se consideraban imperceptibles para el sistema visual humano. Se presentan resultados para la secuencia de vídeo *Miss América* con las funciones wavelet madre Daub-4 y Daub-16. Para 0,02 bpp se obtiene 34,49 y 34,84 dB a nivel de calidad para las dos funciones wavelet, respectivamente. En las imágenes reconstruidas se aprecian algunos errores como artefactos de bloque y ruido en los bordes.

### 3.3. Codificador basado en la transformada de wavelet 3D

Como hemos mencionado en la sección 3.1, y hemos podido observar en la figura 3.1, un codificador basado en transformada se divide en las fases de transformada, umbralización, cuantificación y codificación entrópica. A continuación, vamos a desarrollar cada una de estas fases para construir nuestro propio codificador basado en la transformada de wavelet 3D [Bernabé *et al.*, 2000b][Bernabé *et al.*, 2000a].

#### 3.3.1. La transformada de wavelet 3D. Elección de la función wavelet madre

El primer paso de un codificador basado en la transformada de wavelet 3D, es aplicar la propia transformada sobre la secuencia de vídeo original. Para aplicar dicha transformada, hay que tener en cuenta tres factores fundamentales para determinar tanto la tasa de compresión final como la calidad del vídeo reconstruido:

1. El número de iteraciones que se aplica la transformada sobre cada una de las dimensiones.
2. La función wavelet madre que se utiliza para llevar a cabo la transformada.
3. El orden en el que se aplica la transformada sobre cada una de las dimensiones.

En general, un número mayor de iteraciones de la transformada de wavelet 3D supone una mayor decorrelación de la información original, por lo que en el proceso de umbralización existirá un número mayor de coeficientes que se puedan eliminar y conseguir, por tanto, una tasa de compresión mayor tras la cuantificación y la codificación entrópica. Sin embargo, este hecho tiene un impacto significativo sobre la calidad del vídeo reconstruido, de tal forma que cuanto mayor sea la tasa de compresión, la calidad del mismo será menor. Por lo tanto, la elección del número de iteraciones que se aplica la transformada constituye un factor determinante en los resultados finales, por lo que debe ser adecuadamente elegido para conseguir una gran tasa de compresión y una buena calidad del vídeo reconstruido.

De la misma forma, la función wavelet madre utilizada para llevar a cabo la transformada tiene una influencia sustancial, tanto en la tasa de compresión como en la calidad final del vídeo reconstruido. Por ello, proponemos evaluar diferentes funciones wavelet madre para determinar cuál tiene un mejor comportamiento sobre las diferentes secuencias de vídeo médico. En primer lugar, consideramos la función de Daubechies de cuatro coeficientes (Daub-4) [Daubechies, 1992], ya que se ha utilizado en distintos trabajos que han demostrado su eficiencia. Además, evaluaremos la función de Haar de dos coeficientes [Vidakovic y Müller, 1994], la función de Daubechies de ocho coeficientes (Daub-8) [Daubechies, 1992], la función Bathlet ortonormal de cuatro coeficientes (Bathlet-4) [Monro *et al.*, 1996], y una función híbrida que aplica la Daub-4 sobre la dimensión en el espacio  $(x, y)$  y la Daub-8 sobre la dimensión en el tiempo (Daub-4-8). En la tabla 3.2 se presentan los coeficientes de las diferentes funciones wavelet madre.

Por último, un tercer factor que puede tener influencia sobre la tasa de compresión y la calidad del vídeo reconstruido final es el orden a seguir al aplicar la transformada sobre cada una de las dimensiones. Debemos analizar cuál es el orden que nos reporta un mayor beneficio sobre el rendimiento final a fin de determinar si es mejor aplicar primero la transformada en el tiempo y, posteriormente, en el espacio o viceversa.

### 3.3.2. La umbralización

Una vez que hemos aplicado la transformada de wavelet 3D a una secuencia de vídeo, el siguiente paso consiste en eliminar de la secuencia transformada aquellos coeficientes wavelet que no proporcionan información importante para la reconstrucción del vídeo. Por lo tanto, tenemos que determinar aquellos coeficientes wavelet cuyos valores no influyen sobre los aspectos determinantes del vídeo y establecerlos a valor cero. De esta forma y cuanto mayor sea el número de coeficientes wavelet con valor cero, mayor tasa de compresión podremos alcanzar en la cuantifi-

<b>Daub-4</b>	<b>Bathlet-4</b>
$h_0 = 0,4829629131445341$	$h_0 = 0,48296291314469$
$h_1 = 0,8365163037378079$	$h_1 = 0,83651630373747$
$h_2 = 0,2241438680420134$	$h_2 = 0,2241438680420134$
$h_3 = -0,1294095225512604$	$h_3 = -0,1294095225512604$
<b>Haar</b>	<b>Daub-8</b>
$h_0 = 0,70710678119$	$h_0 = 0,2303778133088964$
$h_1 = 0,70710678119$	$h_1 = 0,7148465705529154$
	$h_2 = 0,6308807679398587$
	$h_3 = -0,0279837694168599$
	$h_4 = -0,1870348117190931$
	$h_5 = 0,0308413818355607$
	$h_6 = 0,0328830116668852$
	$h_7 = -0,0105974017850690$

Tabla 3.2: Coeficientes para las funciones wavelet madre Daub-4 [Daubechies, 1992], Bathlet-4 [Monro *et al.*, 1996], Haar [Vidakovic y Müller, 1994] y Daub-8 [Daubechies, 1992]

cación y en la codificación entrópica del codificador. Sin embargo, la elección del umbral óptimo no es una tarea sencilla, ya que un umbral por encima del óptimo perjudica la calidad del vídeo reconstruido, mientras que un umbral por debajo del óptimo disminuye la tasa de compresión. Por lo tanto, la elección de un umbral influye significativamente sobre la tasa de compresión y la calidad final de un vídeo determinado.

Por todo ello proponemos dos formas para realizar la umbralización:

- Técnica del percentil. Para implementar esta técnica se elige un umbral ( $p$ ) y se aplica la siguiente función a los coeficientes wavelet:

$$f_{x,y,t} = \begin{cases} 0 & \text{si } d_{x,y,t} < p \\ d_{x,y,t} & \text{si } d_{x,y,t} \geq p \end{cases}$$

En la fórmula anterior  $p$  es el percentil- $x$  elegido para el conjunto de coeficientes wavelet cuyos valores vienen representados por  $d_{x,y,t}$ . Por ejemplo, si se desea aplicar un percentil-90,  $p$  debe calcularse de forma que el 90 % de los valores de los coeficientes wavelet sean inferiores a  $p$ . De esta forma, y según este método, el 90 % de los coeficientes wavelet serán cero.

- Descartar un número determinado de bits menos significativos de cada uno de los coeficientes wavelet, una vez que el proceso de cuantificación se haya realizado. La implementación de este método consiste simplemente en poner a cero los bits menos significativos de cada uno de los coeficientes wavelet. De esta forma, trataremos de encontrar el número de bits menos significativos óptimo para reducir el valor de los citados coeficientes y obtener la máxima tasa de compresión posible. Sin embargo, la eliminación de un determinado número de bits deteriora la calidad del vídeo reconstruido, por lo que dicho número deberá conseguir que la degradación no sea apreciable desde el punto de vista del sistema visual humano en el vídeo reconstruido.

### 3.3.3. La cuantificación y la codificación entrópica

En este paso, se cuantifican los coeficientes wavelet cuyo valor es distinto de cero y se realiza el proceso de codificación de dichos coeficientes. La tarea consiste en la transformación de los coeficientes wavelet en punto flotante en coeficientes enteros sin signo. Para realizar esta tarea, proponemos un cuantificador que asigna a cada coeficiente wavelet un número de bits que depende de la sub-banda wavelet a la que pertenece dicho coeficiente. El número y las características de cada una de las sub-bandas wavelet depende del número de veces que se aplique la transformada de wavelet 3D. En la figura 3.4 podemos ver la asignación de bits a cada una de las sub-bandas wavelet cuando se aplica la transformada de wavelet dos o tres veces. Como ya comentamos en la sección 2.1.2.2, cada vez que se aplica la transformada de wavelet, desde el punto de vista de la dimensión en el espacio (x,y) se generan cuatro sub-bandas (sub-banda pasa baja-baja o imagen de referencia, sub-banda pasa alta-baja o detalles horizontales, sub-banda pasa baja-alta o detalles verticales y sub-banda pasa alta-alta o detalles diagonales), en cada una de las imágenes que componen la secuencia de vídeo. Por lo tanto, el número de bits que se establece para cada sub-banda wavelet depende del máximo valor que pueda alcanzar un coeficiente en la mencionada sub-banda tras la aplicación de la transformada wavelet.

Por ejemplo, si aplicamos una vez la transformada de wavelet sobre imágenes de  $512 \times 512$  pixels, desde el punto de vista de la dimensión en el espacio (x,y) se crean cuatro sub-bandas de  $256 \times 256$  pixels. Si se vuelve a aplicar la transformada por segunda vez, se generan dentro de la imagen de referencia (sub-banda baja-baja), otras cuatro sub-bandas de  $128 \times 128$  pixels. Por todo ello, asignamos 11 bits a la imagen de referencia y 10 bits para cada una de las sub-bandas de  $128 \times 128$  pixels que contienen los detalles, ya que el valor máximo que pueden contener es 2048 ( $2^{11}$ ) y 1024 ( $2^{10}$ ) respectivamente. Dicho valor máximo depende del valor máximo inicial para un determinado carácter, es decir 256 (las imágenes están codificadas en escala de grises), y

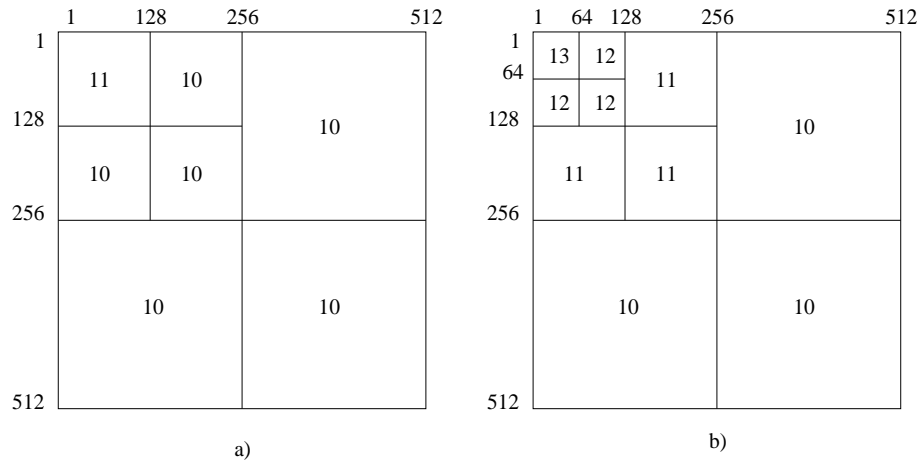


Figura 3.4: Asignación de bits por capas para 2 y 3 iteraciones de la transformada de wavelet

del número de veces que se aplican los filtros pasa-baja y pasa-alta que forman la transformada de wavelet. En este caso, en la imagen de referencia el filtro pasa-baja se ha aplicado seis veces (dos en cada una de las dimensiones), por lo que si suponemos que hemos utilizado la función wavelet madre Daub-4, cada una de las aplicaciones del filtro pasa-baja produce que el valor de un coeficiente aumenta en  $\sqrt{2}$ . Por lo tanto, el valor máximo es  $256x(\sqrt{2})^6 = 2^{11}$ . A continuación y basándonos en el mismo criterio, asignamos 10 bits a las tres sub-bandas que quedan de  $256x256$  pixels, tal y como podemos observar en la figura 3.4 a).

Si se vuelve a aplicar la transformada de wavelet por tercera vez, se generan dentro de la imagen de referencia de  $128x128$  pixels, cuatro sub-bandas de  $64x64$  pixels a las que asignamos 13 bits para la imagen de referencia y 12 bits para cada una de las tres sub-bandas restantes. Las sub-bandas de  $128x128$  pixels tienen 11 bits, mientras que las de  $256x256$  reciben 10 bits para realizar la codificación, tal y como podemos observar en la figura 3.4 b).

Una vez realizada la codificación, y como paso adicional para aumentar la tasa de compresión manteniendo la calidad del vídeo reconstruido, se lleva a cabo la codificación entrópica sobre los coeficientes cuantificados. Proponemos un codificador entrópico que se divide en dos partes:

- Primero, proponemos aplicar un *Run-Length* sobre la representación binaria de los coeficientes cuantificados, es decir enteros sin signo, para poder comprimir las grandes cadenas de ceros consecutivos que se encuentran en la secuencia de vídeo cuantificada. De la misma forma que la cuantificación, el *Run-Length* binario se aplica siguiendo el orden establecido por la descomposición en sub-bandas que produce la aplicación de la transformada de wavelet 3D. Además, dada una sub-banda wavelet y un conjunto de coeficientes representados

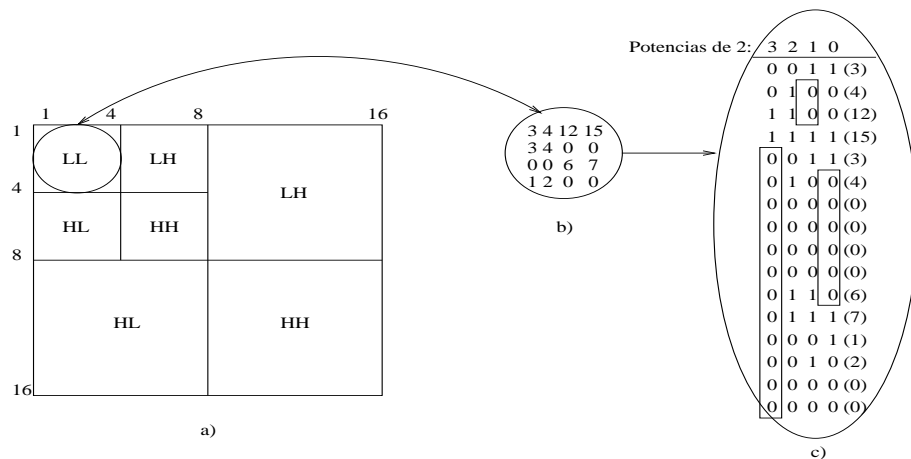


Figura 3.5: a) Resultado de dos iteraciones de la transformada de wavelet sobre una imagen de  $16 \times 16$  pixels. b) Representación decimal de la sub-banda LL. c) Representación binaria de la sub-banda LL y algunos ejemplos de cadenas de ceros consecutivos (rectángulos)

en binario, el *Run-Length* binario sigue un orden de codificación por planos de bits de mayor a menor bits significativos, es decir, primero se codifican los bits que se encuentran en las posiciones más significativas de todos y cada uno de los coeficientes, segundo se codifican los bits que se encuentran en las segundas posiciones más significativas de todos y cada uno de los coeficientes, y así sucesivamente hasta llegar a los bits que se encuentran en las posiciones menos significativas.

En la figura 3.5 a) tenemos una imagen de  $16 \times 16$  pixels a la que se le ha aplicado la transformada de wavelet dos veces. En la partes b) y c) de la figura, podemos ver la representación decimal y binaria respectivamente, de los diferentes coeficientes wavelet de la imagen de referencia de  $4 \times 4$  pixels. En la parte c), se ilustra el funcionamiento del *Run-Length* binario, donde los coeficientes se codifican por planos de bits dentro de una sub-banda y de mayor a menor bits significativos. Los diferentes rectángulos muestran algunas de las cadenas de ceros consecutivos que el *Run-Length* binario va encontrado para conseguir una mayor compresión de los coeficientes.

- Segundo, y una vez aplicado el *Run-Length* binario, proponemos llevar a cabo sobre los coeficientes comprimidos un codificador de Huffman (ver sección 2.3.1), para volver a aumentar la tasa de compresión sin degradar la calidad del vídeo reconstruido. La codificación Huffman asigna un código de longitud variable a cada posible valor en los datos de entrada, tal que los valores que ocurren más a menudo en el conjunto de datos tienen un código de

longitud más pequeña, mientras que los valores que ocurren menos frecuentemente tienen códigos de longitud más largos. Dada la probabilidad de ocurrencia de cada valor de dato individual, el algoritmo de Huffman puede crear automáticamente una asignación de código apropiado para cada valor de datos. Por lo tanto, la asignación de este código es la mejor que se puede hacer cuando se crea un esquema de códigos uno-a-uno (código único para cada valor de datos original). Luego, el código resultante es óptimo, con la restricción de que los símbolos de la fuente se deben codificar uno a uno.

Para determinar el código Huffman a utilizar, debemos tener en cuenta que hemos aplicado previamente el *Run-Length* binario por lo que los distintos símbolos que vamos a encontrar son la longitud de la cadenas de ceros o unos consecutivos, el carácter de repetición y un cero o un uno, que indica el carácter comprimido por el *Run-Length* binario. Por lo tanto, cuanto mayor sea el número de símbolos de Huffman, mayor cantidad de ceros o unos consecutivos podemos permitir en el *Run-Length* binario. Por ejemplo, si tenemos un código Huffman de 16 símbolos, podemos tener cadenas desde uno hasta dieciséis ceros o unos consecutivos, pero no podemos tener cadenas de diecisiete ceros o unos seguidos, ya que sólo tenemos 16 símbolos para representar la información. Sin embargo, un código de Huffman con un número elevado de símbolos aumenta la complejidad, ya que para cada dato de entrada hay que realizar una búsqueda entre un número mayor de símbolos, para asignarle el código correspondiente. Por lo tanto y debido a la naturaleza del código de entrada y la posibilidad de encontrar cadenas de ceros y unos consecutivos, decidimos implementar un codificador de Huffman de 128 símbolos, que nos permite codificar cadenas de hasta 128 ceros o unos consecutivos, y nos ofrece una buena relación entre la complejidad del codificador y el número de ceros o unos consecutivos que puede representar.

### 3.4. Optimizaciones sobre el cuantificador y el codificador entrópico

En esta sección desarrollamos varias propuestas para mejorar sustancialmente la tasa de compresión manteniendo la calidad del vídeo reconstruido [Bernabé *et al.*, 2001]. Por ello, dichas propuestas se realizan sobre el cuantificador y el codificador entrópico respectivamente, ya que de esta forma no se produce una degradación de la calidad del vídeo reconstruido, sino que la tasa de compresión aumenta.

### 3.4.1. *Run-length* binario 3D inteligente

Para obtener un *Run-Length* binario más eficiente para la codificación de los coeficientes cuantificados que el propuesto en la sección 3.3.3, debemos conseguir cadenas con un número elevado de ceros consecutivos. También podemos encontrar cadenas de unos consecutivos, pero se encuentran más dispersos en la secuencia de vídeo por lo que el objetivo fundamental es la compresión eficiente de las cadenas de ceros seguidos. En la sección 3.3.3, el *Run-Length* binario propuesto procesa los coeficientes de forma independiente sobre cada una de las imágenes que componen la secuencia de vídeo, y dentro de cada *frame* sobre cada una de las sub-bandas wavelet obtenidas tras la aplicación de la transformada. Esta forma de procesar los coeficientes es similar a la que realiza el estándar de compresión de imágenes JPEG.

Sin embargo, estamos aplicado la transformada de wavelet 3D sobre una secuencia de vídeo, la cual es decorrelacionada en las dimensiones del espacio  $(x, y)$  y del tiempo. Por lo tanto, el *Run-Length* binario puede obtener una mayor cantidad de cadenas cuyo número de ceros sea muy elevado, si los coeficientes se procesan siguiendo el orden impuesto por la dimensión en el tiempo y dentro de cada uno de los sub-cubos en los que se descompone el vídeo original cuando se aplica la transformada de wavelet 3D. Además, este hecho tiene un mayor impacto en los sub-cubos pasa-alta según el orden determinado por la dimensión en el tiempo. Dichos sub-cubos pasa-alta contienen los detalles del vídeo reconstruido, los cuales si se trata de una secuencia de vídeo médico con poco movimiento, se traducen en una mayoría de coeficientes con valores cero. Además, el resto de coeficientes de dichos sub-cubos cuyos valores son distintos de cero conservan, normalmente, la misma posición espacial en cada una de las imágenes que componen la secuencia.

Por todo ello, proponemos aplicar el *Run-Length* sobre la representación binaria de los valores de los coeficientes cuantificados, siguiendo el orden impuesto por la dimensión en el tiempo [Bernabé *et al.*, 2001]. Además, y de la misma forma que en el cuantificador expuesto en la sección 3.3.3, a cada uno de los sub-cubos en los que se descompone la secuencia de vídeo original al aplicar la transformada de wavelet 3D se le asignará un número diferente de bits para llevar a cabo la cuantificación, dependiendo del número de veces que se lleve a cabo la transformada. Debemos recordar que cada vez que se aplica una nueva iteración de la transformada de wavelet se produce un aumento del valor de los coeficientes. Por tanto, cuanto mayor sea el número de veces que aplica la transformada, mayor será el número de bits que se necesitarán para cuantificar un determinado coeficiente. Por ejemplo, dada una secuencia de vídeo de 64 imágenes de  $512 \times 512$  pixels, la primera iteración de la transformada de wavelet 3D se aplica sobre toda la secuencia generando ocho sub-cubos de 32 imágenes de  $256 \times 256$  pixels. Sin embargo, la segunda iteración de la transformada se aplica sólo sobre el sub-cubo referencia, por lo que dicho sub-cubo generará ocho



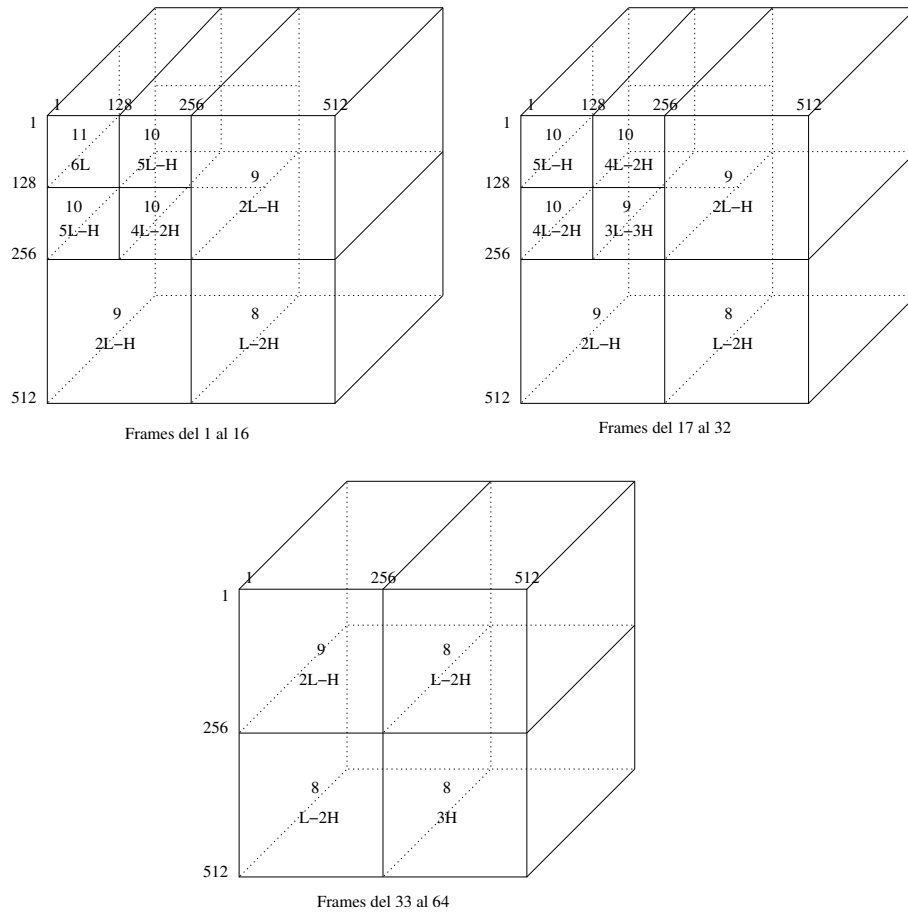


Figura 3.6: Asignación de bits por sub-cubos para dos iteraciones de la transformada de wavelet 3D.

sub-cubos de 16 imágenes de  $128 \times 128$  pixels en los que los valores de los coeficientes wavelet serán mayores que los sub-cubos de la primera iteración. De nuevo, tendremos un sub-cubo referencia sobre el que se puede aplicar por tercera vez la transformada, al que será necesario asignar un número mayor de bits que al resto de sub-cubos para cuantificar correctamente los coeficientes. En la figura 3.6, podemos ver el número de bits que se asignan a cada uno de los sub-cubos que se generan cuando se aplica la transformada de wavelet 3D dos veces. Como podemos observar, el sub-cubo referencia, que se encuentra dentro de las dieciséis primeras imágenes de la secuencia de vídeo y en la parte superior izquierda, tiene el mayor número de bits asignados, es decir 11 bits. El número de bits para cada uno de los sub-cubos depende del número de veces que se aplica los filtros pasa-baja y pasa-alta sobre cada sub-cubo para aplicar la transformada de wavelet y del valor inicial de los pixels que como máximo es 256, ya que las imágenes están codificadas en

escala de grises, tal y como ilustra la figura 3.6.

### 3.4.2. Codificación hexadecimal

En el codificador entrópico expuesto en la sección 3.3.3, el codificador *Run-Length* codifica cadenas de una longitud máxima de 128 ceros o unos consecutivos, ya que el codificador Huffman utiliza 128 símbolos. Este hecho supone una limitación en la tasa de compresión que pueden obtener tanto el *Run-Length* binario original expuesto en la sección 3.3.3 como el *Run-Length* binario 3D inteligente propuesto en el apartado anterior. En este último *Run-Length*, este hecho tiene una importancia superior, ya que los coeficientes wavelet cuantificados se procesan siguiendo el orden determinado por la dimensión en el tiempo, por lo que pueden aparecer cadenas con una longitud de miles de ceros en serie. Un estudio previo sobre la secuencia de vídeo médico *Corazón*, que definiremos en la sección 3.5.2, revela que la media de cadenas de ceros consecutivos se sitúa muy por encima de 128 en la mayoría de sub-cubos que conforman la secuencia. Obviamente, los mayores valores de media se alcanzan en los sub-cubos pasa-alta que contienen un menor número de coeficientes con valores distintos de cero. Por ejemplo, en los sub-cubos etiquetados en la figura 3.6 con  $2L - H$  (9 bits asignados y *frames* del 17 al 32), y  $3H$  (8 bits asignados y *frames* del 33 al 64), se alcanzan medias de 65532 y 65535 ceros consecutivos, respectivamente.

Por todo ello, proponemos utilizar la codificación hexadecimal para representar la longitud de las cadenas de ceros seguidos, de tal forma que no se imponga una limitación en la longitud de dichas cadenas [Bernabé *et al.*, 2001]. La técnica se implementa de la siguiente forma:

- Se utiliza un primer símbolo en hexadecimal que indica el número de dígitos hexadecimales que vamos a usar para codificar la longitud de la cadena de ceros consecutivos.
- Una serie de dígitos hexadecimales que codifican la longitud de la cadena de ceros seguidos.

Por ejemplo, si tenemos una cadena de 15503 ceros consecutivos quedaría representada como  $43C8F$  mediante la codificación hexadecimal. Como podemos observar, el primer dígito, 4, indica el número de cifras hexadecimales necesarios para representar la longitud de la cadena, mientras que  $3C8F$ , codifica la longitud de la cadena, es decir 15503. Además, la tasa de compresión al aplicar la codificación hexadecimal aumenta considerablemente, ya que la codificación de la cadena de 15503 ceros con el *Run-Length* original necesita 366 bytes mientras que la codificación hexadecimal sólo utiliza 2,5 bytes.

De esta forma se pueden representar cadenas con una longitud de miles de ceros consecutivos. La cantidad máxima representada viene determinada por 15 dígitos en hexadecimal con valor  $F$ ,

lo que genera un valor inalcanzable para la longitud de ceros consecutivos de cualquier secuencia de vídeo. Además, podemos sustituir el codificador de Huffman de 128 símbolos por uno de 16 símbolos, lo que reduce la complejidad del mismo y el tiempo de búsqueda para asignar a un símbolo hexadecimal la correspondiente representación en el código Huffman.

Basándonos en el uso de la codificación hexadecimal descrita, vamos a introducir dos mejoras adicionales:

#### **Cadenas de longitud desde uno hasta siete ceros consecutivos**

Se trata de representar las cadenas de longitud desde uno hasta siete ceros consecutivos con un sólo símbolo en hexadecimal o medio byte [Bernabé *et al.*, 2001]. Nuestro estudio preliminar sobre la secuencia de vídeo médico *Corazón* establece una media del 17,5 % de símbolos a los que se le puede aplicar la propuesta actual. Para llevar a cabo esta primera propuesta, debemos de tener en cuenta dos hechos fundamentales:

1. Las cadenas desde uno hasta siete ceros consecutivos necesitan un byte para codificarse según la representación hexadecimal. En este caso, el primer símbolo siempre es el valor uno, mientras que el segundo símbolo es un valor que oscila entre uno y siete.
2. Si reducimos el número máximo de ceros consecutivos a siete cifras hexadecimales con valor  $F$ , el valor máximo representable sería  $7FFFFFFF$  (donde el 7 indica el número de cifras hexadecimales que necesitamos para codificar la longitud de la cadena), o lo que es lo mismo una cadena de 268,435,455 ceros consecutivos que se codifican mediante cuatro bytes. Dicho valor máximo, seguiría siendo un valor inalcanzable para cualquier secuencia de vídeo. Por ejemplo, una secuencia de vídeo de 64 imágenes de  $512 \times 512$  pixels codificada en escala de grises tendría como máximo una cadena de 134,217,728 ceros consecutivos, si todos los coeficientes tuvieran el valor cero.

Basándonos en los dos hechos anteriores, utilizamos el primer bit del primer símbolo hexadecimal, que indica el número de símbolos que codifican la longitud de la cadena, de la siguiente forma:

- Si el primer bit es cero, el primer símbolo se codifica directamente en los tres bits restantes, indicando una cadena de longitud desde uno hasta siete ceros consecutivos. Por lo tanto, sólo necesitamos un símbolo o medio byte para codificar las mencionadas cadenas.

Valor Decimal	Bits			Significado de la codificación hexadecimal
	1	2	3 4	
0	0	0	0 0	Símbolo no utilizado
1	0	0	0 1	Los tres últimos bits codifican directamente la longitud de una cadena desde uno hasta siete ceros consecutivos.
2	0	0	1 0	
3	0	0	1 1	
4	0	1	0 0	
5	0	1	0 1	
6	0	1	1 0	
7	0	1	1 1	
8	1	0	0 0	Codifica una cadena de unos consecutivos.
9	1	0	0 1	Codificación de cadenas de longitud mayores que siete ceros consecutivos: Codificación normal mediante representación hexadecimal
10	1	0	1 0	
11	1	0	1 1	
12	1	1	0 0	
13	1	1	0 1	
14	1	1	1 0	
15	1	1	1 1	

Tabla 3.3: Posibles valores del primer símbolo de codificación hexadecimal

- Si el primer bit es uno, los tres siguientes bits indican el número de dígitos hexadecimales que codifican la cadena de ceros seguidos, es decir, se trata de una representación estándar donde utilizamos un símbolo para indicar el número de bytes que codifican la longitud de la cadena y una serie de símbolos que codifican la longitud de la cadena de ceros seguidos.

En la tabla 3.3 podemos observar los diferentes valores que puede tomar el primer símbolo de cuatro bits cuando se utiliza la codificación hexadecimal. Como se puede ver, el primer bit distingue las cadenas de longitud mayores que siete ceros de las menores o iguales que siete ceros consecutivos. Estas últimas se representan mediante un único símbolo o medio byte, ya que si el primer bit es 0, el primer símbolo representa directamente la longitud de la cadena en los bits 2, 3 y 4. Si el primer bit es uno, los bits 2, 3 y 4 indican el número de cifras hexadecimales (entre uno y siete), que codifican la longitud de la cadena.

### Cadenas de unos consecutivos

Una vez implementadas la codificación hexadecimal y la propuesta anterior, en el primer símbolo, que indica el número de símbolos que codifican la longitud de la cadena, nunca va a aparecer el valor decimal 8 cuya representación binaria es 1000. Por lo tanto, utilizamos el valor 8 para codificar las cadenas de unos consecutivos que pueden aparecer en la secuencia de vídeo [Bernabé *et al.*, 2001]. De esta forma, si el primer símbolo hexadecimal de representación de una cadena es 8, los siguientes símbolos codifican la longitud de una cadena de unos consecutivos, de la misma forma que se codifica la longitud de una cadena de ceros consecutivos, es decir, se utiliza un primer símbolo para indicar el número de símbolos que codifican la cadena y los correspondientes símbolos para codificar la longitud de la cadena de unos seguidos.

En resumen, esta propuesta nos permite representar mediante codificación hexadecimal las cadenas de unos consecutivos para conseguir una mayor compresión que la realizada hasta este momento, mediante el *Run-Length* binario descrito anteriormente en la sección 3.3.3. Por ejemplo, si tenemos una cadena de diez unos consecutivos, se representaría mediante codificación hexadecimal como 81A. De esta forma sólo se necesitan 1,5 bytes para representar la cadena, mientras que con el *Run-Length* binario se habrían necesitado 3 bytes (carácter de repetición, carácter que se repite y número de veces que se repite dicho carácter).

#### 3.4.3. Codificación aritmética

Hasta ahora hemos utilizado un codificador de Huffman para, una vez que hemos aplicado las propuestas anteriores, seguir aumentando la tasa de compresión manteniendo la calidad del vídeo reconstruido. Proponemos sustituir el codificador de Huffman por un codificador aritmético, el cual, basa su funcionamiento en representar una secuencia de símbolos de entrada mediante un número en punto flotante [Nelson y Gailly, 1996]. De esta forma, el codificador aritmético es más eficiente que el codificador de Huffman, ya que este último se basa en representar cada símbolo de entrada mediante un código específico de salida. Además, y teniendo en cuenta que estamos utilizando codificación hexadecimal, que sólo utiliza dieciséis símbolos para la representación, proponemos utilizar un codificador aritmético implementado con dieciséis símbolos [Bernabé *et al.*, 2001].

El codificador aritmético examina los símbolos de entrada para determinar la frecuencia de aparición o la probabilidad de cada uno de los 16 símbolos que componen la secuencia de entrada. A continuación y tal y como explicamos en la sección 2.3.2, dependiendo de la probabilidad de aparición de cada uno de los símbolos, se divide el intervalo  $[0, 1)$  en dieciséis subintervalos. A

partir del valor más bajo y más alto de cada símbolo de entrada y el número total de símbolos de entrada, se determina la salida correspondiente.

### 3.5. Evaluación de resultados

En esta sección, vamos a evaluar las diferentes propuestas explicadas anteriormente sobre tres secuencias de vídeo médicas.

#### 3.5.1. Métricas para determinar la calidad del vídeo comprimido

Para determinar la calidad del codificador propuesto basado en la transformada de wavelet 3D, y de las diferentes mejoras introducidas, vamos a utilizar las siguientes métricas:

- a) La tasa de compresión.
- b) La calidad del vídeo reconstruido.

La tasa de compresión se obtiene dividiendo el tamaño original de la secuencia de vídeo a comprimir por el tamaño comprimido de dicha secuencia.

Para determinar la calidad del vídeo reconstruido mediante una evaluación numérica, vamos a utilizar el pico de la relación señal/ruído (PSNR). La media aritmética del PSNR de cada una de las imágenes que forman la secuencia de vídeo será el PSNR del vídeo reconstruido. Además, hemos observado y verificado visualmente la calidad de los vídeos reconstruidos.

#### 3.5.2. Entorno de trabajo

Las evaluaciones de las diferentes propuestas expuestas en las secciones anteriores se han llevado a cabo sobre un bi-procesador Intel Pentium III con una frecuencia de  $450MHz$ , del que sólo hemos utilizado un procesador. La memoria RAM tiene una capacidad de 256 Mbytes. Como sistema operativo se ha utilizado Linux 2.2.12 – 20smp. El código fuente que incluye el codificador, el decodificador y las distintas mejoras se han escrito usando el lenguaje de programación C.

#### Secuencias de vídeo médico

Para evaluar las diferentes propuestas, hemos comprimido y descomprimido tres secuencias de vídeo médicas codificadas en escala de grises (8 bits por pixel). La obtención de las secuencias

de vídeo y de la información médica en general no es sencilla, ya que se trata de información confidencial de determinados pacientes que no se puede difundir públicamente. Sin embargo, el Hospital Recoletas de Albacete nos donó las dos primeras secuencias que vamos a evaluar:

- *Corazón*. La secuencia de vídeo representa el latido del corazón humano y las regiones cercanas.
- *Catéter*. Esta secuencia representa la introducción de un catéter en el cuerpo humano.

Las dos secuencias descritas constan de 256 imágenes de  $512 \times 512$  pixels. Ambas secuencias disponen de una cantidad de movimiento mayor de la que cabría esperar para un vídeo médico típico. Entre las dos, la secuencia *Catéter* tiene menor cantidad de movimiento que la secuencia *Corazón*.

Debido a la dificultad de obtener un número mayor de secuencias de vídeo médico, la tercera secuencia que vamos a evaluar ilustra el movimiento de una mano humana que mueve los dedos. La creación de esta última secuencia denominada *Mano*, se realizó de forma artificial en nuestro laboratorio mediante una cámara de vídeo y tiene como principal propiedad la existencia de poco movimiento entre las diferentes imágenes que componen la secuencia de vídeo. El objetivo es determinar el potencial de los métodos propuestos en secuencias de vídeo médico que poseen una cantidad de movimiento mínima. Por tanto, la secuencia *Mano* posee una menor cantidad de movimiento que las secuencias *Catéter* y *Corazón*. La secuencia *Mano* se compone de 256 imágenes de  $256 \times 256$  pixels.

Hemos dividido cada uno de las secuencias de vídeo médico en cuatro bloques de 64 imágenes, para no tener que manejar una gran cantidad de memoria en un instante determinado.

### Validación de las secuencias de vídeo médico

Una dificultad igual o superior a la de obtener las secuencias de vídeo médico se pone de manifiesto en la validación de las mismas por parte de especialistas médicos. Para resolver este problema, hemos iniciado una colaboración con varios profesores de la Universidad Pontificia Javeriana de Cali (Colombia). El proyecto tiene como objetivo la creación de una arquitectura genérica de software para la prestación de servicios de telemedicina a través de Internet [wco, 2004], e incluye la compresión y transmisión de secuencias de vídeo médicas utilizando las propuestas basadas en la transformada de wavelet 3D desarrolladas en la presente tesis doctoral.

Actualmente, se ha desarrollado la arquitectura software para varias especialidades médicas

como otorrinolaringología [Vélez *et al.*, 2002], dermatología [Vélez *et al.*, 2003] y oftalmología [Navarro *et al.*, 2003b][Navarro *et al.*, 2003a]. Dicho proyecto cuenta con una serie de especialistas médicos que se encargan de determinar la validez de las secuencias de vídeo médico reconstruidas. Algunas de las secuencias validadas y codificadas mediante las técnicas propuestas en el presente trabajo, así como los resultados en cuanto a tasa de compresión y calidad visual mediante el PSNR se pueden encontrar en la dirección web <http://ditec.um.es/gbernabe/colombia/index.shtml>.

### 3.5.3. Evaluación del codificador base

En este apartado, vamos a analizar diversos factores que influyen en la tasa de compresión y la calidad del vídeo reconstruido cuando se utiliza el codificador basado en la transformada de wavelet 3D, desarrollado en la sección 3.3. Por lo tanto, evaluaremos cómo afecta sobre el rendimiento final el número de iteraciones que se aplica la transformada de wavelet, el valor del percentil para llevar a cabo la umbralización, el número de bits que se descartan en el proceso de cuantificación y la utilización de las diferentes funciones wavelet madre propuestas en la sección 3.3.1: Haar, Daub-4, Daub-8, Bathlet-4 y el híbrido Daub-4-8.

En cuanto al orden para aplicar la transformada sobre cada una de las dimensiones, hemos comprobado que dicho orden no afecta al rendimiento final por lo que hemos decidido aplicar la transformada de wavelet 3D primero en la dimensión en el tiempo, y posteriormente, en el espacio  $(x, y)$ .

En las tablas 3.4, 3.5 y 3.6 se exponen los resultados para tres secuencias de vídeo médico: Corazón, Catéter y Mano. En cada celda de cada una de las tablas anteriores, se muestra la tasa de compresión (valor superior) y el PSNR del vídeo reconstruido (valor inferior). Se pueden observar distintas configuraciones de las propuestas introducidas anteriormente, según los siguientes parámetros:

1. El parámetro  $b$  indica el número de bits menos significativos descartados o que se ponen a cero en el proceso de cuantificación, en cada uno de los coeficientes wavelet.
2. El parámetro  $per-X$  representa el percentil aplicado en la fase de umbralización, para descartar todos aquellos coeficientes wavelet que estén por debajo de un determinado valor.
3. El parámetro  $2(t,x,y)$  o  $3(t,x,y)$  determina el número de iteraciones que se aplica la transformada de wavelet en cada una de las dimensiones (tiempo,  $x$  e  $y$ ). Hemos descartado los resultados que se obtienen al aplicar la transformada de wavelet una sola vez, ya que la tasa de compresión es muy pequeña e insuficiente para el objetivo del trabajo. Por otro lado,



Corazón	Haar		Daub-4		Daub-8	
	2(t,x,y)	3(t,x,y)	2(t,x,y)	3(t,x,y)	2(t,x,y)	3(t,x,y)
b=2-per93	7,53	7,68	7,76	8,00	8,14	8,20
	39,77	34,60	39,97	36,63	38,95	39,15
b=2-per95	9,49	9,76	9,68	9,78	9,87	9,84
	38,17	34,06	37,83	35,77	37,28	37,68
b=2-per96	10,93	10,76	11,6	11,08	11,71	11,48
	37,14	33,75	36,61	35,20	36,05	36,72
b=2-per97	14,40	13,20	14,2	13,73	14,10	13,96
	35,25	33,21	34,78	34,30	34,65	35,47
b=3-per93	9,33	9,61	9,49	9,98	9,89	10,17
	39,31	34,41	39,15	36,34	38,52	38,69
b=3-per95	11,72	12,12	11,78	12,10	11,91	12,24
	37,90	33,93	37,52	35,59	37,02	37,38
b=3-per96	13,31	13,48	13,81	13,53	13,82	13,95
	36,95	33,63	36,42	35,06	35,89	36,52
b=3-per97	16,76	16,08	16,57	16,39	16,64	16,54
	35,18	33,13	34,68	34,22	34,09	35,37

Tabla 3.4: Tasa de compresión y PSNR de la secuencia *Corazón* para las funciones wavelet madre Haar, Daub-4 y Daub-8

los resultados para cuatro o más iteraciones de la transformada no se incluyen ya que la calidad del vídeo reconstruido se deteriora gravemente, lo que genera vídeos inaceptables desde el punto de vista médico.

En cuanto a la función wavelet madre, la función Daub-4 obtiene la mejor relación entre la tasa de compresión y la calidad del vídeo reconstruido para las secuencias *Corazón* y *Mano*. Con respecto a la secuencia *Catéter*, la función Haar obtiene el mejor rendimiento, aunque los resultados de la funciones Daub-4 y Daub-8 son muy similares y cercanos a la Haar. En las tres secuencias, los resultados de la función Daub-8 son muy parecidos al rendimiento de la Daub-4. Sin embargo, si tenemos en cuenta el tiempo de ejecución, la implementación de la Daub-8 es mucho más lenta (alrededor de un 20 %), ya que tiene un número mayor de coeficientes que la Daub-4, lo que aumenta el número de operaciones en punto flotante para calcular la transformada de wavelet.

Catéter	Haar		Daub-4		Daub-8	
	2(t,x,y)	3(t,x,y)	2(t,x,y)	3(t,x,y)	2(t,x,y)	3(t,x,y)
b=2-per93	9,55	9,51	9,69	10,07	9,62	10,01
	40,01	40,53	38,88	39,41	38,87	39,43
b=2-per95	11,74	12,06	11,95	12,09	11,84	11,96
	37,95	38,83	36,58	37,65	36,80	37,84
b=2-per96	14,13	13,95	14,15	14,04	14,17	13,94
	36,10	37,76	35,22	36,46	35,28	36,71
b=2-per97	16,58	16,86	16,68	17,18	16,71	17,38
	34,27	36,22	33,66	35,14	33,49	35,25
b=3-per93	11,58	11,75	11,66	12,23	11,61	12,13
	39,58	39,96	38,45	38,98	38,46	39,02
b=3-per95	14,29	14,94	14,47	14,87	14,33	14,69
	37,72	38,50	36,34	37,39	36,57	37,57
b=3-per96	16,60	17,20	16,68	17,30	16,60	17,21
	36,01	37,52	35,09	36,27	35,17	36,52
b=3-per97	19,33	20,11	19,49	20,53	19,42	20,60
	34,23	36,11	33,48	35,05	33,44	35,17

Tabla 3.5: Tasa de compresión y PSNR de la secuencia *Catéter* para las funciones wavelet madre Haar, Daub-4 y Daub-8

En cuanto a las funciones Bathlet-4 y el híbrido Daub-4-8, la tasa de compresión y la calidad del vídeo reconstruido obtenidos son muy parecidos a los que se obtienen con la función Daub-4. Por lo tanto, se asumen los resultados de la función Daub-4 como equivalentes a los de las funciones Bathlet-4 y Daub-4-8, y no se incluyen en la memoria de la presente tesis, ya que no ofrecen ninguna particularidad relevante.

Desde el punto de vista del número de iteraciones que se aplica la transformada de wavelet 3D, podemos observar que el paso de dos a tres iteraciones para la secuencia de vídeo *Corazón* es perjudicial, ya que aunque la tasa de compresión aumenta, por el contrario la calidad que viene dada por el PSNR disminuye para las funciones madre Haar y Daub-4 en un rango de 1 a 5 dB. Podemos establecer que la calidad del vídeo reconstruido es *excelente* o que no existen diferencias entre el vídeo original y el reconstruido cuando todas las imágenes de la secuencia tienen un PSNR alrededor de 41 dB, mientras que la calidad del vídeo reconstruido es *buena* o aceptable

Mano	Haar		Daub-4		Daub-8	
	$2(t, x, y)$	$3(t, x, y)$	$2(t, x, y)$	$3(t, x, y)$	$2(t, x, y)$	$3(t, x, y)$
b=2-per95	9,46	10,05	9,46	9,67	9,68	10,06
	44,03	44,74	44,54	45,29	44,36	44,78
b=2-per96	10,58	10,84	10,64	11,01	10,98	11,57
	42,78	44,08	43,48	44,37	43,32	43,93
b=2-per97	12,73	12,39	13,05	12,43	13,46	13,04
	41,08	42,75	41,86	43,38	41,73	42,99
b=2-per98	17,75	15,39	18,36	15,99	18,72	16,56
	37,63	41,02	38,73	41,68	38,58	41,44
b=3-per95	11,44	12,87	11,65	13,23	11,93	13,58
	43,28	43,55	43,47	43,75	43,30	43,44
b=3-per96	12,76	13,33	12,80	13,75	13,17	14,15
	42,31	43,29	42,79	43,53	42,66	43,23
b=3-per97	14,92	15,21	15,44	15,22	15,87	15,98
	40,87	42,26	41,46	42,86	41,35	42,50
b=3-per98	20,29	18,35	20,97	19,25	21,34	19,90
	37,58	40,80	38,60	41,41	38,45	41,19

Tabla 3.6: Tasa de compresión y PSNR de la secuencia *Mano* para las funciones wavelet madre Haar, Daub-4 y Daub-8

por parte de la comunidad médica cuando todas las imágenes de la secuencia tienen un PSNR alrededor de 38 dB. Por lo tanto, para la secuencia *Corazón* no es aconsejable aplicar más de dos veces la transformada de wavelet 3D, ya que la calidad del vídeo reconstruido dejaría de ser *buena*. Sin embargo, para la secuencia de vídeo *Catéter* ocurre el fenómeno contrario, es decir, es mejor aplicar la transformada tres veces ya que mejora tanto la tasa de compresión como la calidad del vídeo reconstruido. Este hecho se debe a que la secuencia *Catéter* tiene una menor cantidad de movimiento que la secuencia *Corazón*, por lo que un mayor número de iteraciones de la transformada de wavelet aumenta la tasa de compresión y no deteriora la calidad del vídeo reconstruido. En cuanto a la secuencia de vídeo *Mano*, el paso de dos a tres iteraciones puede perjudicar o mejorar la relación entre la tasa de compresión y la calidad del vídeo reconstruido dependiendo de otros factores, como el percentil utilizado o el número de bits descartados en la fase de cuantificación.

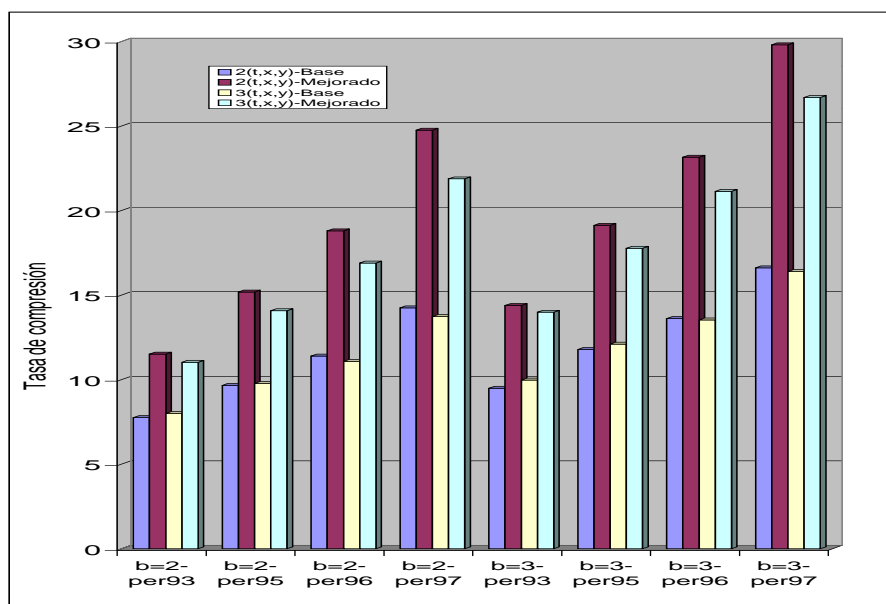
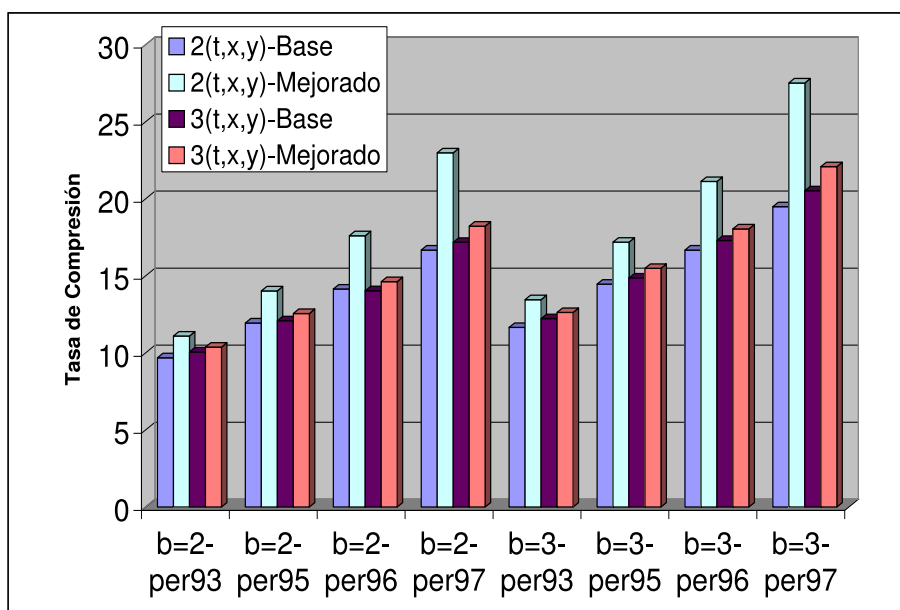
En relación al número de bits descartados en la fase de cuantificación y del percentil utilizado para llevar a cabo la umbralización, se puede observar que cuanto mayor sea el número de ambos factores, mayor será la tasa de compresión alcanzada. Sin embargo, a medida que el percentil aumenta, la calidad del vídeo reconstruido disminuye considerablemente. Este hecho es fundamental para poder elegir una configuración óptima, en la que se obtenga una buena tasa de compresión y no se degrade la calidad del vídeo reconstruido. En cuanto al número de bits descartados, el paso de dos a tres bits mejora claramente la tasa de compresión y no produce una degradación del vídeo reconstruido.

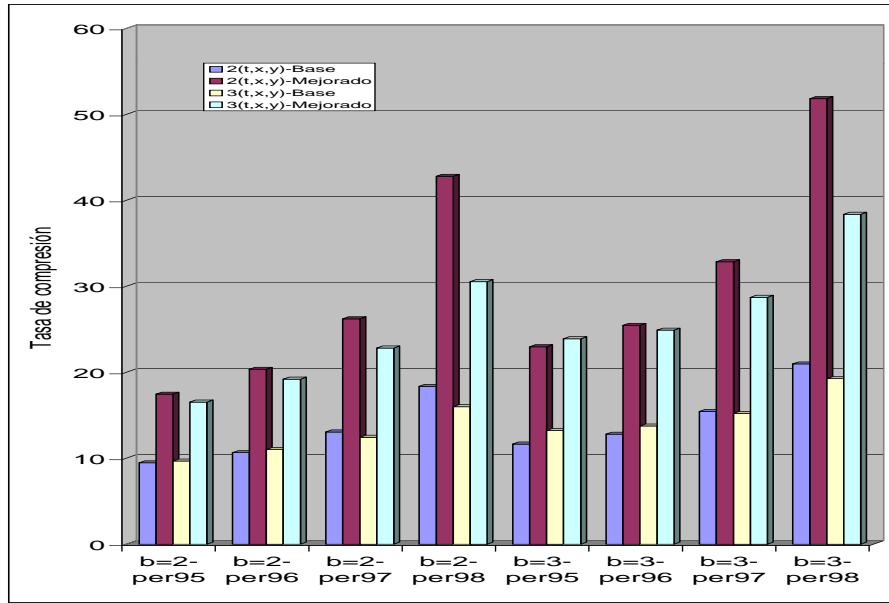
Por lo tanto y una vez analizados los diferentes factores que influyen en la tasa de compresión y en la calidad del vídeo reconstruido sobre las tres secuencias evaluadas, llegamos a las siguientes conclusiones:

- La función wavelet madre Daub-4 tiene el mejor comportamiento en cuanto a la tasa de compresión y calidad del vídeo reconstruido. Además tiene un tiempo de ejecución menor que la Daub-8.
- El número óptimo de veces que se ha de aplicar la transformada de wavelet varía entre dos o tres iteraciones, dependiendo de la secuencia en cuestión.
- Un percentil adecuado para una configuración óptima puede variar entre el 95 y el 96, dependiendo de la secuencia concreta.
- El número de bits que se descartan en el proceso de cuantificación debe ser tres, ya que mejora notablemente la tasa de compresión y no perjudica a la calidad del vídeo reconstruido.

#### 3.5.4. Evaluación del codificador mejorado

En esta sección, vamos a analizar cómo influyen en la tasa de compresión y en la calidad del vídeo reconstruido las propuestas introducidas en la sección 3.4 sobre el cuantificador y el codificador entrópico. Para ello, vamos a comparar los resultados del codificador base con los resultados del codificador mejorado sobre las secuencias de vídeo médico *Corazón*, *Catéter* y *Mano*.

Figura 3.7: Tasa de compresión para la secuencia de vídeo *Corazón*Figura 3.8: Tasa de compresión para la secuencia de vídeo *Catéter*

Figura 3.9: Tasa de compresión para la secuencia de vídeo *Mano*

En las figuras 3.7, 3.8 y 3.9 se pueden observar las tasas de compresión del codificador base y mejorado para las secuencias de vídeo *Corazón*, *Catéter* y *Mano*. Para poder realizar una comparación completa entre los resultados de ambos codificadores, presentamos el mismo abanico de configuraciones que en la sección 3.5.3, salvo que la función wavelet madre es la Daub-4. Por lo tanto, en las figuras podemos ver las tasas de compresión para dos y tres iteraciones de la transformada de wavelet, percentiles desde 93 hasta 98 en la fase de umbralización y dos o tres bits descartados en la fase de cuantificación. En cuanto a la calidad del vídeo reconstruido, el PSNR de las diferentes configuraciones es el mismo que el que aparece en las tablas 3.4, 3.5 y 3.6 para la función wavelet madre Daub-4, ya que las mejoras propuestas se introducen en el cuantificador y en el codificador entrópico, por lo que la tasa de compresión aumenta y la calidad del vídeo reconstruido se mantiene intacta.

En las tres figuras podemos observar que el codificador mejorado obtiene tasas de compresión más elevadas que el codificador base para todas las configuraciones. Además, a medida que el número de bits descartados y el percentil son mayores, la diferencia entre el codificador base y el mejorado aumenta considerablemente. Este hecho se debe a que las distintas mejoras introducidas tienen un mejor funcionamiento cuando hay una gran presencia de ceros consecutivos en las secuencias comprimidas. Por ejemplo, en la secuencia *Corazón*, si se aplica dos veces la transformada de wavelet, se utiliza un percentil-95 y se descartan dos bits, la tasa de compresión

sión aumenta en un 57 %, mientras que para un percentil-97 y descartando tres bits, la tasa de compresión aumenta en un 79 %.

Con respecto a la secuencia *Mano*, el codificador mejorado obtiene mejores resultados que con las secuencias *Corazón* y *Catéter*, ya que la secuencia *Mano* tiene una menor cantidad de movimiento, lo que se traduce en una mayor cantidad de coeficientes con valor cero en posiciones consecutivas según la dimensión en el tiempo. Esto confirma el potencial del codificador mejorado para las secuencias que tienen poco movimiento, y vuelve a poner de manifiesto que las mejoras introducidas explotan la gran presencia de ceros consecutivos. Por ejemplo, si se aplican dos iteraciones de la transformada wavelet, se descartan tres bits y se utiliza un percentil-97, el codificador mejorado aumenta la tasa de compresión en un 41 %, 79 % y 113 % en las secuencias de video *Catéter*, *Corazón* y *Mano*, respectivamente.

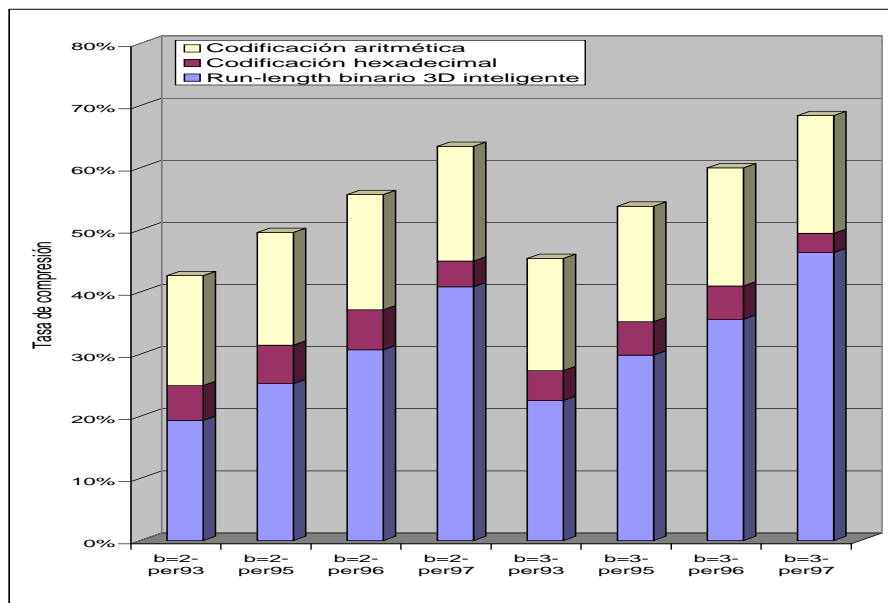


Figura 3.10: Contribución de cada una de las mejoras en la secuencia *Corazón*

En la figura 3.10, se presenta la contribución de cada una de las optimizaciones desarrolladas en la sección 3.4 para dos iteraciones de la transformada de wavelet sobre la secuencia de vídeo médico *Corazón*. En primer lugar, podemos observar que el *Run-Length* binario 3D inteligente es el principal responsable del aumento de la tasa de compresión en el codificador mejorado. De hecho, la utilización del *Run-Length* binario 3D inteligente produce un aumento de la tasa de compresión entre un 20 % y un 40 % en las diferentes configuraciones. Estos porcentajes suponen que dicha propuesta es la responsable de entre un 45 % y un 68 % del aumento global de

la tasa de compresión. Por lo tanto, dichos porcentajes confirman que la cuantificación de los coeficientes wavelet siguiendo el orden determinado por la dimensión en el tiempo genera cadenas de ceros consecutivos de una gran longitud, que pueden ser comprimidas de forma eficiente por el codificador entrópico, aunque se utilice un codificador de Huffman de 128 símbolos.

Por otro lado, la codificación hexadecimal proporciona una contribución sobre el aumento global de la tasa de compresión en un rango que varía desde un 5 % hasta un 13 % para las diferentes configuraciones. Por lo tanto y a la vista de los citados porcentajes, esta propuesta no parece introducir, por ella misma, un beneficio significativo para la tasa de compresión. Sin embargo, el aumento de la tasa de compresión se mantiene constante (alrededor de un 5 %), en todas las configuraciones, por lo que la introducción de la codificación hexadecimal no depende de la longitud de las cadenas de ceros consecutivos y permite una mayor compresión. Además, la utilización de la codificación hexadecimal beneficia el posterior uso de la codificación aritmética en el codificador entrópico, en vez de un codificador de Huffman.

Finalmente, el uso del codificador aritmético proporciona un aumento de la tasa de compresión de alrededor de un 20 % en todas las configuraciones. Dichos resultados son mejores que los que se podrían esperar a priori, ya que en otros trabajos se ha sustituido el codificador de Huffman por un codificador aritmético, y el aumento de la tasa de compresión se ha situado entre un 5 % y un 10 % [Smith, 1997]. Por lo tanto y tal como habíamos anticipado, la utilización previa de la codificación hexadecimal en el codificador entrópico, mejora de forma significativa, y debido a un efecto sinérgico, la sustitución del codificador de Huffman por un codificador aritmético.

### 3.5.5. Comparación del codificador base y mejorado con MPEG-2 y EZW

Con el objetivo de evaluar tanto el codificador base como el codificador mejorado con respecto al estándar de compresión de vídeo MPEG-2 [MPEG, 1994], basado en la transformada discreta del coseno, y el codificador EZW [Shapiro, 1993], basado en la construcción de árboles de ceros y en la transformada de wavelet, hemos comprimido y descomprimido dos de las tres secuencias evaluadas anteriormente: *Corazón* y *Mano*, usando EZW en la fase de cuantificación y MPEG-2.

Para ello, hemos obligado que MPEG-2 y EZW obtengan la misma calidad del vídeo reconstruido, en términos de PSNR, que el codificador base y mejorado en las diferentes configuraciones, para poder realizar una comparación en función de la tasa de compresión. Por lo tanto, la figura 3.11 presenta dos conjuntos de barras para cada una de las secuencias de vídeo *Corazón* y *Mano*. El primer conjunto se obtiene cuando la calidad del vídeo reconstruido es excelente (PSNR alrededor de 41 para todas las imágenes que componen la secuencia de vídeo), es decir, no existen



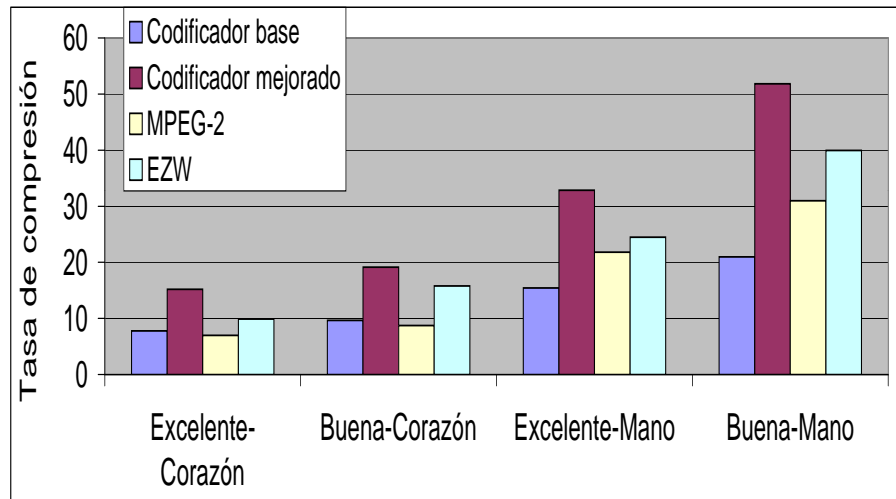


Figura 3.11: Tasas de compresión del codificador base, codificador mejorado, MPEG-2 y EZW para las secuencias *Corazón* y *Mano*

diferencias entre el vídeo original y el reconstruido. El segundo conjunto se obtiene cuando la calidad del vídeo reconstruido es buena (PSNR alrededor de 38 para todas las imágenes que componen la secuencia), constituyendo una secuencia de vídeo aceptable por parte de la comunidad médica.

En primer lugar, podemos observar que EZW tiene una mayor tasa de compresión que el estándar MPEG-2 y nuestro codificador base para ambas secuencias y niveles de calidad. Esto pone de manifiesto los problemas del estándar MPEG-2 para comprimir secuencias de vídeo médico, ya que si configuramos MPEG-2 para que obtenga la máxima compresión posible sin tener en cuenta la calidad del vídeo reconstruido, las tasas de compresión para las diferentes secuencias son significativamente mayores que las que se pueden obtener con un codificador wavelet. Sin embargo, los vídeos reconstruidos tienen una calidad muy mala en la que aparecen los problemas de MPEG-2, como los artefactos de bloque y la aparición de manchas en las imágenes. Además, el algoritmo de MPEG-2 es mucho más complejo que el algoritmo wavelet, ya que después de aplicar la transformada del coseno tiene que llevar a cabo una codificación del movimiento que aumenta, en aproximadamente un 10 %, el tiempo del proceso de compresión con respecto a un codificador wavelet. En cuanto a nuestro codificador base, el mejor comportamiento de EZW, que también se basa en la transformada de wavelet, se debe a las limitaciones impuestas y descritas por nuestro codificador, como la utilización de un codificador de Huffman de 128 símbolos en la codificación entrópica o el procesamiento de los coeficientes wavelet por capas y de forma independiente sobre cada una de las imágenes que componen la secuencia de vídeo.

Finalmente, nuestro codificador mejorado obtiene tasas de compresión significativamente mayores que EZW, MPEG-2 y el codificador original, para ambas secuencias de vídeo y niveles de calidad. Como podemos ver en la figura 3.11, el codificador mejorado aumenta la tasa de compresión del codificador EZW en un 21 % y un 54 % en la secuencia *Corazón* y en un 30 % y un 34 % en la secuencia *Mano* para los niveles de calidad excelente y buena, respectivamente. Estos resultados demuestran que las mejoras introducidas aumentan significativamente los resultados del codificador base y explotan de forma eficiente las características de la transformada de wavelet para la compresión de vídeos médicos. Además, la relación entre las tasas de compresión y el nivel de calidad es excelente, ya que supera notablemente al estándar MPEG-2 y al codificador EZW, sin añadir una complejidad superior al tiempo de ejecución.

### 3.6. Conclusiones

En este capítulo, hemos desarrollado y evaluado un codificador basado en la transformada de wavelet 3D orientado a la compresión de vídeo médico, mediante el desarrollo de las siguientes propuestas.

- El codificador está basado en la transformada de wavelet 3D, por lo que aplica primero la transformada en el tiempo y después en el espacio (x,y). El objetivo consiste en explotar el poco movimiento que deben tener las secuencias de vídeo médico y aprovechar tanto las redundancias espaciales como las temporales.
- Hemos evaluado la utilización de diferentes funciones wavelet madre como la Haar, Daub-4, Daub-8, Bathlet-4 y un híbrido entre la Daub-4-8, y hemos analizado el número de veces que se debe aplicar dicha función para obtener el máximo rendimiento entre la tasa de compresión y la calidad del vídeo reconstruido. De nuestra evaluación se ha seleccionado la función Daub-4 como la que nos ofrece mejor relación coste-prestaciones.
- Se han propuesto dos técnicas para llevar a cabo la umbralización. Por un lado, el uso de la técnica del percentil para eliminar todos aquellos coeficientes wavelet cuyo valor sea menor que un determinado umbral, y por otro lado, la eliminación de los bits menos significativos de los coeficientes wavelet en el proceso de cuantificación.
- Proponemos un cuantificador que asigna un número de bits a cada uno de los coeficientes wavelet que forman cada una de las sub-bandas de una imagen en las que descompone el vídeo la transformada de wavelet. El número de bits depende del número de veces que se aplica la transformada en cada una de las citadas sub-bandas de una imagen determinada.

- Desarrollamos un codificador entrópico basado en la utilización de un *Run-Length* binario que codifica la representación binaria de los coeficientes wavelet en cada una de las sub-capas, y de un codificador de Huffman de 128 símbolos. El codificador entrópico permite aumentar la tasa de compresión y mantener la calidad del vídeo reconstruido.

Con el desarrollo de las propuestas anteriores, tenemos un codificador inicial basado en la transformada de wavelet 3D: *codificador base*. Hemos evaluado dicho codificador sobre varias secuencias de vídeo médico para analizar el rendimiento del mismo, teniendo en cuenta la tasa de compresión y la calidad de los vídeos reconstruidos. Los resultados son buenos y demuestran un potencial importante para el uso de la transformada de wavelet 3D en la compresión de vídeo médico. Sin embargo, hemos encontrado algunas limitaciones, por lo que hemos realizado algunas propuestas sobre el cuantificador y el codificador entrópico, que nos han permitido desarrollar el codificador mejorado. Las propuestas de mejora han sido las siguientes:

- Proponemos un codificador *Run-Length* binario 3D inteligente, que permite comprimir eficientemente las cadenas de ceros y unos consecutivos que se pueden encontrar en las secuencias de vídeo, una vez aplicada la transformada de wavelet y siguiendo el orden determinado por la dimensión en el tiempo.
- Desarrollamos un cuantificador que asigna a cada coeficiente wavelet un número de bits, dependiendo del sub-cubo al que pertenece dicho coeficiente y del número de veces que se ha aplicado la transformada de wavelet sobre cada uno de los sub-cubos en los que se descompone el vídeo original al aplicar la transformada de wavelet 3D.
- Se ha propuesto el uso de la representación hexadecimal para permitir una codificación eficiente de las cadenas de longitud ilimitada de ceros o unos consecutivos en el *Run-Length*. Además, se introducen dos mejoras adicionales para la codificación eficiente de las cadenas de longitud desde uno hasta siete ceros consecutivos y de las cadenas de unos consecutivos.
- Hemos sustituido el codificador de Huffman por un codificador aritmético implementado con 16 símbolos, que comprime de manera más eficiente y en un porcentaje mayor (alrededor de un 20 %) de lo esperado a priori que el codificador de Huffman, manteniendo la calidad original.

Todas estas propuestas tienen en común el aumento de la tasa de compresión del codificador base entre un 40 % y un 70 % para una misma calidad de las secuencias de vídeo reconstruidas, ya que las propuestas se han llevado a cabo sobre el cuantificador y el codificador entrópico. Para

verificar los resultados del codificador mejorado, hemos comparado los resultados del mismo con el estándar MPEG-2 y el codificador EZW. En ambos casos, cuando la calidad de las secuencias de vídeo reconstruidas es la misma para los distintos codificadores, la tasa de compresión del codificador mejorado supera notablemente al resto de codificadores en un rango que varía desde un 21 % hasta un 54 % con respecto a EZW y desde un 51 % hasta un 119 % con respecto al estándar MPEG-2. Además, la calidad obtenida en los vídeos reconstruidos por parte del codificador mejorado es excelente o buena. En la primera, no existen diferencias entre el vídeo original y el reconstruido, mientras que en la segunda la secuencia puede ser aceptable por la comunidad médica porque aunque hay diferencias, no son perceptibles por parte del sistema visual humano.

## Capítulo 4

### La transformada de wavelet 3D en tiempo real

---

#### 4.1. Introducción

En el capítulo anterior hemos desarrollado un codificador basado en la transformada de wavelet 3D que obtiene una gran tasa de compresión y una calidad excelente del vídeo reconstruido, tanto de forma cuantitativa, ya que el PSNR obtiene una calidad buena (38 dB) o excelente (41 dB), como de forma cualitativa, ya que en las secuencias de vídeo reconstruidas no se aprecian diferencias con respecto al original y no aparecen artefactos de bloque, manchas en las imágenes o ruido en los bordes de las mismas.

Sin embargo, el problema principal que surge al utilizar la transformada de wavelet 3D, para codificar y decodificar vídeo médico, es que el tiempo de ejecución es demasiado elevado para que se pueda ejecutar en tiempo real<sup>1</sup> en arquitecturas monoprocesador de propósito general. La utilización de las tres dimensiones, tiempo y espacio (x,y), para obtener una gran tasa de compresión implica que el conjunto de trabajo, es decir, la secuencia de vídeo original, sea demasiado grande y el algoritmo se encuentre totalmente limitado por la cantidad de memoria disponible. De hecho, la frecuencia de visualización de las secuencias de video del codificador mejorado presentado en el capítulo anterior es 7,51 imágenes por segundo.

En este capítulo, desarrollaremos varias propuestas para solucionar dicho problema. En primer lugar, presentaremos varias técnicas de división de los datos en bloques (*blocking*), las cuales se diferencian en la forma en que se divide el conjunto de trabajo inicial, explotando la jerarquía de memoria y el principio de localidad, tanto temporal como espacial, para reducir el tiempo de

---

<sup>1</sup>Definimos tiempo real como la visualización de las secuencias de vídeo a una frecuencia de 24 imágenes por segundo.

ejecución [Bernabé *et al.*, 2002]. La aplicación del *blocking* al cálculo de la transformada de wavelet 3D no es sencillo, si se quiere mantener la tasa de compresión y la calidad de las secuencias de vídeo reconstruidas, es decir, el uso de las técnicas de división en bloques sin el suficiente cuidado y el conocimiento necesario del funcionamiento de la 3D-FWT puede originar la aparición de artefactos de bloque, manchas y ruido en los bordes de las imágenes que componen la secuencia de vídeo o la reducción significativa de la tasa de compresión. Propondremos y evaluaremos varias alternativas para evitar los problemas anteriores y mantener tanto la tasa de compresión como la calidad de las secuencias de vídeo reconstruidas.

A partir de las técnicas de división en bloques propondremos varias optimizaciones para seguir reduciendo el tiempo de ejecución y conseguir la compresión en tiempo real de la transformada de wavelet. Para ello y relacionado con la reducción de la cantidad de memoria necesaria, desarrollaremos el reuso de operaciones en punto flotante para reducir el número de las mismas, y con ello, el número de accesos a memoria [Bernabé *et al.*, 2002].

Por otro lado, la introducción de las extensiones multimedia en los principales procesadores de propósito general tiene como objetivo mejorar el rendimiento de las aplicaciones multimedia. El conjunto de extensiones multimedia se añade al juego de instrucciones básico (ISA) de un procesador para explotar el paralelismo a nivel de datos, ya que operan según un modelo *Single Instruction Multiple Data* (SIMD). Las implementaciones iniciales de las extensiones multimedia estuvieron orientadas al procesamiento de datos de tipo entero como las MMX (MultiMedia eXtensions) [Lempel *et al.*, 1997] de los procesadores Intel y AMD [Favor, 1997], las VIS (*Visual Instruction Set*) [Tremblay *et al.*, 1996] de Sun para los procesadores SPARC, las MAX (*Media Acceleration eXtensions*) [Lee, 1996] de Hewlet Packard para los procesadores PA-RISC, las MVI (*Motion Video Instructions*) [Carlson *et al.*, 1997] de Compaq para los procesadores Alpha y las MDMX (*MIPS Digital Media eXtensions*) [Killian, 1996] para los procesadores MIPS-V. Posteriormente, los procesadores añadieron las extensiones multimedia para punto flotante como las 3DNow [Oberman y Weber, 1999] de los procesadores AMD, las SSE (*Streaming SIMD Extensions*) y SSE2 [Corporation, 2002a] de los procesadores Intel y las AltiVec [Diefendorff *et al.*, 2000] de Motorola para el Power-PC.

La incorporación de las extensiones multimedia a los procesadores de propósito general tiene un coste mínimo. Sin embargo, el uso de las mismas puede generar mejoras significativas en el rendimiento de las aplicaciones multimedia. La relevancia de dicha mejora se hace mayor a medida que aumenta el uso de las aplicaciones multimedia y llegará a convertirse en un aspecto crucial para conseguir que el software multimedia se encuentre altamente optimizado y se pueda obtener rendimiento de las extensiones que incluyen los procesadores. Para ello, es muy importante diseñar

las aplicaciones para que exploten los recursos ofrecidos por los procesadores. En este capítulo realizamos un primer paso para automatizar el proceso de convergencia entre las aplicaciones y los procesadores, a través del uso de las extensiones multimedia y el cálculo de la transformada de wavelet.

Basándonos en el uso de las extensiones multimedia para punto flotante (utilizaremos las extensiones SSE de los procesadores Intel, aunque se puede generalizar para extensiones multimedia equivalentes en otras plataformas), proponemos reducir el tiempo de ejecución de la transformada de wavelet y explotar, por tanto, el paralelismo a nivel de datos. La forma más sencilla de realizar este proceso consiste en que un compilador extraiga el paralelismo de forma automática a partir del análisis del código fuente, es decir, lo que se denomina una vectorización automática. Normalmente, los compiladores actuales analizan las aplicaciones para determinar si pueden o no vectorizar cada uno de los bucles y transformarlo en instrucciones SIMD. En este caso, utilizaremos el nuevo compilador creado por Intel, compilador Intel C/C++ para Linux (v 5.0.1) [Corporation, 2002b]. A pesar de la aparición de compiladores que ofrecen un alto porcentaje de vectorización, es difícil llevar a cabo una vectorización automática y eficiente en el código fuente de cualquier aplicación multimedia como veremos en el caso concreto del cálculo de la transformada de wavelet, debido a las restricciones que impone el compilador para detectar las posibles instrucciones que se pueden vectorizar y, en nuestro caso, a la propia naturaleza del algoritmo de la transformada.

Por tanto y debido a que la vectorización automática no es nada sencilla, proponemos llevar a cabo una vectorización manual para extraer el paralelismo a nivel de datos existente con nuestro propio estudio del código fuente y, en nuestro caso particular, el conocimiento del funcionamiento de la transformada de wavelet [Bernabé *et al.*, 2003] y de las extensiones multimedia disponibles.

Relacionado con el uso de las extensiones multimedia, utilizaremos técnicas clásicas como el desenrollado de bucles y la pre-búsqueda de datos para seguir reduciendo el tiempo de ejecución de la transformada de wavelet, ya que los procesadores disponen de instrucciones para controlar la cache, realizar pre-búsquedas y ordenar las instrucciones [Bernabé *et al.*, 2003].

Finalmente, examinaremos otra vez el código fuente y el comportamiento de la aplicación, para explotar la localidad temporal y espacial de la memoria cache. Desarrollaremos una técnica, denominada *vectorización por columnas*, para mejorar la localidad de la jerarquía de memoria en el cálculo de la transformada de wavelet en las dimensiones  $X$  e  $Y$ , teniendo en cuenta que usamos como función wavelet madre la Daubechies de 4 coeficientes ( $Daub - 4$ ) [Bernabé *et al.*, 2003].

El desarrollo de las diferentes optimizaciones permitirá la compresión y transmisión en tiempo real de la transformada de wavelet 3D cuando se codifica vídeo médico. Este proceso se puede generalizar de forma sencilla para aplicaciones multimedia de características similares. De hecho, las diversas optimizaciones realizadas se podrían integrar como funciones dentro de una librería gráfica para la optimización y mejora del software multimedia existente.

## 4.2. Trabajo relacionado

### 4.2.1. Técnicas de división en bloques

A lo largo del tiempo, las técnicas de división en bloques (*blocking*) se han convertido en una técnica eficiente, conocida y sencilla para mejorar el rendimiento de aplicaciones de tipo científico o numéricas, mediante un uso efectivo de la jerarquía de memoria. Dichas técnicas se pueden aplicar en los diferentes niveles de la jerarquía de memoria, es decir, a nivel de memoria virtual, memoria cache y registros.

El procedimiento consiste en dividir el conjunto de trabajo inicial en varios bloques de pequeño tamaño, permitiendo que los bloques de datos que se encuentran en los niveles más rápidos y cercanos al procesador de la jerarquía de memoria puedan reutilizarse y se reduzca, por tanto, la latencia de acceso a memoria. Posteriormente, se determina el tamaño del bloque óptimo para maximizar el principio de localidad de la jerarquía de memoria.

Cuando se aplican las técnicas de división en bloques, el rendimiento de la memoria cache depende de tres factores: el tamaño del problema, el tamaño del bloque y el tamaño de la propia memoria cache [Lam *et al.*, 1991]. En particular, el rendimiento de la cache mejora cuando el tamaño del bloque cabe en la memoria cache o es una fracción del tamaño de la misma.

Por otro lado, también es muy conocida la utilidad de las técnicas de división en bloques en algoritmos de álgebra lineal como BLAS [Dongarra *et al.*, 1988], LAPACK [Anderson *et al.*, 1990] y ATLAS [Whaley *et al.*, 2001].

Desde el punto de vista de la transformada de wavelet, Xing [1994] desarrolló un modelo basado en multiprocesadores para la compresión de imágenes basado en la transformada de wavelet 2D. Según este esquema, la imagen original se divide en distintas partes que se transmiten de forma simultánea. Los segmentos reconstruidos son interpolados para obtener la imagen final. La calidad y el rendimiento se comparan con modelos de un único procesador obteniendo una calidad superior. A medida que el tamaño de bloque es menor, se produce una mayor presencia de los artefactos de bloque, influyendo en una degradación de la calidad de la imagen.



Posteriormente, y desde el punto de vista de la compresión de imágenes y vídeo usando la transformada de wavelet, las técnicas de división en bloques se han aplicado sobre el algoritmo de compresión de imágenes [Said y Pearlman, 1996b] y vídeo [Kim y Pearlman, 1997] SPIHT. En [Wheeler y Pearlman, 1999], una vez obtenida la descomposición en sub-bandas de los diferentes coeficientes, al aplicar la transformada de wavelet sobre una imagen durante un número determinado de iteraciones, se propone realizar una división en pequeños sub-árboles o bloques que conservan la orientación espacial. Posteriormente, cada uno de los bloques creados se codifica usando el algoritmo SPIHT de manera independiente.

En cuanto a la compresión de vídeo usando la transformada de wavelet, se realizó una implementación eficiente para la compresión de imágenes volumétricas médicas utilizando el algoritmo SPIHT con un consumo bajo de memoria [Kim y Pearlman, 2000]. Para ello, la secuencia de *frames* se divide en pequeñas secuencias, denominadas *stripes*, que se codifican según el algoritmo 3D-SPIHT y se envían separadamente al receptor tras la correspondiente codificación entrópica basada en un codificador aritmético. Sin embargo, la división anterior introduce artefactos en los bordes y reduce la eficiencia del código a medida que el tamaño del *stripe* disminuye. Los resultados obtenidos son inferiores a 3D-SPIHT, ya que el pico de la relación señal/ruído disminuye entre 0,6 y 1,65 dB, aunque este descenso no es significativo y permite utilizar el nuevo método con un consumo de memoria inferior al algoritmo original.

Por tanto, el uso de las técnicas de división en bloques para el cálculo de la transformada de wavelet 3D es una tarea complicada, pues afecta a la tasa de compresión o a la calidad del vídeo reconstruido [Kim y Pearlman, 2000]. Al aplicar las técnicas de *blocking*, sin tener en cuenta la forma de realizar el cálculo de la transformada de wavelet 3D, si se desea mantener la calidad alcanzada entonces la tasa de compresión disminuye, mientras que si se pretende mantener la tasa de compresión entonces la calidad disminuye.

#### 4.2.2. Extensiones Multimedia

En los últimos años, los principales procesadores de propósito general han añadido, a su juego de instrucciones básico (ISA), un conjunto de extensiones multimedia para explotar el paralelismo a nivel de datos SIMD, y disponer de tipos de datos e instrucciones básicas que permitan ejecutar eficientemente las aplicaciones multimedia. Entre otros, podemos destacar las extensiones 3DNow de AMD [Favor, 1997][Oberman y Weber, 1999], las VIS [Tremblay *et al.*, 1996] de Sun para los procesadores SPARC, las MAX [Lee, 1996] de Hewlett Packard para los procesadores PA-RISC, las AltiVec [Diefendorff *et al.*, 2000] de Motorola para el PowerPC, las

extensiones multimedia para enteros (MMX) [Lempel *et al.*, 1997] y las extensiones SIMD para punto flotante (SSE) [Corporation, 2002a] de Intel para los procesadores de la familia x86. Debido a la plataforma empleada durante el desarrollo del trabajo, en esta tesis nos centraremos en el uso de las extensiones multimedia de los procesadores Intel para la consecución de nuestros objetivos, aunque su uso y programación se puede generalizar a cualquiera de las extensiones multimedia de los citados procesadores.

La aparición de las extensiones SSE se produce con la llegada al mercado del procesador Pentium III [Thakkar y Huff, 1999], el cual añade a la arquitectura del procesador 8 registros de datos de 128 bits, denominados desde *XMM0* hasta *XMM7*. Para crear los nuevos registros se añade un nuevo estado al procesador. La creación del nuevo estado implica reducir la complejidad de la implementación, facilitar el modelo de programación y aumentar el tamaño del chip del procesador en un 10 %. Además, se permite el uso de forma concurrente de las instrucciones *SIMD* para punto flotante y de las instrucciones *MMX* o *x87*. Las instrucciones SSE pueden ejecutar operaciones en paralelo sobre 4 números en punto flotante de simple precisión o 32 bits guardados en los registros XMM, denominadas instrucciones de anchura 4, y sobre 1, 2, 4 u 8 enteros almacenados en los registros MMX. El procesador implementa cada macro-instrucción de anchura 4 como dos micro-instrucciones de 64 bits, aunque como dispone de una implementación superescalar, cada operación SIMD de anchura 4 se puede ejecutar mediante un solo ciclo de reloj. De esta forma, las aplicaciones pueden obtener un *speed-up* teórico de 4, aunque el *speed-up* real que puede conseguir un programa determinado es 2 debido, en parte, a la presión de la micro-instrucción en el interior de la micro-arquitectura. El número de nuevas instrucciones que se añaden al juego de instrucciones del procesador es 70, las cuales se dividen en cuatro grupos:

- Instrucciones SIMD para punto flotante de simple precisión que operan sobre los registros XMM: transferencia de datos, aritmética empaquetada, comparación, lógicas, mezcla, desempaquetado y conversión.
- Instrucciones de manejo y control del registro de estado MXSCR.
- Instrucciones SIMD de 64 bits para enteros que operan sobre los registros MMX.
- Instrucciones para el control de la cache, pre-búsqueda y ordenación de instrucciones.

El objetivo fundamental de las extensiones multimedia es aumentar el rendimiento de una aplicación secuencial. Sin embargo la utilización de las mismas no es sencilla, ya que implican la utilización de un modelo de programación complejo que a veces no viene acompañado del adecuado soporte software. En [Conte *et al.*, 2000] se introduce el modelo de programación de las

extensiones MMX/SSE de Intel y se determina cómo mejorar el rendimiento de dos aplicaciones de procesamiento de imágenes en tiempo real mediante la utilización de dichas extensiones.

En [Nachtergaele *et al.*, 2000] se propone una implementación de MPEG-4 que utiliza las extensiones MMX y usa la transformada de wavelet basada en enteros sobre un Pentium-II. De esta forma, se consigue transmitir video QCIF (imágenes de 176x144 pixels de 24 bits) en tiempo real.

De forma más general, en [Ranganathan *et al.*, 1999] se analiza el rendimiento de varias aplicaciones de procesamiento de imágenes y vídeo utilizando las extensiones VIS de Sun, sobre dos modelos típicos de procesadores de propósito general, en orden y fuera de orden. Además, se identifican los beneficios y limitaciones del uso de las extensiones VIS, así como la influencia de las técnicas tradicionales de paralelismo a nivel de instrucción sobre las aplicaciones, el comportamiento de la memoria y las técnicas de pre-búsqueda a nivel de software.

Para utilizar las extensiones multimedia MMX/SSE se necesitan lenguajes de alto nivel y compiladores que soporten dichas extensiones. Al principio, la única forma posible de desarrollar aplicaciones que utilizaban la tecnología SIMD era a nivel de lenguaje ensamblador, mediante ficheros en ensamblador x86 o funciones *inline* en C o C++. Por ello, Intel desarrolló con la llegada del Pentium III al mercado, el compilador Intel C/C++, el analizador de rendimiento VTune y el juego de librerías de rendimiento de Intel, las cuales proporcionan a los programadores los siguientes métodos de programación [Wolf, 1999]:

- Funciones esenciales. Se trata de funciones en C/C++ a partir de las cuales el compilador genera automáticamente el código SIMD correspondiente. Permiten la utilización de tipos de datos de 128 bits o 64 bits para llevar a cabo diversas operaciones.
- Librería de clases de vector. Se trata de una abstracción en C++ de las funciones esenciales.
- Vectorización automática. El compilador realiza búsquedas de bucles que operan sobre arrays de caracteres, enteros o reales para transformarlos en bucles más eficientes utilizando las instrucciones SIMD. En [Bik *et al.*, 2001] se describe la forma de llevar a cabo una paralelización y una vectorización automática utilizando el compilador Intel C/C++ sobre procesadores Pentium III y Pentium IV.
- Juego de librerías de rendimiento de Intel. Se trata de una serie de librerías que hacen uso de las extensiones SIMD para el desarrollo de algoritmos ampliamente utilizados. Se dispone de librerías para el procesamiento de la señal, para el procesamiento de imágenes, para el

reconocimiento de primitivas, para el desarrollo de operaciones matemáticas y una librería de funciones relacionadas con JPEG.

- El analizador de rendimiento VTune permite analizar el rendimiento de una aplicación e informa sobre el comportamiento de la misma, con respecto al procesador Pentium III, para poder mejorar las prestaciones de dicha aplicación.

Con respecto al analizador de rendimiento VTune, el desarrollo de esta herramienta viene determinada por la existencia en la mayoría de los procesadores actuales de una serie de contadores de rendimiento que permiten comprender como se está ejecutando una aplicación en una máquina determinada. El análisis de los valores de dichos contadores permiten determinar los aspectos que se deben mejorar en un programa determinado. Sin embargo, y en el momento del desarrollo de las propuestas descritas en este capítulo, la herramienta VTune (v.4.0) no estaba disponible para el sistema operativo Linux, por lo que decidimos utilizar una librería denominada Rabbit (v.2.0.1) [Heller, 2001] que nos permite leer y manipular los contadores del Pentium III usando el lenguaje de programación C y bajo el sistema operativo Linux.

La familia de procesadores Intel P6, que incluye los procesadores Pentium Pro, Pentium II y Pentium III, incluyen dos contadores de rendimiento de 40 bits con una lista de eventos y una semántica adicional que depende de cada procesador en particular. Estos contadores nos permiten medir de forma simultánea dos tipos diferentes de eventos. Se puede medir el número de veces que se produce un determinado evento o el número de ciclos del procesador que transcurren, mientras una determinada condición es verdadera, es decir, mientras se cumple un determinado evento. La lista de eventos nos permite controlar cada una de las partes del procesador, pudiendo obtener el rendimiento de la cache de datos L1, la unidad de búsqueda de instrucciones, la cache L2, el bus externo, la unidad de punto flotante, la unidad de memoria, la unidad de decodificación y ejecución de instrucciones, las interrupciones, los saltos, las paradas o detenciones, el número de registros de segmento leídos, los ciclos de reloj, la unidad de MMX/SSE y el número de operaciones relacionadas con el re-nombramiento de registros de segmento.

### 4.3. Técnicas de división en bloques

Cuando se desea comprimir o transmitir vídeo en tiempo real, necesitamos procesar 24 *frames* por segundo, por lo que una secuencia de vídeo de 64 *frames* de 512x512 pixels, es decir 2,5 segundos, ocuparía 16 Mbytes. El gran tamaño de estas secuencias que utilizan la transformada de wavelet 3D limita el rendimiento del algoritmo de compresión e impide la compresión y transmi-

sión de vídeo en tiempo real. Resultados preliminares demuestran que el algoritmo de compresión se encuentra totalmente limitado por la memoria disponible, por lo que las técnicas de división en bloques pueden ser apropiadas para reducir los requerimientos de memoria, y por lo tanto, el tiempo de ejecución, ya que el codificador mejorado desarrollado en el capítulo 3 tiene un tiempo de ejecución que permite una frecuencia de visualización de tan sólo 7,51 imágenes por segundo.

El objetivo de los algoritmos de división en bloques consiste en explotar la localidad temporal y espacial de las diferentes referencias a memoria. Para ello, se divide el vídeo original en bloques más pequeños, de forma que estos bloques caben dentro de los diferentes niveles de la jerarquía de memoria. Con ello se obtienen dos mejoras significativas: en primer lugar, se aceleran los accesos a memoria ya que los datos se encuentran en niveles más altos de la jerarquía de memoria y, por lo tanto, más cercanos al procesador y, en segundo lugar, se reduce significativamente el tráfico de información entre la memoria principal y el procesador, por lo que se obtiene un mejor uso del ancho de banda proporcionado por el sistema base del ordenador.

Sin embargo, la aplicación del *blocking* puede no ser efectiva en el caso del cálculo de la transformada de wavelet 3D, ya que puede afectar a la calidad del vídeo reconstruido y a la más que posible aparición de artefactos de bloque, manchas en las imágenes o ruido en los bordes de las mismas, al dividir el vídeo original en diferentes bloques. Por todo ello, se proponen y evalúan dos técnicas, dependiendo de la forma en la que se divide el vídeo original: división en cubos y división en rectángulos.

#### 4.3.1. Técnicas de división en cubos

En esta técnica proponemos dividir la secuencia de vídeo original en varios cubos, como se puede observar en la figura 4.1, a los que se le aplica la transformada de wavelet 3D de forma independiente. El tamaño de los diferentes sub-cubos en los que se divide la secuencia original va a ser el mismo en los ejes  $X$  e  $Y$ , donde evaluaremos diferentes tamaños de bloque, mientras que en el eje del tiempo se establece un número fijo de *frames* a 16, ya que es el número mínimo de *frames* necesario para poder aplicar la transformada de wavelet dos veces con la función wavelet madre *Daub - 4*.

La principal desventaja de esta técnica es la aparición de los artefactos de bloque en los bordes de los diferentes sub-cubos adyacentes, provocando una disminución significativa del valor pico de la relación señal/ruido (PSNR). Esto se debe a la forma en que se calcula la transformada de wavelet, ya que dado un determinado pixel, el valor del coeficiente wavelet correspondiente una vez aplicada la transformada depende de los valores originales de los pixels que se encuentran

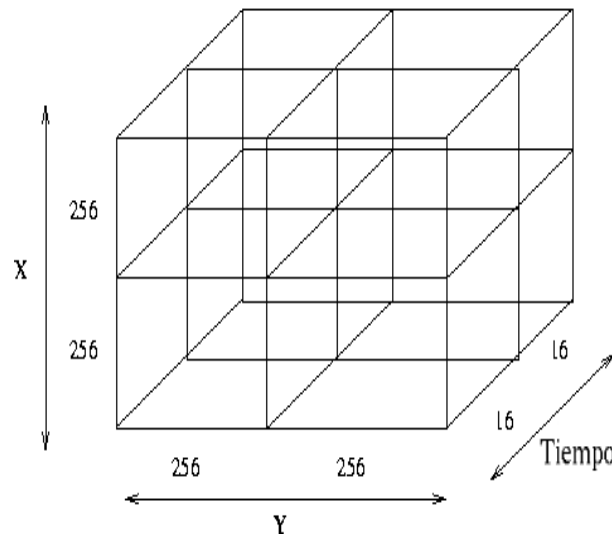


Figura 4.1: Técnicas de división en cubos

cercanos al mismo.

Para ilustrar mejor este problema, la figura 4.2 muestra como se aplica la transformada de wavelet a una señal de una dimensión de 8 pixels usando la función wavelet madre *Daub - 4*. El algoritmo divide la señal original en dos bloques de cuatro pixels a los que se le calcula la transformada de wavelet independientemente. Dado el primer bloque de cuatro pixels, el coeficiente wavelet para el primer pixel depende de los pixels primero, segundo, tercero y cuarto, todos ellos pertenecientes a este mismo bloque. Sin embargo, el coeficiente wavelet para el segundo pixel depende de los pixels tercero, cuarto, quinto y sexto, donde los dos últimos pixels pertenecen al segundo bloque y no se encuentran disponibles en el bloque actual. De la misma forma, ocurre con el resto de pixels de este primer bloque. Por lo tanto, cuando se aplica un algoritmo de división en bloques tradicional sobre la transformada de wavelet se necesitan pixels de bloques adyacentes para calcular la transformada en un bloque determinado. Para solucionar este problema aparecen dos alternativas: transformada de wavelet no-solapada y solapada.

La transformada de wavelet no solapada se implementa de forma que cuando se necesitan pixels que no se encuentran en el mismo bloque que se está procesando, se utilizan los últimos o los primeros pixels de una fila, columna o *frame* del mismo bloque, para poder realizar el cálculo de la transformada. De esta forma, para calcular la transformada de wavelet en el tiempo es necesario mantener en memoria los dos primeros o últimos *frames*. Este hecho influye significativamente en el rendimiento de la jerarquía de memoria, ya que implica mantener en memoria una serie de *frames* ya procesados durante un tiempo determinado. Por lo tanto, y de una forma más

```

/* c0, c1, c2, c3: Coeficientes Daub-4 */
/* pixels 1..8 = p[0..7] */
float bajo[8], alto[8];
n = 8;
for(i = 0, j = 0; j < (n/2) - 1; i += 2, j++) {

    bajo[j]=c0*p[i]+c1*p[i+1]+c2*p[i+2]+c3*p[i+3];
    alto[j+n/2]=c3*p[i]-c2*p[i+1]+c1*p[i+2]-c0*p[i+3];

}
bajo[j]=c0*p[n-2]+c1*p[n-1]+c2*p[0]+c3*p[1];
alto[j+n/2]=c3*p[n-2]-c2*p[n-1]+c1*p[0]-c0*p[1];

```

Figura 4.2: Algoritmo de la Transformada de Wavelet 1D con la función Daub-4

natural, es mejor no tener que almacenar dichos *frames* y usar los *frames* del siguiente bloque para mantener una continuidad en el cálculo de la transformada. Por todo ello, proponemos una segunda opción denominada transformada de wavelet solapada, en la cual siempre se usan pixels del siguiente bloque. Esta última opción aumentará la tasa de compresión, la calidad visual de la secuencia de vídeo reconstruida y explotará mejor la localidad de la jerarquía de memoria, como podremos observar posteriormente.

Otro aspecto a tener en cuenta para el cálculo de la transformada de wavelet solapada o no-solapada, es el número de filas, columnas o *frames* que nos hacen falta del siguiente bloque o del mismo bloque. Este número depende del número de veces que se aplica la transformada de wavelet y del número de coeficientes de la función wavelet madre. Por ejemplo, con la función wavelet madre *Daub - 4*, y aplicando la transformada de wavelet una vez, se necesitan dos filas, dos columnas o dos *frames* del siguiente o del mismo bloque. Si aplicamos la transformada dos veces, el número necesario de filas, columnas o *frames* asciende a seis, mientras que si la transformada se aplica tres veces el número necesario se eleva a catorce.

La elección entre las dos propuestas para el cálculo de la transformada, no-solapada y solapada, es una de las principales decisiones para obtener una buena relación entre el tiempo de ejecución y la calidad del vídeo reconstruido. Mientras que la opción no-solapada, parece más eficiente desde el punto de vista de la jerarquía de memoria, ya que todos los cálculos se realizan utilizando sólo los pixels del bloque que se está procesando actualmente, la calidad del vídeo reconstruido se ve

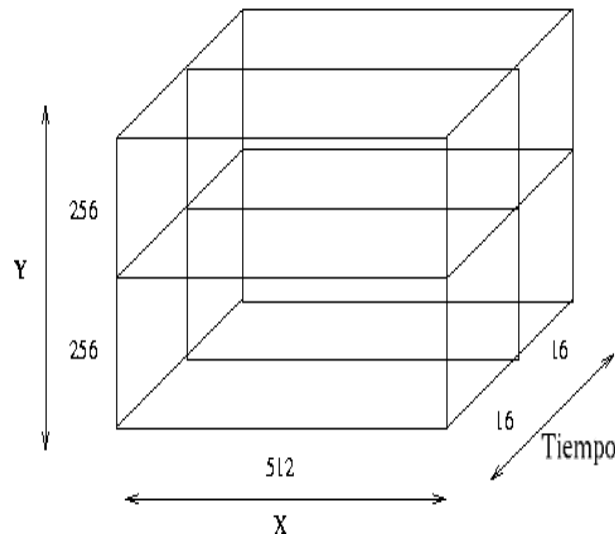


Figura 4.3: Técnicas de división en rectángulos

claramente afectada por la aparición de los artefactos de bloque en los bordes de los diferentes bloques en los que se ha dividido la secuencia original, debido a que el cálculo de los coeficientes wavelet en las proximidades de los bordes se lleva a cabo sin tener en cuenta el valor de los pixels cercanos.

Por todo ello, y con el fin de evitar la aparición de los artefactos de bloque, proponemos utilizar la opción solapada para los tres ejes,  $X$ ,  $Y$  y tiempo, en los que se calcula la transformada de wavelet. Por ejemplo, si aplicamos la transformada de wavelet dos veces, utilizamos como función wavelet madre la *Daub - 4*, y dividimos en sub-cubos de 16 *frames* de  $256 \times 256$  pixels, necesitaremos solapar para cada sub-cubo 6 filas, 6 columnas y 6 *frames* del siguiente sub-cubo, por lo que tendremos, finalmente, sub-cubos de 22 *frames* de  $262 \times 262$  pixels.

#### 4.3.2. Técnicas de división en rectángulos

El lenguaje utilizado para la programación de la transformada de wavelet 3D es el lenguaje C, por lo que los *frames* se almacenan en memoria siguiendo un orden basado en las filas que componen cada uno de los *frames*. Con el objetivo de explotar la localidad espacial de las referencias a memoria, proponemos analizar una distribución diferente a la división en cubos propuesta anteriormente. En este caso se divide la secuencia original en varios rectángulos, como podemos observar en la figura 4.3, para explotar mejor la localidad espacial.

Además, y con la finalidad de evitar la aparición de los artefactos de bloque y la degradación



de la calidad del vídeo reconstruido, aplicamos la transformada de wavelet solapada. Al realizar una división en rectángulos, sólo tenemos que solapar en las dimensiones  $Y$  y tiempo, ya que la dimensión  $X$  siempre permanece fija con el valor original de la secuencia original. Por ejemplo, dada una secuencia de vídeo de 64 *frames* de  $512 \times 512$  pixels, podemos dividirla en 8 rectángulos de 16 *frames* de  $512 \times 256$  pixels o 32 rectángulos de 16 *frames* de  $512 \times 128$  pixels, y aplicando la transformada de wavelet solapada dos veces en las dimensiones correspondientes con la función wavelet madre Daub-4, nos quedarían rectángulos de 22 *frames* de  $512 \times 262$  pixels o rectángulos de 22 *frames* de  $512 \times 134$  pixels.

## 4.4. Optimización de la técnica de división en rectángulos usando la transformada de wavelet solapada

En este apartado presentamos varias propuestas y optimizaciones, a partir de las técnicas de división en bloques aplicadas sobre la transformada de wavelet 3D, con el objetivo de reducir por un lado el número de instrucciones en punto flotante ejecutadas y, por otro lado, el número de accesos a memoria que se producen. Ambos factores implican una reducción significativa del tiempo de ejecución de la transformada de wavelet 3D.

### 4.4.1. Reuso de cálculos

Una vez que aplicamos las técnicas de división en bloques y la transformada de wavelet 3D solapada, proponemos reutilizar los cálculos ya realizados en un bloque, ya que se van a necesitar en el bloque siguiente, para reducir el tiempo de ejecución. Por ejemplo, dada una secuencia de vídeo de 64 *frames* de  $512 \times 512$  pixels que se divide en 8 rectángulos de 16 *frames* de  $512 \times 256$  pixels, se necesita solapar en cada uno de los mencionados rectángulos 6 filas y 6 *frames*. Cuando la transformada de wavelet se aplica por primera vez en la dimensión  $Y$  se obtienen 130 filas pasa-baja y 130 filas pasa-alta. Las dos últimas filas pasa-baja y pasa-alta son las dos primeras pasa-baja y pasa-alta del siguiente rectángulo, que por lo tanto no sería necesario volverlas a calcular, como podemos observar en la figura 4.4.

La cantidad de cálculos que podemos reutilizar aumenta a medida que dividimos en rectángulos de menor tamaño. Por ejemplo, si dividimos una secuencia de vídeo de 64 *frames* de  $512 \times 512$  pixels en rectángulos de 16 *frames* de  $512 \times 32$  pixels, en la dimensión  $Y$  tendremos ya realizados el 12 % de los cálculos. Por otro lado, si dividimos en rectángulos de 16 *frames* de  $512 \times 16$  pixels, el porcentaje de operaciones de la transformada de wavelet que ya se han calculado, y que se

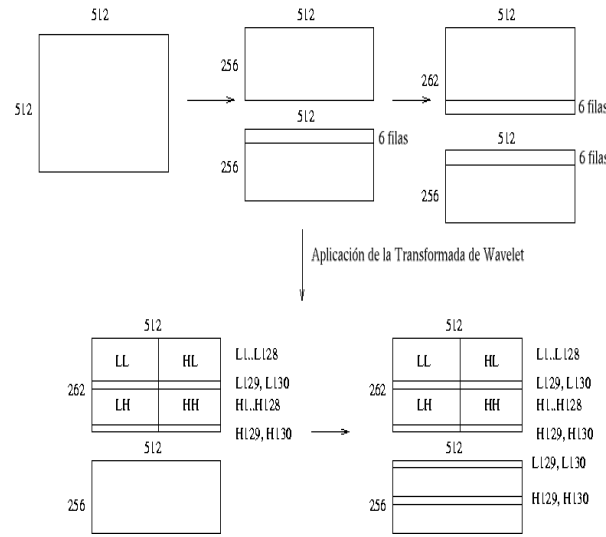


Figura 4.4: Reutilización en las técnicas de división en rectángulos

pueden volver a utilizar en el siguiente bloque, asciende al 25 %. Esta propuesta constituye una variante software del reuso de instrucciones por hardware propuesto por Sodani y Sohi [1997].

#### 4.4.2. Uso de las Extensiones SIMD (SSE)

La introducción de las extensiones multimedia MMX para enteros y de las extensiones SIMD para operaciones en punto flotante en los microprocesadores actuales, proporciona una nueva tecnología diseñada para la aceleración del software multimedia y de las comunicaciones, que permite reducir el tiempo de ejecución de las aplicaciones multimedia. Por todo ello, uno de los objetivos de esta tesis es evaluar el uso de las extensiones SIMD introducidas tanto de forma automática como manual. En la primera opción, un compilador analiza el código fuente y extrae el paralelismo a nivel de datos de forma automática (vectorización automática). En la segunda opción, el programador examina el código fuente e intenta explotar el paralelismo a nivel de datos mediante la agrupación de operaciones en punto flotante en instrucciones SIMD (vectorización manual).

Las extensiones SIMD para operaciones en punto flotante se utilizan para la vectorización de bucles explotando el paralelismo de datos, ya que realizan la misma operación sobre múltiples elementos de un conjunto de datos. Por lo tanto, podemos aplicar el uso de las extensiones SIMD para el algoritmo de la transformada de wavelet solapada 1D sobre  $n$  pixels, usando la función Daub-4 como wavelet madre.

```

/* c0, c1, c2, c3: Coeficientes Daub-4 */
/* pixels 1..8 = p[0..7] */
float bajo[8];

bajo[0]=c0*p[0]+c1*p[1]+c2*p[2]+c3*p[3];
bajo[1]=c0*p[2]+c1*p[3]+c2*p[4]+c3*p[5];
bajo[2]=c0*p[4]+c1*p[5]+c2*p[6]+c3*p[7];
bajo[3]=c0*p[6]+c1*p[7]+c2*p[8]+c3*p[9];

```

Figura 4.5: Cálculo de los cuatro primeros coeficientes wavelet pasa-baja.

Como podemos observar en la figura 4.2, el valor de cada coeficiente wavelet para un determinado pixel depende de cuatro pixels. Por lo tanto, para calcular cada uno los coeficientes pasa-bajo y pasa-alto, tenemos que realizar 8 multiplicaciones y 6 sumas en punto flotante. Por todo ello, para cuatro pixels necesitamos llevar a cabo 32 multiplicaciones y 24 sumas en punto flotante para obtener los correspondientes coeficientes pasa-bajo y pasa-alto, tal y como observar en la figura 4.5 para una señal de una dimensión de  $n$  pixels.

#### 4.4.2.1. Vectorización automática

El compilador *Intel C/C++* para Linux (v 5.0.1) [Corporation, 2002b] es capaz de realizar paralelización y vectorización automática. Dicho compilador se ha desarrollado siguiendo una propuesta estándar para la vectorización automática de bucles anidados [Wolfe, 1996]. En primer lugar, las instrucciones que pertenecen a un bucle se ordenan siguiendo una ordenación topológica acíclica y condensada del grafo de dependencias de datos del bucle. A continuación, se identifican las instrucciones que se encuentran en un ciclo de la dependencia de datos para que puedan ser vectorizadas o distribuidas en un bucle que se ejecutará secuencialmente. Finalmente, los bucles vectorizados se transforman en instrucciones SIMD [Bik *et al.*, 2001].

#### 4.4.2.2. Vectorización manual

A continuación, vamos a ver cómo realizamos el cálculo de los cuatro primeros coeficientes pasa-baja para una señal de una dimensión de  $n$  pixels, mediante una vectorización manual en la

que utilizaremos las extensiones SIMD para operaciones en punto flotante tal y como se ilustra en la figura 4.6.

En primer lugar, cargamos el valor de cada uno de los coeficientes de la función wavelet madre  $Daub - 4$ , es decir,  $C0$ ,  $C1$ ,  $C2$  y  $C3$  en los cuatro primeros registros SIMD,  $XMM0$ ,  $XMM1$ ,  $XMM2$  y  $XMM3$ . Por lo tanto, cada registro SIMD de 128 bits queda cargado con cuatro números en punto flotante de simple precisión o 32 bits, o lo que es lo mismo, en cada registro SIMD se guarda de forma cuatriplicada el valor de cada uno de los coeficientes de la función wavelet madre  $Daub - 4$  (figura 4.6-a).

A continuación, en los registros restantes SIMD,  $XMM4$ ,  $XMM5$ ,  $XMM6$  y  $XMM7$ , cargamos, en grupos de cuatro, los valores de los pixels necesarios para realizar los cálculos correspondientes, es decir, desde el primer pixel hasta el décimo pixel de la señal original (figura 4.6-b).

Una vez que tenemos todos los datos necesarios en los registros SIMD, realizamos cuatro multiplicaciones entre los registros SIMD para obtener los sumandos de cada uno de los coeficientes wavelet pasa-baja (figura 4.6-c).

Finalmente, sólo nos queda realizar tres sumas entre los registros SIMD para obtener los coeficientes wavelet pasa-baja. (figura 4.6-d) ). De esta forma, obtenemos los mismos valores finales que en el algoritmo de la transformada de wavelet solapada 1D usando como función wavelet madre la  $Daub - 4$ . Sin embargo, en este caso hemos realizado un número significativo menor de operaciones en punto flotante.

De forma análoga a la que hemos aplicado para obtener los coeficientes wavelet pasa-baja se pueden obtener los coeficientes wavelet pasa-alta correspondientes. Se realiza el mismo número de operaciones en punto flotante con los registros SIMD, es decir, cuatro multiplicaciones y tres sumas. Además, el cálculo de los coeficientes depende, al igual que en el caso de los cuatro primeros coeficientes pasa-baja, de los de los diez primeros pixels de la señal.

Por lo tanto, el uso de las extensiones SIMD nos permite reducir significativamente el número de instrucciones en punto flotante para el cálculo de los coeficientes wavelet pasa-baja y pasa-alta. Por ejemplo, para calcular los coeficientes de los cuatro primeros pixels usando las extensiones SIMD necesitamos sólo 15 instrucciones frente a las 56 necesarias con el algoritmo de la transformada de wavelet 1D con la función wavelet madre  $Daub - 4$ .

Para la implementación de la vectorización manual usando los registros SIMD que acabamos de exponer, hemos utilizado las funciones esenciales que podemos observar en la parte derecha de la figura 4.6. A partir de estas funciones en C, el compilador Intel C/C++ genera automáticamente el código SIMD correspondiente, evitando que el programador tenga que manejar directamente el

XMM0	<table><tr><td>C0</td><td>C0</td><td>C0</td><td>C0</td></tr></table>	C0	C0	C0	C0	<code>_mm_set_ps(C0, C0, C0, C0)</code>
C0	C0	C0	C0			
XMM1	<table><tr><td>C1</td><td>C1</td><td>C1</td><td>C1</td></tr></table>	C1	C1	C1	C1	<code>_mm_set_ps(C1, C1, C1, C1)</code>
C1	C1	C1	C1			
XMM2	<table><tr><td>C2</td><td>C2</td><td>C2</td><td>C2</td></tr></table>	C2	C2	C2	C2	<code>_mm_set_ps(C2, C2, C2, C2)</code>
C2	C2	C2	C2			
XMM3	<table><tr><td>C3</td><td>C3</td><td>C3</td><td>C3</td></tr></table>	C3	C3	C3	C3	<code>_mm_set_ps(C3, C3, C3, C3)</code>
C3	C3	C3	C3			
a) Carga de los registros SIMD con los coeficientes de la Daub-4.						
XMM4	<table><tr><td>p[0]</td><td>p[2]</td><td>p[4]</td><td>p[6]</td></tr></table>	p[0]	p[2]	p[4]	p[6]	<code>_mm_set_ps(p[6], p[4], p[2], p[0])</code>
p[0]	p[2]	p[4]	p[6]			
XMM5	<table><tr><td>p[1]</td><td>p[3]</td><td>p[5]</td><td>p[7]</td></tr></table>	p[1]	p[3]	p[5]	p[7]	<code>_mm_set_ps(p[7], p[5], p[3], p[1])</code>
p[1]	p[3]	p[5]	p[7]			
XMM6	<table><tr><td>p[2]</td><td>p[4]</td><td>p[6]</td><td>p[8]</td></tr></table>	p[2]	p[4]	p[6]	p[8]	<code>_mm_set_ps(p[8], p[6], p[4], p[2])</code>
p[2]	p[4]	p[6]	p[8]			
XMM7	<table><tr><td>p[3]</td><td>p[5]</td><td>p[7]</td><td>p[9]</td></tr></table>	p[3]	p[5]	p[7]	p[9]	<code>_mm_set_ps(p[9], p[7], p[5], p[3])</code>
p[3]	p[5]	p[7]	p[9]			
b) Carga de los valores de los pixels, en grupos de 4, en los registros SIMD.						
XMM0	<table><tr><td>C0*p[0]</td><td>C0*p[2]</td><td>C0*p[4]</td><td>C0*p[6]</td></tr></table>	C0*p[0]	C0*p[2]	C0*p[4]	C0*p[6]	<code>mulps xmm0, xmm4</code>
C0*p[0]	C0*p[2]	C0*p[4]	C0*p[6]			
XMM1	<table><tr><td>C1*p[1]</td><td>C1*p[3]</td><td>C1*p[5]</td><td>C1*p[7]</td></tr></table>	C1*p[1]	C1*p[3]	C1*p[5]	C1*p[7]	<code>mulps xmm1, xmm5</code>
C1*p[1]	C1*p[3]	C1*p[5]	C1*p[7]			
XMM2	<table><tr><td>C2*p[2]</td><td>C2*p[4]</td><td>C2*p[6]</td><td>C2*p[8]</td></tr></table>	C2*p[2]	C2*p[4]	C2*p[6]	C2*p[8]	<code>mulps xmm2, xmm6</code>
C2*p[2]	C2*p[4]	C2*p[6]	C2*p[8]			
XMM3	<table><tr><td>C3*p[3]</td><td>C3*p[5]</td><td>C3*p[7]</td><td>C3*p[9]</td></tr></table>	C3*p[3]	C3*p[5]	C3*p[7]	C3*p[9]	<code>mulps xmm3, xmm7</code>
C3*p[3]	C3*p[5]	C3*p[7]	C3*p[9]			
c) Multiplicaciones en punto flotante entre los registros SIMD.						
XMM0	<table><tr><td>C0*p[0] + C1*p[1]</td><td>C0*p[2] + C1*p[3]</td><td>C0*p[4] + C1*p[5]</td><td>C0*p[6] + C1*p[7]</td></tr></table>	C0*p[0] + C1*p[1]	C0*p[2] + C1*p[3]	C0*p[4] + C1*p[5]	C0*p[6] + C1*p[7]	<code>addps xmm0, xmm1</code>
C0*p[0] + C1*p[1]	C0*p[2] + C1*p[3]	C0*p[4] + C1*p[5]	C0*p[6] + C1*p[7]			
XMM0	<table><tr><td>C0*p[0] + C1*p[1] + C2*p[2]</td><td>C0*p[2] + C1*p[3] + C2*p[4]</td><td>C0*p[4] + C1*p[5] + C2*p[6]</td><td>C0*p[6] + C1*p[7] + C2*p[8]</td></tr></table>	C0*p[0] + C1*p[1] + C2*p[2]	C0*p[2] + C1*p[3] + C2*p[4]	C0*p[4] + C1*p[5] + C2*p[6]	C0*p[6] + C1*p[7] + C2*p[8]	<code>addps xmm0, xmm2</code>
C0*p[0] + C1*p[1] + C2*p[2]	C0*p[2] + C1*p[3] + C2*p[4]	C0*p[4] + C1*p[5] + C2*p[6]	C0*p[6] + C1*p[7] + C2*p[8]			
XMM0	<table><tr><td>C0*p[0] + C1*p[1] + C2*p[2] + C3*p[3]</td><td>C0*p[2] + C1*p[3] + C2*p[4] + C3*p[5]</td><td>C0*p[4] + C1*p[5] + C2*p[6] + C3*p[7]</td><td>C0*p[6] + C1*p[7] + C2*p[8] + C3*p[9]</td></tr></table>	C0*p[0] + C1*p[1] + C2*p[2] + C3*p[3]	C0*p[2] + C1*p[3] + C2*p[4] + C3*p[5]	C0*p[4] + C1*p[5] + C2*p[6] + C3*p[7]	C0*p[6] + C1*p[7] + C2*p[8] + C3*p[9]	<code>addps xmm0, xmm3</code>
C0*p[0] + C1*p[1] + C2*p[2] + C3*p[3]	C0*p[2] + C1*p[3] + C2*p[4] + C3*p[5]	C0*p[4] + C1*p[5] + C2*p[6] + C3*p[7]	C0*p[6] + C1*p[7] + C2*p[8] + C3*p[9]			
d) Sumas en punto flotante entre los registros SIMD.						

Figura 4.6: Pasos para calcular los cuatro primeros coeficientes wavelet pasa-baja usando los registros SIMD

código en ensamblador. Además, se puede observar la simplicidad de dichas funciones esenciales para realizar el trabajo deseado en cada uno de los pasos del cálculo de los cuatro primeros coeficientes wavelet pasa-baja.

#### 4.4.3. Desenrollado de bucles y pre-búsqueda de datos

Otra forma de reducir el tiempo de ejecución de la transformada de wavelet 3D es la aplicación de técnicas clásicas como desenrollado de bucles y pre-búsqueda. Se realiza un estudio en cada una de las dimensiones, espacio (x,y) y tiempo, para aplicar las técnicas mencionadas anteriormente.

El compilador aplica de forma automática el desenrollado de bucles siempre que estime que puede obtener una mejora con respecto al algoritmo original. En el algoritmo de la transformada

de wavelet 3D para realizar los cálculos en la dimensión tiempo tenemos tres bucles anidados, ya que la transformada en el tiempo se aplica para todos los pixels que se encuentran en la misma posición  $(x, y)$  de cada uno de los *frames* que componen la secuencia de vídeo original.

Sin embargo, y debido a la propia naturaleza del algoritmo wavelet para la dimensión en el tiempo, el compilador es incapaz de desenrollar dicho bucle de forma automática. Por lo tanto, proponemos desenrollar dicho bucle de forma manual. Por ejemplo, si aplicamos la transformada de wavelet dos veces para un bloque de 16 *frames* y elegimos la opción solapada para el calculo de la transformada, entonces para la dimensión tiempo la primera iteración se aplica sobre 22 *frames* y la segunda iteración sobre 10 *frames*. Si además, hacemos uso de los registros SIMD para operaciones en punto flotante, tal y como hemos mostrado en el apartado anterior, el bucle para calcular la transformada en el tiempo para la primera iteración se desglosa en tres secuencias de instrucciones idénticas que calculan cuatro coeficientes pasa-baja y otros cuatro coeficientes pasa-alta, a las que hay que suministrar los pixels de entrada correspondientes. Para la segunda iteración de la transformada en el tiempo sólo se necesita una secuencia de instrucciones que calcule cuatro coeficientes pasa-baja y cuatro coeficientes pasa-alta, y que tenga como datos de entrada los coeficientes pasa-baja que se obtuvieron como resultado de la primera iteración.

La pre-búsqueda de datos permite al procesador disponer de los datos en el momento que los necesita, mientras se están realizando otras operaciones, y ocultar parte de la latencia de acceso a memoria, siempre que se haga una predicción acertada de la páginas de memoria que vamos a necesitar en un futuro para trasladarlas a la memoria caché, antes de que el programa realice las peticiones correspondientes. En el cálculo de la transformada de wavelet en la dimensión en el tiempo se pueden predecir los pixels que se necesitan para realizar el cálculo de los coeficientes wavelet en función de la iteración correspondiente. De esta forma, se disminuye la latencia de acceso a memoria, y por lo tanto el tiempo de ejecución.

#### 4.4.4. Vectorización por columnas

Por último, y basándonos en el estudio del código fuente, analizamos el calculo de la transformada wavelet en el espacio  $(x, y)$  para explotar la localidad temporal y espacial de la memoria cache, basado en el uso de las extensiones SIMD y la utilización de la función wavelet madre *Daub - 4*.

Como ya hemos dicho anteriormente, la transformada de wavelet 3D se aplica sobre las dimensiones tiempo,  $X$  e  $Y$ , en ese orden. En los apartados anteriores, hemos analizado el cálculo de la transformada de wavelet en el tiempo y hemos aplicado la vectorización manual usando los

registros SIMD, el desenrollado del bucle y la pre-búsqueda de datos con el objetivo de disminuir el tiempo de ejecución.

En la dimensión  $X$  la transformada de wavelet se aplica sobre cada una de las filas que componen un *frame* sucesivamente. La secuencia de video se encuentra almacenada en memoria siguiendo un orden por filas, por lo que el cálculo de la transformada wavelet en esta dimensión permite una buena explotación de la localidad espacial de la memoria cache.

En cuanto a la memoria cache, la principal dificultad para el cálculo de la transformada de wavelet 3D surge cuando aplicamos la transformada en la dimensión  $Y$ , ya que necesitamos todos y cada uno de los pixels de cada una de las columnas que componen un *frame*. Esto provoca un gran número de fallos en la memoria caché, incluso en la versión del algoritmo wavelet donde hemos aplicado las técnicas de división en bloques.

En este apartado, presentamos la técnica denominada vectorización por columnas que permite explotar la localidad de la memoria cache y el hecho de que la transformada ya se haya aplicado en la dimensión  $X$ , como una forma eficiente de aplicar la transformada de wavelet en la dimensión  $Y$ .

Dado el primer pixel de un determinado *frame*,  $(x, y, t)$ , para calcular el coeficiente wavelet correspondiente en la dimensión  $Y$  con la función wavelet madre Daub-4, necesitamos los coeficientes wavelet que se encuentran en las cuatro primeras filas de la misma columna y el mismo *frame*, es decir,  $(x, y, t)$ ,  $(x+1, y, t)$ ,  $(x+2, y, t)$  y  $(x+3, y, t)$ , que hemos obtenido como resultado de aplicar la transformada en las dimensiones tiempo y  $X$ , como podemos observar en la figura 4.2.

Teniendo en cuenta que la transformada de wavelet en la dimensión  $X$  se aplica previamente a la dimensión  $Y$ , sólo necesitamos las cuatro primeras filas de un determinado *frame* calculadas en la dimensión  $X$  para poder calcular los coeficientes wavelet en la primera fila de la dimensión  $Y$ . Por lo tanto, para poder calcular una nueva fila en la dimensión  $Y$ , necesitamos calcular dos nuevas filas en la dimensión  $X$ , ya que como podemos observar en la figura 4.2, el incremento en los pixels de los que depende cada coeficiente wavelet se realiza de dos en dos unidades. De esta forma, podemos llevar a cabo el cálculo de la transformada de wavelet en la dimensión  $X$  a la misma vez que se realiza el cálculo en la dimensión  $Y$ , para cada uno de los *frames* que forman una secuencia de vídeo. Por tanto, la vectorización por columnas se pueden definir como otro nivel de *blocking*, en el cual se calculan los coeficientes en la dimensión  $Y$ , en cuanto tenemos los suficientes y necesarios coeficientes calculados en la dimensión  $X$ .

En la figura 4.7 podemos observar un ejemplo de cómo se realiza la vectorización por columnas

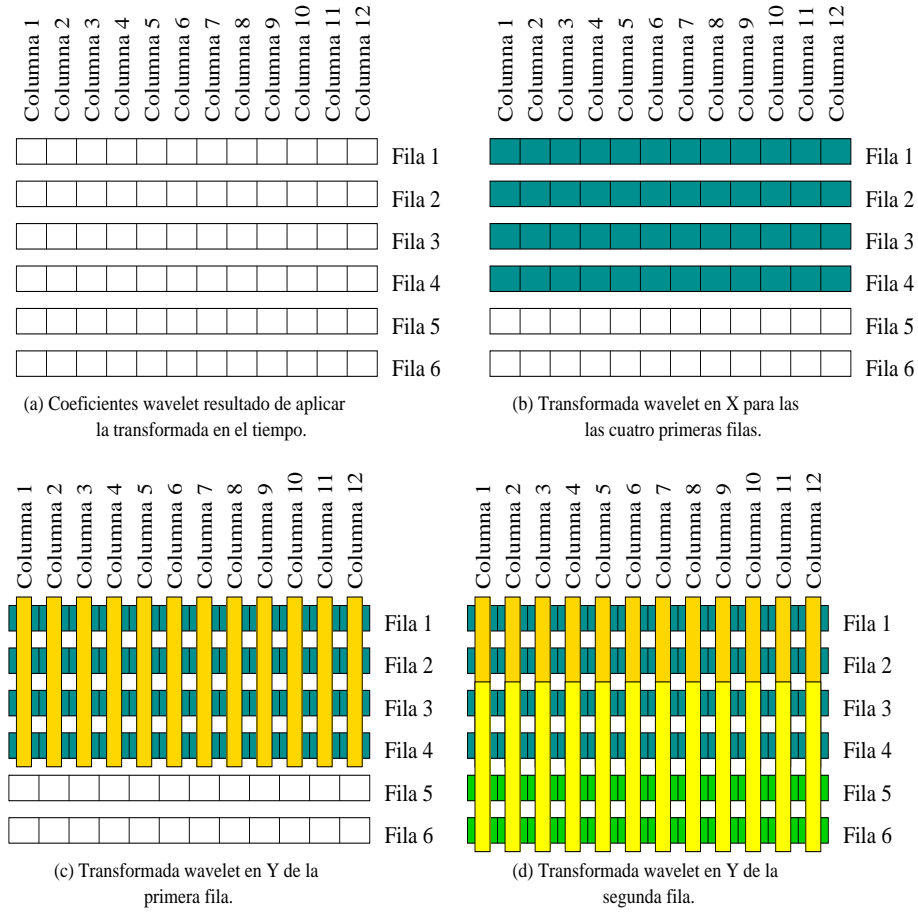


Figura 4.7: Vectorización por columnas para un *frame* de 6 filas y 12 columnas.

para un *frame* de 6 filas y 12 columnas. Inicialmente tenemos los coeficientes wavelet que se obtienen como resultado de aplicar la transformada en el tiempo (figura 4.7 (a)). Aplicamos la transformada de wavelet en la dimensión  $X$  para las cuatro primeras filas (figura 4.7 (b)). Una vez realizado este paso, calculamos los coeficientes wavelet en la dimensión  $Y$  para la primera fila del *frame*, que dependen del resultado de la dimensión  $X$  en las cuatro primeras filas (1, 2, 3 y 4), ver figura 4.7 (c). Además, hacemos uso de los registros SIMD, ya que podemos introducir cuatro coeficientes en un registro SIMD. Para calcular la segunda fila en la dimensión  $Y$ , sólo tenemos que calcular las filas 5 y 6 en la dimensión  $X$ , ya que los coeficientes wavelet en la dimensión  $Y$  para la segunda fila dependen de los coeficientes obtenidos en la dimensión  $X$  que se encuentran en las filas 3, 4, 5 y 6, como podemos observar en la figura 4.7 (d).



TLBs	TLB L1 instr, 4K página, 4-vías, 32 entradas TLB L1 datos, 4K página, 4-vías, 64 entradas
Nivel 1	Cache L1 instr, 16 KB, 4-vías, 32 byte línea Cache L1 datos, 16 KB, 4-vías, 32 byte línea
Nivel 2	Cache Unificada L2, 256 KB, 8-vías, 32 byte línea
Nivel 3	512 Mbytes DRAM

Tabla 4.1: Características de la jerarquía de memoria

## 4.5. Evaluación de resultados

### 4.5.1. Entorno de trabajo

La evaluación de las diferentes propuestas expuestas anteriormente, se han llevado a cabo sobre un procesador Intel Pentium III con una frecuencia de  $1GHz$  y 512 Mbytes de memoria RAM. Las características principales de la jerarquía de memoria se encuentran en la tabla 4.1. Como sistema operativo se ha utilizado Linux 2.2.14. Todos los programas se han escrito usando el lenguaje de programación *C* y se ha utilizado el compilador *Intel C/C++* para Linux (v 5.0.1) [Corporation, 2002b]. En las diferentes propuestas y con el objetivo de mejorar el tiempo de ejecución, hemos aplicado la opción del compilador `-tpp6`, la cual, genera código optimizado para un procesador Pentium III, y aprovecha las ventajas del compilador Intel C/C++.

Para evaluar las diferentes propuestas hemos utilizado los contadores disponibles en la familia de procesadores P6. Para manejar estos contadores hemos utilizado la librería Rabbit (v.2.0.1) [Heller, 2001] que nos permite leer y manipular los contadores y la lista de eventos, usando el lenguaje de programación *C* y bajo el sistema operativo Linux.

Las diferentes propuestas se evalúan sobre la secuencia de vídeo médico *Corazón* de 64 *frames* y  $512 \times 512$  pixels, codificada en escala de grises (8 bits por pixel). Los resultados se pueden generalizar a cualquier secuencia de vídeo del mismo tamaño, ya que el tiempo de ejecución es independiente de la entrada.

### 4.5.2. Evaluación del tiempo de ejecución de las técnicas de división en bloques

En la figura 4.8 se muestra el tiempo de ejecución transcurrido para aplicar la transformada de wavelet dos veces sobre la secuencia *Corazón* para las diferentes técnicas de división en bloques:

- Técnicas de división en cubos utilizando la transformada de wavelet no solapada (*Cubo no solapado*).
- Técnicas de división en cubos utilizando la transformada de wavelet solapada (*Cubo solapado*).
- Técnicas de división en rectángulos utilizando la transformada de wavelet solapada (*Rectangular*).

Los resultados se presentan para diferentes tamaños de bloque, desde 16 *frames* de 16x16 pixels hasta 16 *frames* de 512x512 pixels para las técnicas de división en cubos y desde 16 *frames* de 512x16 pixels hasta 16 *frames* de 512x512 pixels para las técnicas de división en rectángulos. En la figura se ha omitido el número de *frames* ya que siempre se mantiene constante, y sólo se indica el tamaño de la dimensión *Y*, ya que en las técnicas de división en cubos los tamaños de las dimensiones *X* e *Y* coinciden, mientras que en las técnicas de división en rectángulos el tamaño de la dimensión *X* se mantiene constante en 512 pixels para cada una de las configuraciones. Además, en la figura 4.8 se incluye el tiempo de ejecución sin aplicar las técnicas de división en bloques y usando la transformada de wavelet solapada (*Solapada sin técnica de bloques*) y no solapada (*No solapada sin técnica de bloques*). Se representa mediante líneas discontinuas con un tiempo de ejecución constante en cada una de las configuraciones, ya que no se realiza ninguna división en bloques.

En primer lugar, podemos observar que las técnicas de división en bloques reducen drásticamente el tiempo de ejecución del algoritmo original para todas y cada una de las configuraciones. En las técnicas de división en cubos no solapada, la configuración óptima, 16 *frames* de 64x64 pixels, obtiene un *speed-up* de 2,51 sobre la transformada de wavelet original no-solapada, mientras que las propuestas de técnicas de división en bloques solapadas, la división en cubos (configuración óptima 16 *frames* de 128x128 pixels), y la división en rectángulos (configuración óptima 16 *frames* de 512x64 pixels), obtienen un *speed-up* de 1,77 y 2,47 sobre la transformada de wavelet original solapada, respectivamente.

En la figura 4.9, se presenta la contribución de cada una de las propuestas desarrolladas en la secciones 4.3 y 4.4. Como se puede observar las técnicas de división en cubos y rectángulos proporcionan respectivamente un 54 % y un 130 % de media de mejora en el tiempo de ejecución con respecto al cálculo de la transformada de wavelet original solapada. Estos porcentajes reflejan la importancia de la aplicación de las técnicas de división en bloques y establecen al *blocking* como contribución más significativa para reducir el tiempo de ejecución.

Entre las diferentes técnicas de división en bloques, la técnica de división en rectángulos obtiene los mejores resultados, como se esperaba a priori. Esto se debe a que la división en rectángulos explota mejor la localidad espacial de los datos. En las propuestas en las que se

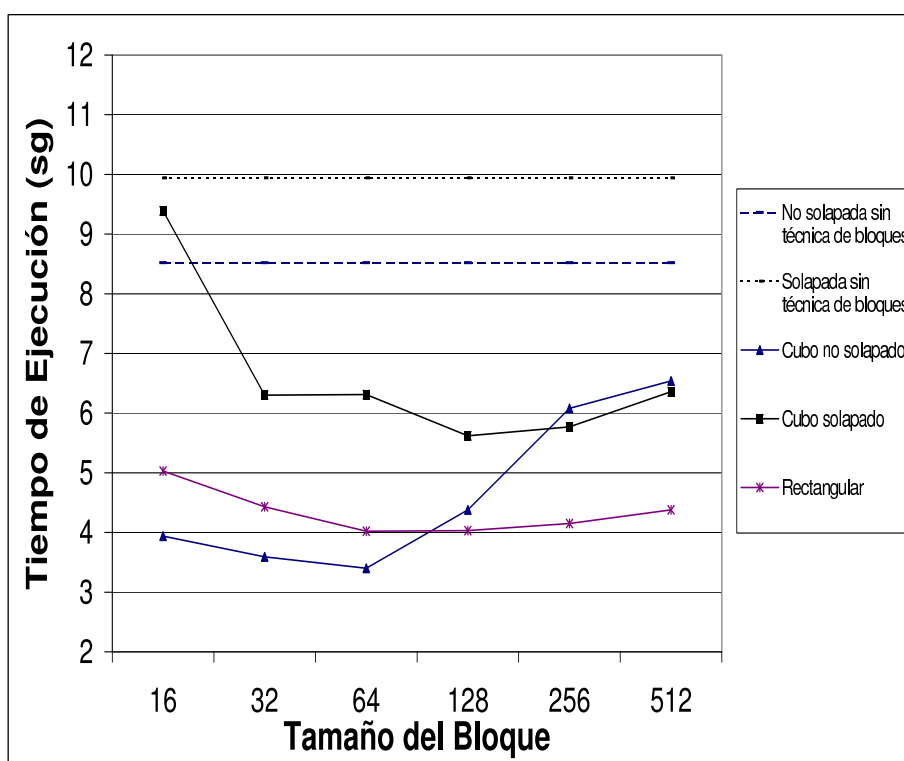


Figura 4.8: Tiempo de ejecución de las técnicas de división en bloques para la secuencia de vídeo médico *Corazón*

utiliza la transformada de wavelet solapada se obtienen mayores tiempos de ejecución que en las que se utiliza la no solapada, salvo si el tamaño de bloque es grande (configuraciones de 16 *frames* de 512x512 y 256x256 pixels en las técnicas de división en cubos), y el porcentaje de datos solapados con respecto al tamaño del bloque es menor. Este aumento en el tiempo de ejecución viene determinado por el incremento del tamaño de los bloques, ya que al aplicar la transformada de wavelet solapada se añaden a cada uno de los bloques, un número determinado de filas, columnas y *frames* en los límites de los bloques, como se indicó en la sección 4.3.1. Sin embargo, las propuestas solapadas obtienen la mejor relación entre rendimiento (tiempo de ejecución y tasa de compresión) y la calidad del vídeo reconstruido. Por ejemplo, en algunas configuraciones, 16 *frames* de 16x16, 32x32 y 64x64 pixels, la técnica de división en cubos usando la transformada de wavelet no solapada obtiene mejores tiempos de ejecución que la propuesta de división en rectángulos. Sin embargo, la división en cubos presenta el inconveniente de la aparición de artefactos en el vídeo reconstruido, lo que implica una reducción de alrededor de 4 puntos en el valor pico de la relación señal/ruido (PSNR), lo que hace descartar esta técnica

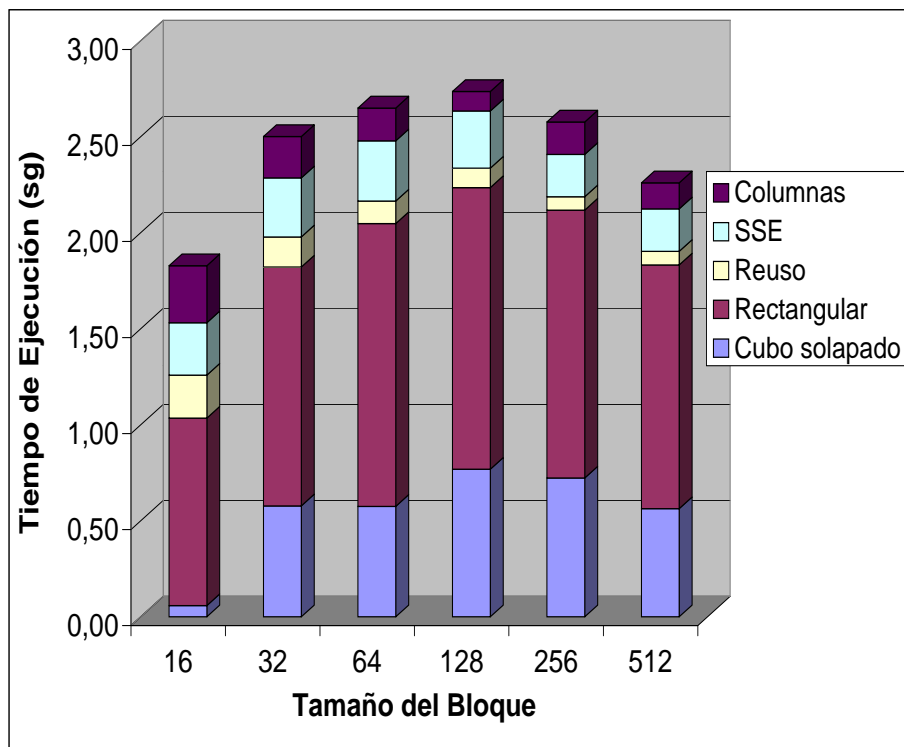


Figura 4.9: Contribución de cada una de las propuestas al tiempo de ejecución

para conseguir una propuesta de alta calidad y compresión para el vídeo médico. Por lo tanto, las técnicas que utilizan la transformada de wavelet solapada mantienen el ratio de compresión y la calidad original, mientras que las técnicas que usan la transformada de wavelet no solapada producen una degradación significativa en el vídeo reconstruido.

Con respecto a la técnicas de división en bloques utilizando la transformada de wavelet solapada, la técnica de división en rectángulos explota mejor el uso de la memoria cache que la división en cubos, lo que implica una reducción significativa del tiempo de ejecución. Por ejemplo, en la técnica de división en rectángulos, la configuración óptima, 16 *frames* de 512x64 pixels, obtiene un *speed-up* de 1,40 con respecto a la configuración equivalente de 16 *frames* de 128x128 pixels en la técnica de división en cubos. Además, el efecto de las técnicas de división en bloques se pone de manifiesto en la mejora del tiempo de ejecución en un 13 % y en un 9 % de las configuraciones óptimas de las técnicas de división en rectángulos y cubos, respectivamente, con respecto a los bloques originales de 16 *frames* de 512x512 pixels.

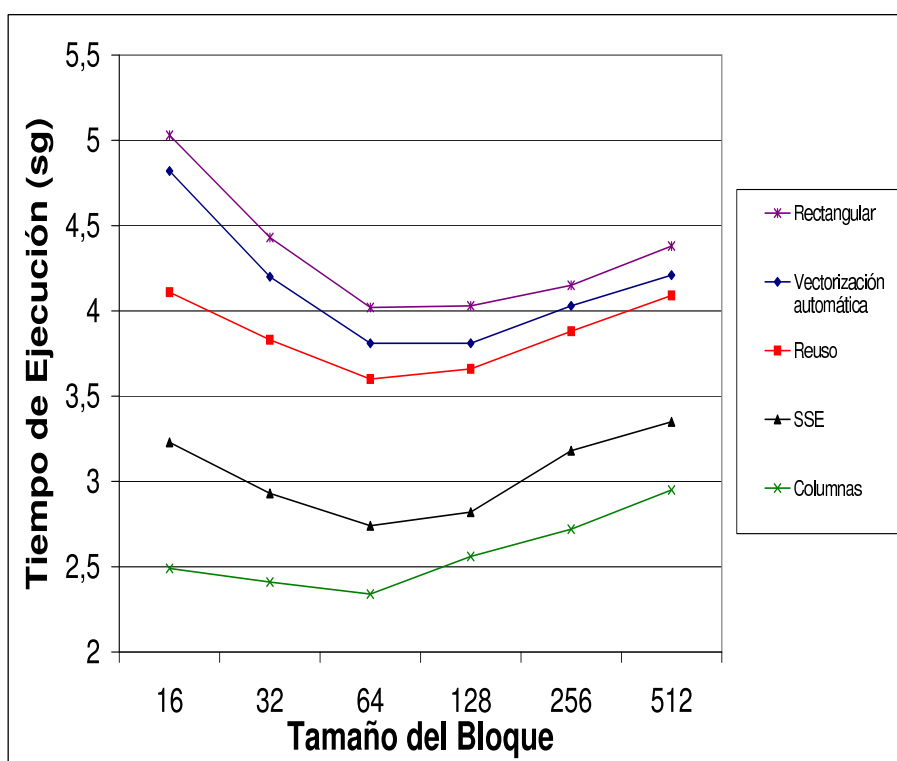


Figura 4.10: Tiempo de ejecución de las diferentes optimizaciones para la secuencia de vídeo médico *Corazón*

### 4.5.3. Análisis del resto de propuestas

En la figura 4.10 se presenta el tiempo de ejecución transcurrido para aplicar la transformada de wavelet dos veces sobre la secuencia *Corazón* para las propuestas descritas en la sección 4.4. Los resultados se presentan para los mismos tamaños de bloque que en la figura 4.8.

Una vez que se ha establecido, según el apartado anterior, que la técnica de división en rectángulos utilizando la transformada de wavelet solapada (*Rectangular*) obtiene los mejores resultados, hemos ido aplicando sucesivamente las diferentes propuestas u optimizaciones expuestas en la sección 4.4 sobre dicha técnica.

La técnica *Reuso* supone la incorporación del reuso de cálculos a la técnica de división en rectángulos utilizando la transformada de wavelet solapada. La *Vectorización automática* presenta los resultados obtenidos por el compilador *Intel C/C++* para Linux (v 5.0.1) [Corporation, 2002b] con las opciones  $-xK$  y  $-axK$  para extraer el paralelismo a nivel de datos a través de instrucciones SIMD. La propuesta denominada *SSE* incluye la vectorización manual usando los

registros SIMD, así como el desenrollado de bucles y la pre-búsqueda de datos sobre la dimensión tiempo. Finalmente, *Columns* incluye la vectorización por columnas y la vectorización manual usando los registros SIMD en el cálculo de los coeficientes wavelet para la dimensión  $Y$ .

En primer lugar, podemos observar en la figura 4.10, que la introducción de cada nueva optimización o propuesta reduce claramente el tiempo de ejecución para todas las configuraciones. La configuración óptima, cuyo tamaño de rectángulo es 16 *frames* de 512x64 pixels se mantiene en todas las propuestas.

La introducción del reuso de los cálculos en punto flotante implica una mejora del tiempo de ejecución entre un 7% y un 22%. Dicho porcentaje aumenta a medida que disminuye el tamaño del rectángulo (ver figura 4.9), ya que el número de operaciones en punto flotante ya realizadas es mayor en los rectángulos de menor tamaño.

La activación de la vectorización automática con las opciones  $-xK$  y  $-axK$  del compilador, para generar código especializado para el uso de las extensiones SIMD no obtiene resultados satisfactorios, ya que los tiempos de ejecución empeoran el comportamiento que se consigue con el reuso de operaciones en punto flotante. Esto se debe a que la vectorización de la transformada de wavelet es complicada y debe ser aplicada con cuidado para poder obtener un beneficio significativo. Debemos recordar que en el cálculo de la transformada de wavelet hay tres bucles anidados y, por ejemplo, la vectorización del bucle más interno no proporciona ningún beneficio. Por lo tanto, la mejor opción para obtener buenos resultados en el cálculo de la transformada de wavelet usando los registros SIMD es la utilización de la vectorización manual, tal y como hemos propuesto anteriormente.

La introducción del uso de los registros SIMD, así como el desenrollado de bucles y la pre-búsqueda de datos sobre la dimensión en el tiempo obtiene un *speed-up* de 1,47, mientras que la vectorización por columnas obtiene un *speed-up* de 1,72, con respecto a la técnica de división en rectángulos (*Rectangular*). Además, es muy importante recordar que la introducción de todas estas propuestas u optimizaciones mantienen tanto la tasa de compresión como la calidad alcanzada en la técnica de división en rectángulos usando la transformada de wavelet solapada, lo que confirma el potencial de todos estos métodos.

Con respecto a *SSE*, la mejora en el rendimiento tiene su origen en tres causas principales. Primero, la utilización de las extensiones SSE para la dimensión en el tiempo. Segundo, el efecto del desenrollamiento del bucle para incrementar el paralelismo a nivel de instrucción y, tercero, el efecto de la pre-búsqueda de datos, que permite que al mismo tiempo que se están calculando cuatro coeficientes wavelet se esté realizando la pre-búsqueda de los pixels necesarios para calcular

los siguientes coeficientes. La unión de estos factores produce una contribución de entre un 22 % y un 31 % (27 % de media) a la mejora del tiempo de ejecución en cada una de las configuraciones con respecto a la técnica de división en rectángulos utilizando el reuso de operaciones en punto flotante, tal y como podemos observar en la figura 4.9.

Finalmente, la propuesta *Columns* reduce, significativamente, el tiempo de ejecución para todas las configuraciones, permitiendo la compresión y transmisión de vídeo en tiempo real (25,7, 26,56, 27,35 y 25,00 imágenes por segundo para las configuraciones de 16 *frames* de  $512 \times 16$ ,  $512 \times 32$ ,  $512 \times 64$  y  $512 \times 128$  pixels respectivamente), excepto las dos configuraciones con mayor tamaño de rectángulo, es decir, 16 *frames* de  $512 \times 512$  y  $512 \times 256$  pixels respectivamente. Esta importante mejora se debe a que se explota mejor la localidad temporal y espacial de la memoria cache gracias a la vectorización por columnas. Además, hemos vectorizado manualmente y usado los registros SIMD en el cálculo de los coeficientes wavelet para la dimensión *Y*. Aunque esto último no obtiene, por si mismo, ningún beneficio sobre el rendimiento final, es necesario aplicarlo para obtener una mejora en el uso de las extensiones SSE al aplicar la vectorización por columnas. La contribución de esta última propuesta ofrece un 18 % con respecto a *SSE*, poniendo de manifiesto la importancia de la vectorización por columnas por si misma.

#### 4.5.4. Análisis del comportamiento de la memoria cache

En esta sección analizamos el comportamiento de la memoria cache cuando se aplica la transformada de wavelet sobre la secuencia de vídeo *Corazón*, a fin de razonar los *speed-ups* obtenidos por las diferentes propuestas, expuestas anteriormente.

Hemos analizado el comportamiento de la memoria cache usando dos eventos, que representan las líneas de entrada en la cache de datos L1 y la líneas de entrada en la cache unificada L2, figuras 4.11 y 4.12 respectivamente. Ambos, representan el número de líneas asignadas a la cache L1 de datos y a la cache unificada L2, respectivamente, es decir, el número de accesos que producen un fallo en cada una de las caches, respectivamente. Se especifica el tamaño del bloque como una relación del tamaño de cada una de las caches, respectivamente. Por ejemplo, el valor 32 significa que el tamaño del bloque es de 16 *frames* de  $64 \times 512$  pixels, lo que da lugar a un conjunto de trabajo de 512 KBytes que es 32 veces el tamaño de la cache L1 de datos y 2 veces la cache unificada L2, respectivamente. En las citadas gráficas no se incluyen las técnicas de división en bloques que utilizan la transformada de wavelet no solapada.

Como podemos observar, en ambas figuras, la técnica de división en rectángulos utilizando la transformada de wavelet solapada (Rectangular) pone de manifiesto que un tamaño de bloque

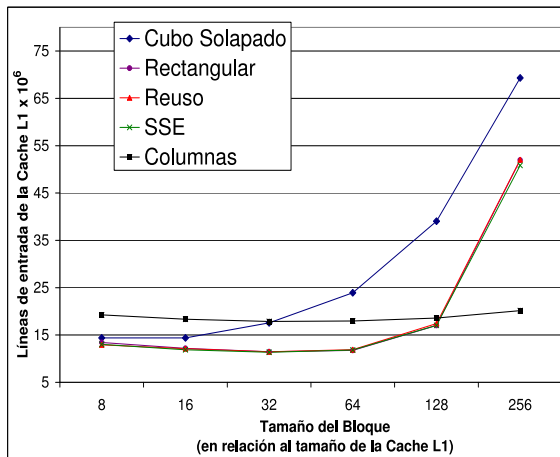


Figura 4.11: Líneas de entrada de la Cache de Datos L1 para la secuencia de vídeo *Corazón*

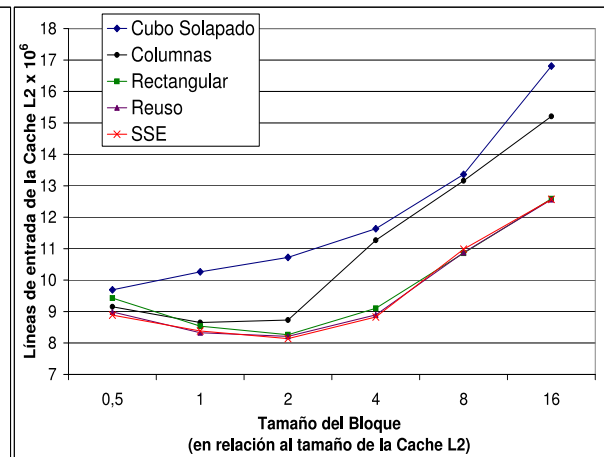


Figura 4.12: Líneas de entrada de la Cache Unificada L2 para la secuencia de vídeo *Corazón*

menor produce un menor número de fallos, tanto en la L1 como en la L2, hasta un tamaño determinado de bloque. El resto de propuestas tienen curvas cuyas tendencias son muy similares o equivalentes, donde las configuraciones óptimas se alcanzan cuando se tiene un tamaño de bloque de 32 y 2 veces el tamaño de la cache L1 y L2, respectivamente. Un tamaño de bloque menor no mejora el número de fallos de cache, ya que, a medida que el tamaño de bloque disminuye, el uso de la transformada de wavelet solapada añade un mayor número de pixels de los siguientes bloques con respecto a la proporción entre el número de pixels que no pertenecen al bloque original y el tamaño del bloque original. Esto implica un mayor trasiego de datos en las caches L1 y L2, respectivamente.

En las figuras 4.13 y 4.14 podemos observar las líneas de entrada en las caches L1 y L2, respectivamente, divididas en las diferentes dimensiones en la que se aplica la transformada de wavelet, *X*, *Y* y *Tiempo*, sobre el vídeo médico *Corazón*, para la primera iteración de dicha transformada. En ambas figuras, se pone de manifiesto el mismo comportamiento en las dimensiones *Y* y *Tiempo*, donde a medida que el tamaño del bloque es más pequeño, el número de fallos en la cache L1 y L2 es mayor, respectivamente, debido a la sobrecarga de la transformada de wavelet solapada con tamaños de bloque pequeños. Por lo tanto, podemos concluir que el tamaño de bloque óptimo depende del tamaño de la memoria cache (como era de esperar) y de la sobrecarga introducida por las operaciones adicionales que se tienen que llevar a cabo.

Volviendo a las figuras 4.11 y 4.12, podemos observar que la técnica de división en rectángulos utilizando la transformada de wavelet solapada produce un número de fallos menor en las caches



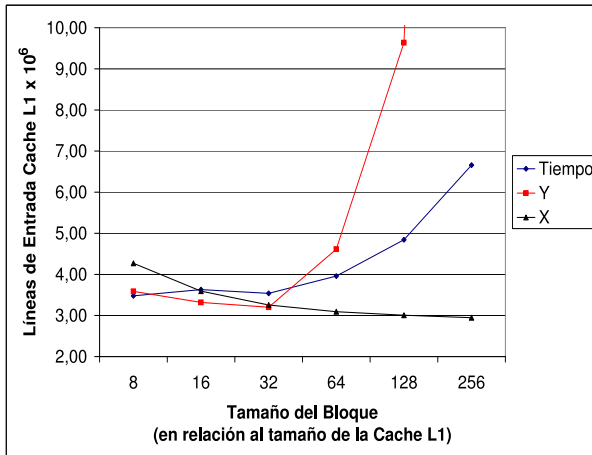


Figura 4.13: Líneas de entrada de la Cache de Datos L1 divididas por dimensiones para la técnica de división en rectángulos utilizando la Transformada de Wavelet Solapada

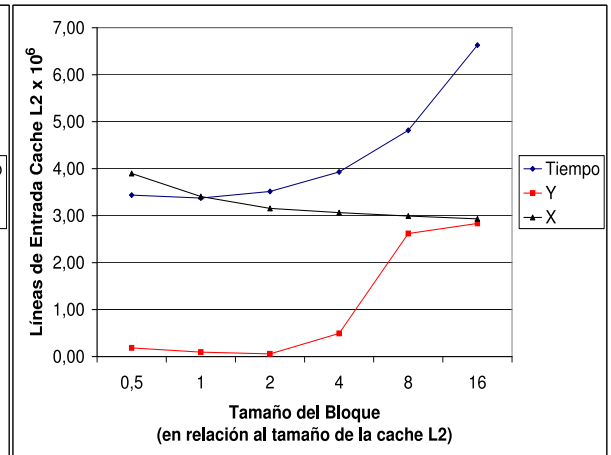


Figura 4.14: Líneas de entrada de la Cache Unificada L2 divididas por dimensiones para la técnica de división en rectángulos utilizando la Transformada de Wavelet Solapada

L1 y L2 que la técnica de división en cubos. Este hecho justifica los tiempos de ejecución menores en la técnica de división en rectángulos. Aunque el tamaño de las filas es mayor debido a la división en rectángulos y no en cubos, los datos se almacenan por filas, por lo que se explota mejor la localidad espacial y se reduce el número de fallos de las caches L1 y L2.

Con respecto al reuso de operaciones en punto flotante, el número de fallos en las caches L1 y L2 es menor que en la técnica de división en rectángulos en la mayoría de configuraciones. Sin embargo, el descenso no es muy significativo, ya que la principal aportación de la reutilización de operaciones consiste en la reducción del número de instrucciones en punto flotante. La figura 4.15 muestra el número de instrucciones en punto flotante ejecutadas por cada configuración. Se puede observar que en la configuraciones donde el tamaño de rectángulo es menor, y por tanto la reutilización de cálculos es mayor, se produce una mayor disminución del número de instrucciones en punto flotante ejecutadas. Estos dos hechos justifican la reducción del tiempo de ejecución.

En cuanto a la vectorización manual utilizando los registros SIMD, *SSE*, se produce el mismo comportamiento descrito anteriormente con respecto al reuso de operaciones en punto flotante, es decir, se reduce el número de fallos en las caches L1 y L2 en la mayoría de las configuraciones, aunque estas diferencias no son muy significativas, ya que el principal beneficio que se obtiene del uso de las extensiones SSE en la vectorización manual reside en una notable disminución del número de instrucciones en punto flotante debido al uso de extensiones multimedia, tal y como

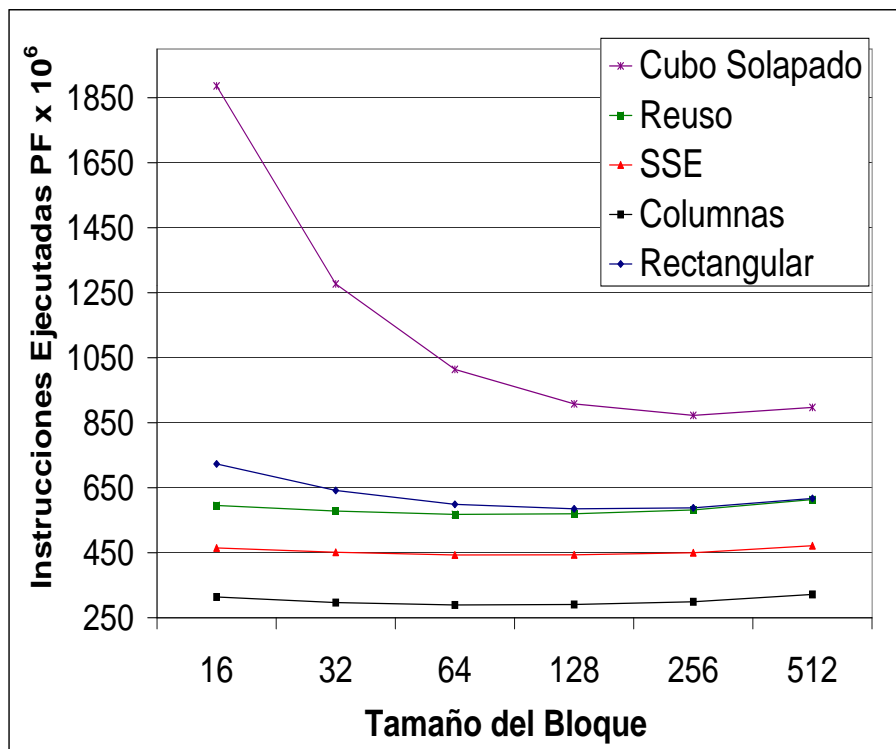


Figura 4.15: Instrucciones en punto flotante ejecutadas para la secuencia de vídeo *Corazón*

podemos observar en la figura 4.15. Hay que dejar claro que el uso de las extensiones SSE no reduce el número total de operaciones en punto flotante, las cuales sólo dependen del algoritmo en particular. Sin embargo, el número de instrucciones en punto flotante ejecutadas disminuye, ya que estas operaciones se realizan en paralelo (de cuatro en cuatro), a través de los registros SIMD. Por lo tanto, estamos explotando el paralelismo a nivel de datos. Además, en *SSE* no podemos medir el beneficio obtenido únicamente a través del número de fallos en las caches L1 o L2, ya que las instrucciones de pre-búsqueda siguen provocando los correspondientes fallos de cache. Sin embargo, como se realizan las pre-búsquedas, los datos estarán disponibles cuando sea necesario, evitando la ejecución de instrucciones con dependencias de datos que pueden parar el flujo de ejecución del procesador.

Finalmente, en *Columnas* se produce un aumento significativo del número de fallos en las caches L1 y L2 con respecto a las propuestas anteriores, aunque la vectorización por columnas explota mejor la localidad espacial y temporal en el cálculo de la transformada wavelet en las dimensiones *X* e *Y*. Este aumento en el número de fallos de cache se debe a una cuestión de implementación. Cuando aplicamos la vectorización por columnas se generan dos bandas para la

dimensión  $Y$ , una banda pasa-baja y una banda pasa-alta. Los coeficientes pasa-alta deben ser guardados en posiciones diferentes a las que se encuentran los pixels originales para no eliminar dichos pixels, los cuales, se necesitan para el resto de cálculos en la dimensión  $X$ . Por lo tanto, necesitamos utilizar una mayor cantidad de memoria para calcular la transformada que en las propuestas anteriores. Además, se produce un mayor movimiento de datos, lo que provoca una peor localidad en el uso de las caches. Este hecho se refleja en el aumento del número de fallos en las caches L1 y L2.

Sin embargo y a pesar de este problema de implementación, la vectorización por columnas obtiene tiempos de ejecución significativamente menores que el resto de propuestas en cada una de las configuraciones. Esta mejora en el tiempo de ejecución se debe a dos razones fundamentales:

- La introducción de las diferentes propuestas analizadas en este capítulo (técnicas de división en bloques, pre-búsqueda, etc), incluida la vectorización por columnas provocan que el cálculo de la transformada de wavelet 3D no se encuentre tan limitada por la cantidad de memoria disponible como en el algoritmo original, incluso con el problema de implementación comentado anteriormente.
- Dado que el algoritmo no se encuentra tan limitado por la memoria disponible, y siguiendo la ley de Amdahl, cualquier propuesta que reduzca la parte computacional del algoritmo tiene un gran impacto sobre el rendimiento final. Con la introducción de la vectorización por columnas se obtiene una reducción espectacular del número de instrucciones en punto flotante ejecutadas. Esto se debe, de nuevo, a una explotación del paralelismo a nivel de datos mediante el uso de la vectorización manual en el cálculo de la transformada de wavelet en la dimensión  $Y$ . Por ejemplo, en la vectorización por columnas se reduce el número de instrucciones ejecutadas en punto flotante en un 72 % y un 48 % con respecto a las técnicas de división en cubos y rectángulos, respectivamente, y en un 34 % con respecto a *SSE*, tal y como podemos observar en la figura 4.15.

En la figura 4.15 también podemos ver que el comportamiento del número de instrucciones en punto flotante ejecutadas para la técnica de división en cubos usando la transformada de wavelet solapada, es totalmente diferente al resto de propuestas. Esto se debe, precisamente, a la división de la secuencia original en cubos, lo que implica un aumento del número de instrucciones en punto flotante ejecutadas cuando el tamaño del bloque es menor, ya que al aplicar la transformada de wavelet solapada se necesitan filas, columnas y *frames* de los siguientes bloques. De hecho, la proporción entre el número de pixels que no pertenecen al cubo original y el tamaño del cubo original aumenta considerablemente. Por todo ello, el reuso de operaciones en punto flotante,

introducido a partir de la técnica de división en rectángulos, toma una importancia vital, ya que además de reducir el número de instrucciones en punto flotante ejecutadas (hasta un 21 % en las distintas configuraciones), también es determinante en el mantenimiento de un número similar de operaciones en punto flotante en todas y cada una de las configuraciones.

## 4.6. Conclusiones

En este capítulo hemos implementado la compresión y transmisión en tiempo real de la transformada de wavelet 3D cuando se comprime vídeo médico, mediante el desarrollo de las siguientes propuestas:

- Hemos desarrollado y evaluado varias técnicas de división en bloques para explotar el uso de la jerarquía de memoria.
- Hemos propuesto el reuso de operaciones en punto flotante para disminuir el número de instrucciones ejecutadas y la cantidad de accesos a memoria.
- Hemos desarrollado y analizado la vectorización manual y automática que explota el paralelismo a nivel de datos mediante la realización en paralelo de varias operaciones en punto flotante a través de instrucciones SIMD. Se ha demostrado que la vectorización automática a través del compilador propietario de Intel no obtiene mejores resultados que la vectorización manual, que hemos propuesto aplicar para el cálculo de la transformada wavelet en las dimensiones *Tiempo* e *Y*.
- Se ha propuesto el desenrollado manual del bucle de la dimensión en el tiempo para el cálculo de la transformada wavelet, y se han introducido instrucciones de pre-búsqueda para reducir el número de fallos de cache y explotar el paralelismo a nivel de instrucción.
- Finalmente, hemos propuesto el desarrollo de la vectorización por columnas en la dimensión *Y* para disminuir el número de instrucciones en punto flotante ejecutadas y llevar a cabo un uso más óptimo de las extensiones multimedia.

La evaluación de resultados demuestra que la introducción de todas las propuestas presentadas en este capítulo permiten la compresión y transmisión de la transformada de wavelet en tiempo real. La configuración óptima de 16 *frames* de 512x64 pixels consigue unos *speed-ups* de 4,25 sobre el cálculo de la transformada de wavelet sin aplicar las técnicas de división en bloques, 2,40 sobre la configuración óptima (16 *frames* de 128x128 pixels) de la técnica de división en cubos usando

la transformada de wavelet solapada, 1,72 sobre la técnica de división en rectángulos usando la transformada de wavelet solapada, 1,54 con relación al reuso de cálculos en punto flotante y 1,17 con respecto a la vectorización manual usando los registros SIMD. Además, todas las propuestas desarrolladas en este capítulo mantienen la calidad y la tasa de compresión del codificador wavelet original, desarrollado en el capítulo 3.

Por último y con una importancia máxima, las diferentes propuestas desarrolladas en este capítulo son bastante sencillas y fáciles de implementar, por lo que se puede generalizar su aplicación a aplicaciones multimedia con características similares como los estándares de codificación de vídeo MPEG-2 o MPEG-4. De hecho, la integración de las diferentes técnicas suponen un primer paso para la automatización del proceso de convergencia entre las aplicaciones multimedia y los procesadores de propósito general que disponen de extensiones multimedia. Dicha integración de las propuestas realizadas podría formar parte de una librería gráfica que tuviera como objetivo la optimización y la mejora de las aplicaciones multimedia.



## Capítulo 5

# La transformada de wavelet 3D en un entorno multi-hilo

---

### 5.1. Introducción

Las aproximaciones tradicionales para la mejora del diseño de procesadores se han centrado principalmente en conseguir un aumento de la frecuencia de reloj del procesador, un aumento del paralelismo a nivel de instrucción (ILP) y la mejora del comportamiento de las memorias caches.

Las técnicas para conseguir una mayor velocidad de reloj se basan en la división del procesador en un número elevado de etapas, dando lugar a lo que se conoce como pipelines super-segmentados. El aumento de la frecuencia de reloj debido al incremento del número de etapas del pipeline pueden mejorar el rendimiento de una manera considerable. Sin embargo, la existencia de un número mayor de etapas en el pipeline supone un aumento de la complejidad, el consumo y el coste de recuperación ante determinados eventos (fallo de cache, fallo de predicción de un salto, interrupciones, etc).

A fin de explotar el paralelismo a nivel de instrucción se propusieron los procesadores superescalares [Smith y Sohi, 1995] y los VLIW [Semiconductors, 1999]. Los procesadores superescalares cuentan con diversas unidades funcionales capaces de ejecutar multiples instrucciones en paralelo. Aún así, no siempre es posible encontrar instrucciones libres de dependencias que mantengan las unidades funcionales ocupadas. El desafío consiste en encontrar el número suficiente de instrucciones para mantener ocupadas el máximo tiempo posible a las distintas unidades de ejecución. Para conseguir este objetivo se propone la ejecución en desorden, donde en la ventana de instrucciones se evalúan de forma simultánea un número elevado de instrucciones que se envían a las unidades de ejecución una vez que se encuentran libres de dependencias, sin tener en cuenta el

orden secuencial de las instrucciones en los programas.

Por otro lado, la latencia de acceso a la memoria principal DRAM es elevada comparada con la velocidad de ejecución del procesador. Una técnica para reducir dicha latencia es añadir al procesador memorias caches, que pueden ofrecer un rápido acceso a los datos e instrucciones más frecuentemente utilizadas. Sin embargo, las memorias caches sólo pueden ser rápidas si son pequeñas. Por esta razón, los procesadores se diseñan con una jerarquía de memoria multi-nivel de mayor a menor velocidad y de menor a mayor tamaño según la cercanía con el procesador, es decir, las memorias caches más rápidas y pequeñas se sitúan más cerca del procesador y las memorias caches con una mayor capacidad y menor velocidad se colocan más lejos del procesador. Sin embargo, siempre se producirán situaciones en las que los datos o instrucciones que se necesiten en un instante determinado no se encuentren en las memorias caches más cercanas al procesador (fallo de datos o de instrucciones), por lo que habrá que seguir accediendo a memoria principal.

La gran mayoría de las técnicas para mejorar el rendimiento de los procesadores de una generación con respecto a la anterior son complejas y suelen implicar el aumento del número de transistores y del consumo. Estas técnicas aumentan el rendimiento, aunque la eficiencia no alcanza el 100 %. Es decir, la duplicidad del número de unidades de ejecución de un procesador no consigue doblar el rendimiento debido a la limitación del paralelismo en los flujos de instrucción y a la latencia de memoria. De la misma forma, doblar la frecuencia de reloj a coste de un aumento del número de etapas del pipeline no consigue doblar el rendimiento debido al número de ciclos perdidos por los fallos que se producen en la predicción de los saltos.

Un estudio del software actual revela que las aplicaciones de servidor consisten en múltiples hilos o procesos que se pueden ejecutar en paralelo. El procesamiento de una transacción *on-line* y los servicios web contienen muchos hilos de software que se pueden ejecutar simultáneamente para conseguir un mayor rendimiento. Por otro lado, el nivel de paralelismo también aumenta en otro tipo de aplicaciones como las orientadas hacia los computadores personales. Se trata de explotar el denominado paralelismo a nivel de hilo (*Thread-Level Parallelism*), en adelante TLP [Marr *et al.*, 2002], para conseguir mejorar el rendimiento, frente al aumento del número de transistores y del consumo de energía.

Tradicionalmente, la mejora del TLP en un sistema se ha conseguido mediante el uso de multiprocesadores. Se trata de disponer de un número mayor de procesadores, de tal forma que las aplicaciones consiguen mejorar el rendimiento mediante la ejecución de múltiples hilos en los distintos procesadores al mismo tiempo. Estos hilos pueden pertenecer a una única aplicación, a diferentes aplicaciones ejecutándose simultáneamente, a servicios del sistema operativo, o a hilos del sistema operativo que realizan tareas de mantenimiento en segundo plano. Los sistemas



multiprocesadores se han utilizado durante muchos años, y los programadores están familiarizados con las técnicas para conseguir altos niveles de rendimiento en ellos.

En los últimos años, se han propuesto otras técnicas para fomentar la explotación del TLP. Una de estas técnicas es el multiprocesador en un chip (*Chip MultiProcessor* CMP) [Hammond *et al.*, 1997][Hammond *et al.*, 2000], donde dos o más procesadores se implementan en un único chip. Cada uno de los dos procesadores tienen un juego completo de recursos de ejecución y un estado de la arquitectura. Los procesadores pueden o no compartir el chip de memoria cache, que por lo general es grande. Se pueden tener múltiples procesadores CMP en una configuración multiprocesador.

Otra aproximación, consiste en permitir que un solo procesador ejecute múltiples hilos mediante la conmutación entre ellos. La técnica conocida como *time-slice multithreading* [Marr *et al.*, 2002], permite al procesador conmutar entre los diferentes hilos tras un periodo de tiempo determinado. Dicha técnica puede incrementar el tiempo de ejecución de cada uno de los hilos independientemente, pero minimiza el efecto de las latencias grandes de acceso a memoria. Otra técnica, denominada *switch-on-event multithreading* [Marr *et al.*, 2002], conmuta entre los hilos en aquellos instantes en los que se producen eventos con una latencia elevada, como por ejemplo un fallo de cache. Esta aproximación puede ser eficiente para aplicaciones de servidor, que tienen un gran número de fallos de cache y los hilos están ejecutando tareas parecidas. Sin embargo, ambas técnicas multi-hilo no consiguen un solapamiento óptimo de la ejecución debido al uso ineficiente de los recursos por causas tales como fallos en la predicción de saltos, dependencias entre instrucciones, etc.

La aparición del concepto de ejecución simultánea de varios hilos (*Simultaneous MultiThreading* SMT) [Tullsen *et al.*, 1995][Eggers *et al.*, 1997][Marcuello y González, 1999] propone una técnica hardware que aumenta la productividad (*throughput*) de un procesador mediante la emisión simultánea de instrucciones que pertenecen a hilos diferentes, utilizando los múltiples recursos de un procesador superescalar [Tullsen *et al.*, 1995]. De esta forma se aumenta la productividad del procesador, ya que ejecutando múltiples hilos se utilizan recursos que no serían usados por un sólo hilo debido a la falta de paralelismo a nivel de instrucción causado por eventos como fallos de cache o de predicción de saltos. Por lo tanto, SMT puede obtener beneficios significativos en diferentes aplicaciones como bases de datos comerciales, servidores web y aplicaciones científicas, a través del uso eficiente de los recursos del procesador.

La compañía Intel utiliza el concepto de ejecución simultánea de varios hilos para proponer la tecnología *Hyper-Threading* (HT) en los microprocesadores Pentium IV Xeon [Marr *et al.*, 2002]. La tecnología HT permite que un único procesador implementado físicamente aparezca como dos

procesadores lógicos desde el punto de vista del sistema operativo, ya que se duplica el estado de la arquitectura en cada uno de los procesadores lógicos. Dichos procesadores lógicos pueden ejecutar diferentes tareas o hilos de forma simultánea, mediante el uso compartido de los recursos hardware del único procesador físico. Por lo tanto, un procesador HT no es equivalente a dos procesadores reales aunque puede mejorar el rendimiento de las aplicaciones de forma sustancial.

En los dos capítulos anteriores hemos desarrollado un codificador basado en la transformada de wavelet 3D para la compresión de vídeo médico que permite la compresión y la transmisión de vídeo en tiempo real en la fase de cálculo de la transformada de wavelet 3D. Dicho codificador obtiene una gran tasa de compresión y una calidad excelente, ya que no se aprecian diferencias entre la secuencia de vídeo original y la reconstruida.

Hasta ahora, aunque hemos mejorado considerablemente el tiempo de ejecución, el algoritmo sigue estando limitado hasta cierto punto por los accesos a memoria, provocando un uso ineficiente de los recursos del procesador, los cuales se mantienen inactivos durante muchos ciclos. En este capítulo pretendemos aprovechar la tecnología SMT para ocupar dichos recursos no utilizados en tareas relacionadas con el procesamiento del codificador basado en la transformada de wavelet 3D. Para ello, estudiaremos y analizaremos las principales características, ventajas e inconvenientes de un procesador con tecnología *Hyper-Threading*. A partir del conocimiento de la tecnología HT, desarrollaremos varias propuestas basadas en un esquema multi-hilo para aprovechar las ventajas de un procesador HT. Las propuestas se diferencian en la forma de descomponer la aplicación original, por lo que podemos tener una descomposición basada en el dominio de los datos o una descomposición basada en una división funcional. En la primera propuesta, cada hilo realiza de forma simultánea las mismas tareas sobre una parte independiente de los datos. En la segunda técnica, cada hilo procesa una tarea diferente sobre el mismo conjunto de datos simultáneamente. Basándonos en una división funcional, desarrollaremos tres técnicas que se diferencian en cómo se realiza la división de cada una de las fases del codificador entre los distintos hilos de ejecución.

En la parte de evaluación de resultados, analizaremos el comportamiento de las diferentes propuestas sobre un procesador HT y compararemos los resultados con respecto a un procesador sin tecnología *Hyper-Threading*. Determinaremos las ventajas e inconvenientes de cada una de las propuestas con la tecnología *Hyper-Threading*.

## 5.2. La tecnología *Hyper-Threading*

La tecnología *Hyper-Threading* incorpora el concepto de SMT a la arquitectura de los procesadores Intel. De esta forma, se permite que un único procesador físico pueda aparecer como dos

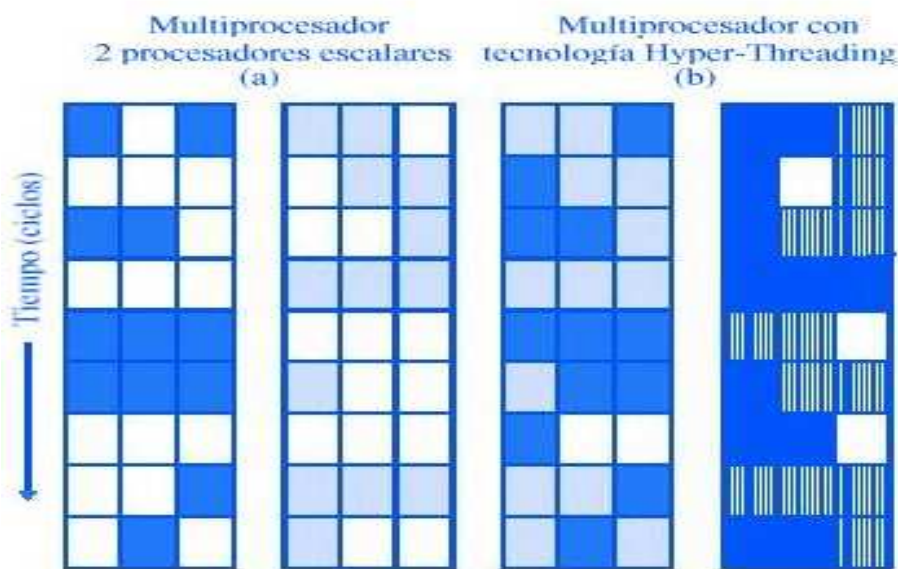


Figura 5.1: (a) Multiprocesador con dos procesadores sin tecnología *Hyper-Threading*. (b) Multiprocesador con dos procesadores con tecnología *Hyper-Threading*.

procesadores lógicos que pueden ejecutar simultáneamente dos hilos de una misma aplicación. Los recursos del procesador como la memoria cache, las unidades de ejecución, los predictores de saltos, la lógica de control y los buses se comparten, mientras que el estado de la arquitectura se duplica para cada uno de los procesadores lógicos. El estado de la arquitectura consiste en una serie de registros que incluye los registros de propósito general, los registros de control, los registros que controlan la programación avanzada de interrupciones (APIC) y algunos registros sobre el estado del procesador. La compartición de los recursos del procesador puede aumentar el tiempo de ejecución de una aplicación basada en un sólo hilo. Sin embargo, en una aplicación basada en un esquema multi-hilo, el uso compartido de los recursos del procesador mejora el rendimiento global. En la figura 5.1, podemos observar como un procesador HT puede mejorar el uso de los recursos del procesador. En la parte izquierda de la figura 5.1 aparece un multiprocesador que contiene dos procesadores superescalares, los cuales se encuentran ejecutando hilos diferentes y pueden ejecutar hasta tres instrucciones cada ciclo de reloj. Una casilla dentro de un procesador representa una unidad de ejecución del mismo. Un conjunto de tres casillas horizontales determina el uso de cada una de las unidades de ejecución en un ciclo de reloj determinado, representándose la inactividad de una unidad de ejecución mediante el color blanco. Los conjuntos de casillas en sentido vertical indican la utilización de las unidades de ejecución a través del tiempo. En la parte derecha de la figura 5.1 podemos ver el mismo multiprocesador que en la parte izquierda

con dos procesadores con tecnología *Hyper-Threading*, cada uno de los cuales está ejecutando dos hilos de forma simultánea. Como se puede observar, en el multiprocesador HT se puede producir una mejora significativa en el uso de los recursos de cada uno de los procesadores y se reduce el tiempo de inactividad con respecto al multiprocesador sin tecnología HT.

Desde el punto de vista de la arquitectura, la aparición de la tecnología *Hyper-Threading* significa que el sistema operativo y los programas de usuario pueden planificar procesos o hilos sobre los procesadores lógicos, como si se tratase de múltiples procesadores físicos. En un procesador HT se pueden ejecutar dos aplicaciones o hilos de forma paralela o simultánea, de la misma forma que en un multiprocesador con dos procesadores. Por lo tanto, desde el punto de vista de la micro-arquitectura, la tecnología HT significa la ejecución simultánea de varias instrucciones, las cuales proceden de distintos procesadores lógicos, sobre los recursos compartidos de un único procesador.

Desde el punto de vista de la programación, la incorporación de la tecnología *Hyper-Threading* a la arquitectura de los procesadores Intel representa un desafío para el desarrollo de aplicaciones paralelas que comparten recursos como unidades de ejecución o memorias cache. El nivel de paralelización de las aplicaciones va ser de un grano más fino que el realizado hasta este momento, ya que en cada instante se deben determinar aquellos recursos compartidos que está utilizando un hilo y que, por lo tanto, otro hilo no puede utilizar y viceversa, es decir, aquellos recursos que un hilo no está utilizando y que otro hilo puede usar libremente.

## 5.3. Trabajo relacionado

### 5.3.1. Paralelización de la transformada de wavelet

A lo largo del tiempo, se han desarrollado diferentes esquemas para paralelizar la transformada de wavelet sobre distintas arquitecturas que han sentado las bases para desarrollar esquemas de programación multi-hilo.

Con respecto a una paralelización de grano grueso, Yang y Misra [1998] desarrollan un algoritmo para la paralelización de la transformada de wavelet de  $n$  dimensiones sobre máquinas de memoria compartida y distribuida. Para las máquinas de memoria distribuida, se proponen varios métodos para dividir los datos entre los distintos procesadores y calcular la transformada de wavelet:

- División tradicional. Los datos se dividen entre los diferentes procesadores según la primera

dimensión. Una vez calculada la primera dimensión de la transformada de wavelet en cada uno de los procesadores, los datos se vuelven a dividir entre los diferentes procesadores, según la segunda dimensión, y así, sucesivamente. Finalmente, los datos se agrupan para obtener el resultado final.

- División en bloques: Los datos se dividen en diferentes bloques, teniendo en cuenta cada una de las dimensiones, para obtener varios bloques del mismo tamaño que se reparten entre los procesadores. En cada uno de los procesadores, se calcula la transformada de wavelet del bloque. Para realizar este cálculo, cada procesador necesita algunos datos que se encuentran en otros procesadores para poder calcular los coeficientes de los extremos del bloque. La cantidad de datos necesaria depende del número de coeficientes de la función wavelet madre, de forma que, cuanto mayor sea el número de coeficientes, mayor será el número de comunicaciones entre los distintos procesadores.
- División en bloques sin comunicaciones. Se trata de realizar una división en bloques entre los distintos procesadores y realizar el cálculo de la transformada de forma independiente, por lo que no hay comunicaciones entre los diferentes procesadores.
- División en capas sin comunicaciones. Se lleva a cabo una división en bloques entre los distintos procesadores, teniendo en cuenta sólo la última dimensión para conseguir una distribución más eficiente que la anterior. La transformada se calcula independientemente sobre cada uno de los bloques para evitar las comunicaciones entre los distintos procesadores.

Los dos últimos esquemas se implementan sobre una red Ethernet de 10Mbits/s de estaciones de trabajo SGI, utilizando MPI como librería de paso de mensajes. El último método consigue un escalado casi lineal hasta un número de seis procesadores. Sin embargo, ambos métodos modifican la semántica original de la transformada de wavelet, al no realizar las comunicaciones de los datos necesarios entre los procesadores para calcular la transformada de wavelet en los extremos de los bloques o capas de forma correcta. Por lo tanto, se puede producir la aparición de artefactos en la reconstrucción de las imágenes o videos, lo cual es muy negativo en las aplicaciones objetivo de esta tesis, es decir, en los videos médicos.

En cuanto a las máquinas de memoria compartida, se implementan dos propuestas denominadas paralelismo homogéneo y heterogéneo. En la primera, se utiliza la división tradicional para dividir los datos entre los diferentes procesadores, mientras que en la segunda, se utiliza la división en bloques sin comunicaciones. Los resultados sobre una SGI Power Challenge con directivas de multi-procesamiento sobre el lenguaje C para implementar las comunicaciones demuestran

que a medida que el número de procesadores aumenta, el paralelismo homogéneo tiene un mejor comportamiento que el heterogéneo.

Por otro lado y con el objetivo de proponer un método que no altere la semántica de la transformada de wavelet, Moller y Hegland [2000] implementan el cálculo de la transformada de wavelet 2D sobre el multicomputador IBM SP2 y el multiprocesador vectorial Fujitsu VPP300. Para ello, distribuye un número determinado de columnas consecutivas de la imagen entre los diferentes procesadores para aplicar, localmente, la transformada sobre cada una de las filas ( $x$ ) de todos los procesadores. Para realizar este paso se necesita que cada procesador envíe al que se encuentra a su derecha  $D - 2$  pixels, donde  $D$  es el número de coeficientes de la función wavelet madre. Por ejemplo, con la función wavelet madre Daub-4 cada procesador tendría que enviar dos pixels. Una vez realizada la transformada por filas, se transpone cada una de las matrices almacenadas en los distintos procesadores, para aplicar la transformada por columnas ( $y$ ) (de nuevo, la transformada se aplica, localmente, por filas en cada uno de los procesadores), sin la necesidad de realizar ninguna comunicación entre los procesadores. Ambas implementaciones del método propuesto, sobre el IBM SP2 y el Fujitsu VPP300, consiguen un buen escalado con respecto al número de procesadores.

### 5.3.2. La tecnología *Hyper-Threading*

Con respecto a la tecnología *Hyper-Threading*, recientemente se han desarrollado varios trabajos que han demostrado la mejora del rendimiento de las aplicaciones sobre un procesador con tecnología *Hyper-Threading*.

Wang *et al.* [2002] describen la técnica de pre-cálculo especulativo para mejorar la latencia de las aplicaciones de un sólo hilo mediante el uso de recursos hardware inactivos en un procesador HT. Se trata de efectuar operaciones de pre-búsqueda de datos al mismo tiempo que se realizan el resto de operaciones de una aplicación determinada. La técnica puede mejorar el rendimiento de aplicaciones que realizan una gran cantidad de accesos a memoria, el patrón de acceso a memoria es difícil de predecir y el conjunto de datos es demasiado grande causando muchos fallos de datos. Para ello, se determinan las lecturas de memoria más críticas (*delinquent loads*), que se definen como aquellas lecturas que tienen un mayor impacto sobre el rendimiento de la aplicación. Se construye un ejemplo con dos hilos: el principal, que realiza las operaciones, y el hilo que lleva a cabo las pre-búsquedas de datos para evitar los fallos de cache y reducir la latencia de acceso a memoria. Un procesador HT obtiene un *speed-up* entre un 22 % y un 45 % con respecto a un procesador sin tecnología HT.

A la misma vez, Chen *et al.* [2002] analizan el comportamiento de aplicaciones multimedia representativas como el estándar de compresión de vídeo MPEG-2 y la detección del *watermarking* de vídeo, en arquitecturas superescalares y en procesadores con tecnología *Hyper-Threading*. Se realiza un estudio del tiempo de CPU que utiliza la decodificación de vídeo MPEG-2 en cada uno de sus principales procesos, en el que se detectan dos tendencias en su comportamiento: procesos que se encuentran limitados por la capacidad de computo como la decodificación de longitud variable y la transformada inversa del coseno, y procesos limitados por la memoria, como la compensación de movimiento. Los primeros emiten la mayor parte de sus instrucciones (90 %) a través de la unidad de ejecución de punto flotante (FP/SIMD), lo que hace que la unidad de ejecución de enteros se encuentre prácticamente sin utilizar, mientras que los segundos mantienen ocupado el bus durante una parte significativa del tiempo (30 %). Este análisis se puede generalizar a la correspondiente codificación de vídeo MPEG-2. A partir de este estudio, se proponen dos métodos basados en una descomposición basada en el dominio de los datos, en la que la decodificación de vídeo MPEG-2 se realiza mediante un esquema multi-hilo, en el que se asigna a cada hilo, de forma estática, una de las imágenes a decodificar, o las imágenes se dividen en trozos que se asignan, de forma dinámica, a cada uno de los hilos. Además, se propone una descomposición basada en una división funcional para la detección del *watermarking* de vídeo en la que se identifican dos procesos principales: la decodificación de vídeo y la detección del *watermark*. A partir de esta división, se proponen dos métodos que asignan un hilo para cada una de las tareas, o un hilo para la decodificación de vídeo y dos hilos para la detección de *watermark*, con el objetivo de conseguir un mejor balanceo de la carga entre los diferentes hilos. Un procesador HT obtiene un *speed-up* de un 4 % y un 7 % para los métodos de asignación estática y dinámica, respectivamente, basados en una descomposición basada en el dominio de los datos con respecto a un procesador sin la tecnología HT, mientras que los dos métodos de descomposición funcional basados en dos y tres hilos obtienen un *speed-up* de un 8 % y un 18 %, respectivamente. Este último resultado determina que un uso eficiente de la tecnología *Hyper-Threading* requiere un balanceo correcto de la carga entre los diferentes hilos, para obtener un rendimiento significativo.

Finalmente, Magro *et al.* [2002] realizan un estudio para determinar el rendimiento de varias aplicaciones numéricas de calculo intensivo sobre un procesador HT. Por ejemplo, se analizan aplicaciones de tipo genético, análisis y diseño mecánico, predicción meteorológica, etc. Los resultados obtienen un *speed-up* de entre un 5 % y un 28 % con respecto a un procesador sin tecnología de varios hilos simultáneos. Concluye estableciendo que dicha tecnología puede aumentar la eficiencia del solapamiento de la ejecución de flujos de instrucciones fuertemente correlacionados, reducir la latencia de acceso de memoria, disminuir el tiempo de penalización por los saltos mal predichos y el tiempo de las paradas en la ejecución debidas a un insuficiente paralelismo a nivel

de instrucción, mediante la eficiente utilización de los recursos del procesador que aumentan el rendimiento final. También se analiza el efecto la introducción de la tecnología *Hyper-Threading* sobre la jerarquía de memoria, el cual va a depender de la aplicación, aunque normalmente se producirá una mejoría en la tasa de aciertos de la misma.

En conclusión la tecnología HT parece efectiva para maximizar los recursos del procesador al ejecutar aplicaciones que se complementan (hacen uso de diferentes recursos a la vez) o que se ayudan (mediante la pre-búsqueda). Por tanto, la clave para conseguir mejoras usando HT radica en la creación inteligente de hilos, capaces de explotar las características antes mencionadas.

## 5.4. Optimización del codificador basado en wavelet usando la tecnología *Hyper-Threading*

En esta sección vamos a desarrollar varias propuestas para mejorar el tiempo de ejecución de nuestro codificador basado en la transformada de wavelet 3D a través del uso de la tecnología *Hyper-Threading* [Bernabé *et al.*, 2004].

Existen varios enfoques a la hora de obtener un algoritmo paralelo a partir de un algoritmos secuencial como es el caso que nos ocupa.

La paralelización automática [Banerjee *et al.*, 1993][Blume *et al.*, 1994] resulta atractiva y prometedora para algunas aplicaciones, pero es incapaz de obtener buenos resultados en muchas otras. Dicha técnica permite aprovechar las ventajas de las arquitecturas paralelas sin que el programador tenga que enfrentarse directamente a la paralelización de un programa secuencial. Las diferentes versiones del compilador de *Intel C/C++* realizan en el siguiente orden un análisis sintáctico del programa, la generación de una representación intermedia adecuada, la estructuración del programa y la identificación de bucles. Llegado a este punto, el compilador realiza un análisis de dependencias que consiste en una serie de tests de creciente precisión y tiempo de ejecución para detectar las dependencias del código y determinar las transformaciones a realizar para generar una representación paralela del programa. También es capaz de realizar una reestructuración del código para permitir paralelización en tiempo de ejecución en algunos casos, por ejemplo para comprobar que dos punteros o *arrays* apuntan a direcciones distintas de memoria. Durante la reestructuración del programa se realizan otras transformaciones u optimizaciones tradicionales, tales como la propagación de constantes y copias, evaluación de expresiones constantes, reodernación de bucles y reconocimiento de reducciones. En la sección 4.5.3 se puso de manifiesto que la vectorización automática no obtenía resultados satisfactorios para el codifica-



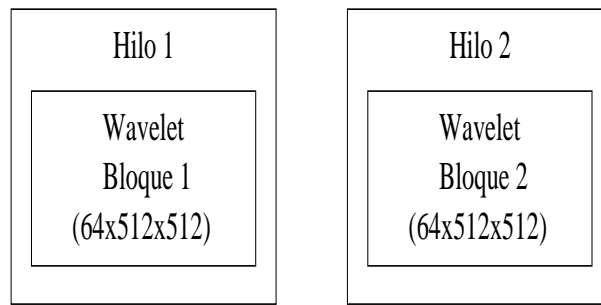


Figura 5.2: Descomposición multi-hilo basada en el dominio de los datos

dor 3D-FWT. En el caso de la paralelización automática, resultados preliminares demuestran un aumento en el tiempo de ejecución del codificador 3D-FWT para un procesador con tecnología Hyper-Threading [Fernández *et al.*, 2004].

Por tanto, es necesario realizar una paralelización manual mediante la intervención directa del programador para extraer el paralelismo existente. A continuación, vamos a desarrollar varias propuestas basadas en un esquema multi-hilo y teniendo en cuenta el conocimiento del funcionamiento de la aplicación para aprovechar las ventajas de la tecnología *Hyper-Threading*.

#### 5.4.1. Descomposición multi-hilo basada en el dominio de los datos

La primera descomposición que evaluaremos es la más tradicional, basada en el dominio de los datos. De hecho, es la que suelen aplicar los compiladores que paralelizan automáticamente. Una descomposición basada en el dominio de los datos implica una división de los datos de la aplicación de tal forma que varios hilos ejecutan simultáneamente la citada aplicación sobre una parte independiente de los datos. En el caso de la compresión de vídeo médico utilizando la transformada de wavelet 3D, cada uno de los hilos va a procesar un bloque de imágenes de la secuencia de vídeo. Supondremos que la complejidad de cada uno de los bloques de imágenes es similar, por lo que los distintos hilos terminarán la ejecución al mismo tiempo. Por ejemplo, dada una secuencia de vídeo de 128 imágenes con una resolución de  $512 \times 512$  pixels, podemos dividirla en dos bloques de 64 imágenes, a los que se le aplica cada una de las fases del codificador basado en la transformada de wavelet 3D. Por lo tanto, en este caso se crean dos hilos que se ejecutan simultáneamente y codifican de forma totalmente independiente cada uno de los bloques de 64 imágenes, tal y como se puede observar en la figura 5.2.

Sin embargo, esta descomposición basada en el dominio de los datos podría tener un potencial limitado sobre el rendimiento final de una aplicación que se ejecuta en un procesador con

tecnología *Hyper-Threading*, ya que existen los siguientes problemas:

- Según el esquema descrito, cada uno de los hilos procesa una parte totalmente independiente y diferente del conjunto de datos de la aplicación. Por lo tanto, tenemos que reservar una cantidad de memoria suficiente para almacenar y procesar los diferentes bloques de imágenes de forma independiente. Esto supone que los dos hilos concurrentes van a iniciar su fase de acceso a memoria intensiva casi al mismo tiempo, creando una mayor presión sobre la jerarquía de memoria.
- Según la descomposición propuesta, cada hilo ejecuta la mismas tareas de forma concurrente sobre una parte independiente de los datos. Por lo tanto, se puede producir una contención en el uso de los recursos del procesador, ya que los distintos hilos van a competir por los mismos recursos en el mismo intervalo de tiempo.

#### 5.4.2. Descomposición multi-hilo basada en una división funcional

En una descomposición funcional, la aplicación se divide en varios procesos o tareas que se ejecutan de forma concurrente sobre un mismo conjunto de datos. En nuestro caso, y con el objetivo de implementar el codificador basado en la transformada de wavelet 3D en una arquitectura con tecnología *Hyper-Threading*, se deben buscar los procesos del codificador que sean funcionalmente independientes, y asignar a cada hilo una de estas tareas para realizar una ejecución simultánea. Finalmente, y siempre que sea necesario, los resultados obtenidos por cada uno de los hilos se sincronizan para obtener los mismos resultados que el correspondiente programa secuencial.

Basándonos en este esquema, vamos a desarrollar varias propuestas para llevar a cabo una descomposición funcional del codificador basado en la transformada de wavelet 3D.

##### 5.4.2.1. Pre-búsqueda de datos y la transformada de wavelet 3D

En los microprocesadores actuales la latencia de memoria es uno de los cuellos de botella clásicos para conseguir un rendimiento adecuado. Con respecto al calculo de la transformada de wavelet, el conjunto de trabajo inicial, es decir la secuencia de vídeo es demasiado grande, aunque se puede predecir el acceso a las diferentes referencias de memoria. De esta forma, en la sección 4.4.3, propusimos añadir instrucciones de pre-búsqueda en el calculo de la transformada de wavelet en la dimensión en el tiempo para reducir la latencia de acceso a memoria y por tanto el tiempo de ejecución.

Basándonos en esta propuesta, podemos extender el modelo a un esquema multi-hilo, en el que un hilo puede realizar el cálculo de la transformada de wavelet, mientras que otro hilo puede ejecutar las instrucciones de pre-búsqueda. De esta manera, el codificador basado en la transformada de wavelet 3D se divide en dos procesos funcionalmente independientes, los cuales se asignan de forma estática a cada uno de los hilos para que se ejecuten simultáneamente. Por lo tanto, un primer hilo, denominado hilo maestro, procesa todas y cada una de las fases del codificador basado en la transformada de wavelet 3D, mientras que el segundo hilo, denominado hilo de ayuda [Collins *et al.*, 2001][Zilles y Sohi, 2001], realiza las instrucciones de pre-búsqueda para traer a la memoria cache los pixels necesarios para el cálculo de la transformada de wavelet en la dimensión en el tiempo.

Para implementar el modelo descrito, el hilo maestro debe activar el hilo de ayuda en el momento que sea necesario llevar a cabo las instrucciones de pre-búsqueda. Por lo tanto, es necesario realizar una sincronización entre los dos hilos para que realicen las operaciones en el instante adecuado. Para llevar a cabo la sincronización, hemos considerado la utilización de un bucle de espera que examina el contenido de una variable compartida entre los dos hilos o el uso de regiones críticas. Esta opción puede introducir un retardo considerable en la ejecución de la aplicación, ya que el hilo de ayuda va a estar consumiendo constantemente recursos del procesador mediante la ejecución de instrucciones inútiles de comparación sobre la variable compartida, ya que dicha variable no cambia su valor durante determinados intervalos de tiempo. Por lo tanto, hemos implementado el hilo de ayuda mediante el uso de semáforos (ver figura 5.3) de la siguiente forma:

- El hilo maestro crea el hilo de ayuda, el cual, consiste en un bucle infinito que realiza instrucciones de pre-búsqueda.
- El hilo de ayuda es una región crítica bloqueada por el hilo maestro que se encuentra esperando una dirección para llevar a cabo la correspondiente instrucción de pre-búsqueda.
- El hilo maestro desbloquea la región crítica cuando proporciona la dirección de un dato al hilo de ayuda, para que este último lleve a cabo la instrucción de pre-búsqueda correspondiente.

Al igual que en la sección 4.4.3 propusimos añadir instrucciones de pre-búsqueda para el cálculo de la dimensión en el tiempo de la transformada de wavelet 3D, hemos implementado varias propuestas en las que el hilo de ayuda lleva a cabo dichas instrucciones de pre-búsqueda. Las propuestas se diferencian en la distancia existente entre los datos que se encuentra procesando la

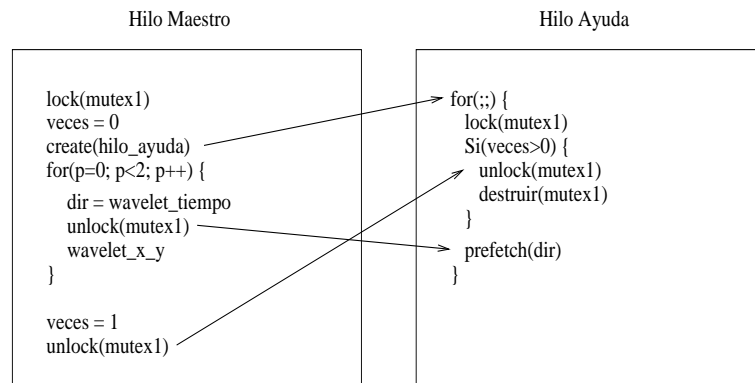


Figura 5.3: Descomposición multi-hilo basada en un hilo de ayuda

aplicación en un instante determinado y los datos a pre-buscar mediante software, y el tiempo que transcurre para llevar a cabo las sucesivas instrucciones de pre-búsqueda. Por lo tanto, tenemos las siguientes propuestas:

- Distancia de 384 o 512 bytes y tiempo de espera de 1000 nseg. Para llevar a cabo las diferentes instrucciones de pre-búsqueda, cuando el codificador esta accediendo a un determinado coeficiente wavelet, se realiza una pre-búsqueda del coeficiente que se encuentra a una distancia de 96 o 128 numeros en punto flotante. Como cada número en punto flotante ocupa cuatro bytes, estaríamos hablando de una distancia de 384 o 512 bytes entre el coeficiente actual y el coeficiente a pre-buscar. Por ejemplo, cuando accedemos al primer coeficiente wavelet, se puede pre-buscar el coeficiente wavelet 96 o el 128, que se encuentran a una distancia de 384 o 512 bytes, respectivamente. El tiempo de espera de 1000 nseg indica el tiempo que transcurre entre que se termina de realizar una instrucción de pre-búsqueda y se invoca a la siguiente, por lo tanto se produce una parada en la ejecución del hilo de ayuda durante el tiempo establecido.
- Distancia desde una hasta cuatro filas de coeficientes y tiempo de espera de 750 nseg. En este caso, las instrucciones de pre-búsqueda van a traer de memoria los coeficientes wavelet que se encuentran en la siguiente fila (distancia de una fila) o sucesivas (distancia de dos, tres o cuatro filas), de la fila de coeficientes que se le esta aplicando la transformada wavelet en el tiempo de una imagen determinada de una secuencia de vídeo. Por ejemplo, dada la fila número uno de una imagen que contiene 512 pixels, se realizan pre-búsquedas de todos los pixels que se encuentran en la fila número dos. Se puede cambiar la distancia para que se pre-busquen los pixels que se encuentran en la fila número tres (distancia de dos filas), cuatro (distancia de tres filas), o cinco (distancia de cuatro filas), para proporcionar un

mayor tiempo a las instrucciones de pre-búsqueda entre las diferentes iteraciones. El tiempo de espera de 750 nseg indica el tiempo que transcurre entre que termina una instrucción de pre-búsqueda de un pixel de una fila determinada y el comienzo de la siguiente.

#### 5.4.2.2. La transformada de wavelet 3D, la cuantificación y la codificación entrópica

En un codificador basado en la transformada de wavelet 3D, el coste computacional principal reside en el calculo de la transformada de wavelet 3D y en el proceso de cuantificación y codificación entrópica. En nuestro caso, a la misma vez que se transforman los coeficientes wavelet en punto flotante a enteros sin signo, con un número de bits que depende del sub-cubo al que pertenece cada uno de los coeficientes, se realiza el *Run-Length* binario 3D inteligente y la codificación hexadecimal, tal y como detallamos en la sección 3.4.

Por lo tanto, la caracterización de las cargas de trabajo del calculo de la transformada de wavelet 3D y de la cuantificación junto con el *Run-Length* binario 3D inteligente y la codificación hexadecimal son totalmente diferentes. La gran mayoría de las instrucciones necesarias para calcular la transformada de wavelet 3D son operaciones en punto flotante que utilizan las extensiones SIMD, mientras que la mayor parte de las instrucciones necesarias para llevar a cabo la cuantificación y parte de la codificación entrópica son operaciones basadas en enteros. De esta forma, cuando se realiza el cálculo de la transformada de wavelet 3D, el procesador utiliza principalmente la unidad de ejecución de punto flotante y mantiene intacta la unidad de ejecución de enteros. En el caso contrario, cuando se realiza la cuantificación, el *Run-Length* binario 3D inteligente y la codificación hexadecimal, el procesador ejecuta la mayor parte de las instrucciones a través de la unidad de ejecución de enteros y apenas utiliza la unidad de ejecución de punto flotante.

Por todo ello, podemos crear un esquema multi-hilo en el cual se asigna un hilo al cálculo de la transformada de wavelet 3D, hilo wavelet, y otro hilo cuantificador, al cálculo de la cuantificación y parte de la codificación entrópica. En este caso, hemos incluido el proceso de umbralización en el primer hilo, ya que el cálculo de la transformada de wavelet 3D consume menos tiempo que el proceso de cuantificación y parte de la codificación entrópica. De esta forma, el hilo wavelet se constituye en el hilo maestro, por lo que crea y bloquea el hilo cuantificador, además de realizar el cálculo de la transformada de wavelet 3D a cada uno de los bloques que componen la secuencia de vídeo. Una vez que se ha realizado el cálculo de la transformada, el hilo wavelet despierta al hilo cuantificador, el cual realiza las operaciones correspondientes sobre los coeficientes transformados, como podemos observar en la figura 5.4 (b).

En cuanto al tiempo de ejecución de cada uno de los hilos, el proceso de cuantificación y parte

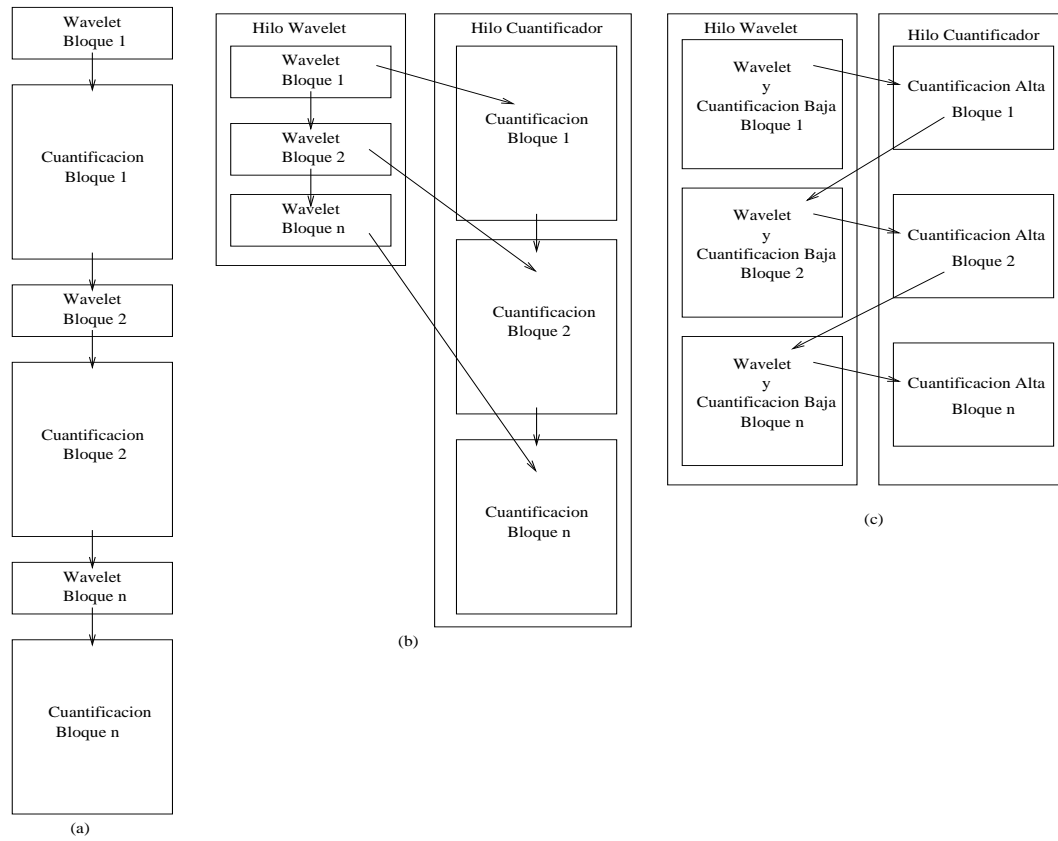


Figura 5.4: Tres métodos para el codificador 3D-FWT (a) Método secuencial; (b) Método funcional multi-hilo; (c) Método funcional balanceado correctamente

de la codificación entrópica necesita un tiempo de procesamiento superior al de la transformada de wavelet 3D, por lo que el hilo cuantificador consume tres cuartas partes del tiempo de CPU mientras que el hilo wavelet consume una cuarta parte del tiempo de CPU. Por lo tanto, necesitamos guardar la salida del hilo wavelet en un lugar diferente de memoria al lugar donde se han realizado los cálculos de la transformada, ya que el cálculo de la transformada de los diferentes bloques se lleva a cabo sobre un lugar común de memoria. De esta forma, se evita la interrupción de la ejecución del hilo wavelet hasta que el hilo cuantificador finalice el procesamiento del bloque en curso, tal y como ocurría en el programa serie, cuyo esquema de procesamiento ilustra la figura 5.4 (a). Este hecho implica un aumento de la cantidad de memoria necesaria para implementar el esquema multi-hilo con respecto al programa serie original.

En cuanto a la sincronización, hemos implementado los diferentes hilos utilizando semáforos, de la misma forma que en el apartado anterior.

#### 5.4.2.3. La transformada de wavelet 3D, la cuantificación y la codificación entrópica alta y baja

En la propuesta descrita en el apartado anterior, se ha presentado un esquema multi-hilo con un hilo wavelet y un hilo cuantificador, en el que la carga de trabajo no se encuentra correctamente balanceada entre los dos hilos, ya que el hilo cuantificador consume tres cuartas partes del tiempo de CPU mientras que el hilo wavelet consume una cuarta parte del tiempo de CPU.

Con respecto al cálculo de la transformada de wavelet 3D cuando se aplica por primera vez sobre un bloque de una secuencia de vídeo, desde el punto de vista de la dimensión en el tiempo, la mitad de las imágenes representan una banda baja (*low*) mientras que la otra mitad constituyen una banda alta (*high*). Una vez que se aplica la transformada en el espacio  $(x, y)$ , cada una de las bandas se divide en otros cuatro sub-cubos, de los cuales sólo se necesita uno de ellos (el sub-cubo que representa la banda baja-baja), para llevar a cabo la segunda iteración de la transformada de wavelet. De esta forma y volviendo al esquema multi-hilo propuesto en el apartado anterior, una vez que el hilo wavelet ejecuta la primera iteración de la transformada de wavelet, el hilo cuantificador puede realizar la cuantificación y la codificación entrópica correspondiente sobre la banda alta sin interferir en el cálculo de la segunda iteración de la transformada de wavelet, que se realiza sobre un sub-cubo de la banda baja. Por lo tanto, una vez que el hilo wavelet realiza el cálculo de la primera iteración de la transformada, dicho hilo maestro activa el hilo cuantificador para que realice los cálculos correspondientes sobre la banda alta. Además, mientras se realiza el cálculo de la cuantificación y la codificación entrópica sobre la banda alta, el hilo wavelet calcula la segunda iteración de la transformada de wavelet y la correspondiente cuantificación y codificación entrópica sobre la banda baja. Por lo tanto, en el nuevo esquema multi-hilo, el hilo maestro o wavelet ejecuta la transformada de wavelet y la cuantificación y la codificación entrópica sobre la banda baja, mientras que el hilo cuantificador realiza la cuantificación y la codificación entrópica sobre la banda alta.

En el esquema propuesto en el apartado anterior, la salida del hilo wavelet se copiaba en un lugar diferente de memoria al que se realizaba el cálculo de la transformada de wavelet para poder procesar el siguiente bloque, sin esperar a la finalización de la cuantificación y la codificación entrópica del bloque en curso. En este nuevo esquema no es necesario realizar la citada copia, ya que el hilo cuantificador va a terminar su trabajo sobre la banda alta antes que el hilo wavelet empiece a procesar un nuevo bloque. Por lo tanto, los dos hilos se van ejecutar simultáneamente sobre el mismo lugar de memoria en el que se encuentran almacenados los datos. Además, la utilización de la CPU va a estar mejor balanceada entre los dos hilos que en la propuesta anterior, tal y como se puede observar en la figura 5.4 (c). A pesar del correcto balanceo de la carga y para

Level 1	Cache L1 instr, Trace Cache Cache L1 datos, 8 KB, 4 vías, 64 byte línea
Level 2	Cache L2, 512 KB, 8 vías, 128 byte línea
Level 3	1024 Mbytes DRAM

Tabla 5.1: Características de la jerarquía de memoria

evitar posibles errores en la ejecución del esquema multi-hilo, hemos protegido con un semáforo el lugar de memoria donde se realizan los cálculos, para que el hilo wavelet no empiece con un nuevo bloque hasta que el hilo cuantificador termine de procesar el bloque actual.

## 5.5. Evaluación de resultados

En esta sección vamos a analizar las diferentes propuestas descritas anteriormente mediante la evaluación del tiempo de ejecución del codificador basado en la transformada de wavelet 3D, y manteniendo la tasa de compresión y la calidad del vídeo reconstruido del codificador original.

### 5.5.1. Entorno de trabajo

La evaluación de las diferentes propuestas se han llevado a cabo sobre un biprocesador *Intel Xeon* con tecnología *Hyper-Threading* y una frecuencia de  $2GHz$ . Las principales características de la jerarquía de memoria se pueden observar en la tabla 5.1. Como sistema operativo se ha utilizado Linux 2.2.18 – 3smp. Las distintas propuestas se han escrito usando el lenguaje de programación *C* y se han compilado con el compilador Intel *C/C++ v.6.0*. Además, hemos utilizado los hilos POSIX del sistema operativo Linux para desarrollar las aplicaciones multi-hilo.

Para evaluar las diferentes propuestas presentadas en la sección 5.4, hemos medido el tiempo transcurrido que proporciona el sistema operativo para las siguientes configuraciones:

- Ejecución serie o simple sobre un procesador con tecnología *Hyper-Threading*, desactivando dicha tecnología.
- Ejecución paralela o basada en dos threads sobre un procesador con tecnología *Hyper-Threading*.
- Ejecución paralela o basada en dos threads sobre un multiprocesador con dos procesadores, desactivando la tecnología *Hyper-Threading* de cada uno de ellos.



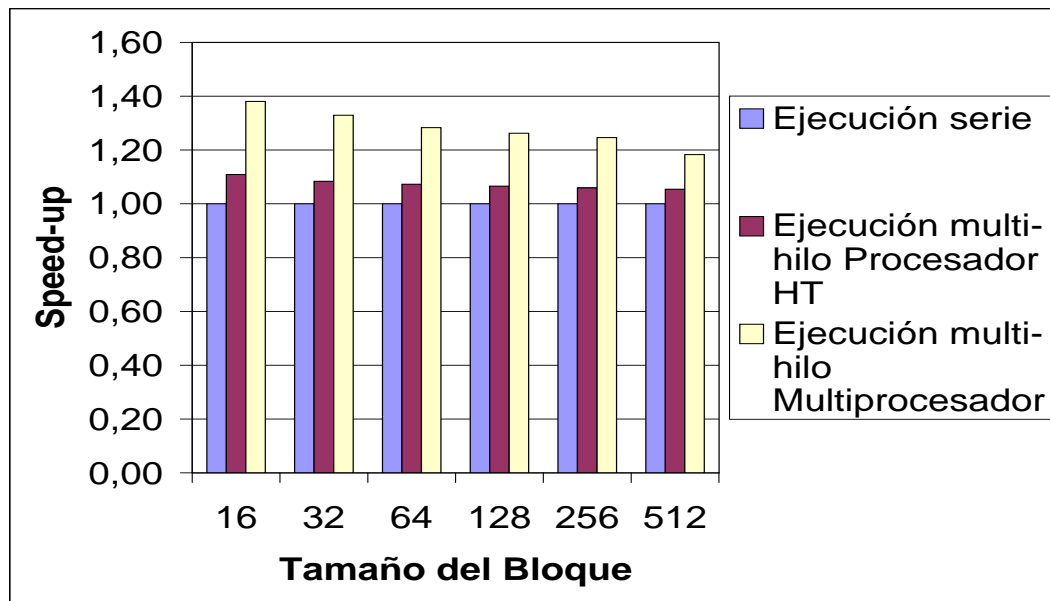


Figura 5.5: *Speed-ups* para la descomposición multi-hilo basada en el dominio de los datos

Hemos calculado el *speed-up* del codificador basado en la transformada de wavelet 3D, como el cociente de la media aritmética del tiempo transcurrido de 100 ejecuciones de la aplicación mediante una ejecución serie y una ejecución paralela sobre un procesador con tecnología *Hyper-Threading* o sobre un multiprocesador con dos procesadores sin tecnología *Hyper-Threading*. El coeficiente de variación (la desviación típica dividida por la media aritmética), que mide la dispersión de la muestra es inferior al 2,5 % en todos los casos.

### 5.5.2. Descomposición basada en el dominio de los datos

Para evaluar el tiempo de ejecución, hemos comprimido la secuencia de vídeo médica *Corazón* de 128 imágenes y una resolución de  $512 \times 512$  pixels, codificada en escala de grises (8 bits por pixel). Los resultados se pueden generalizar a cualquier secuencia de vídeo del mismo tamaño, ya que el tiempo de ejecución es independiente de la entrada.

La figura 5.5 muestra los *speed-ups* con respecto al codificador basado en la transformada de wavelet 3D desarrollado en el capítulo 4, es decir, la vectorización por columnas que incluye el resto de optimizaciones (técnica de división en rectángulos utilizando la transformada de wavelet solapada, reuso de cálculos, uso de las extensiones SIMD, desenrollado de bucle de la dimensión *Tiempo* y pre-búsqueda de datos), para comprimir la secuencia de vídeo médico *Corazón*. Los

resultados se presentan para diferentes tamaños de bloque de 16 imágenes de  $512 \times 16$  pixels hasta  $512 \times 512$  pixels. En la figura, se ha omitido el número de imágenes ya que siempre se mantiene constante, y sólo se indica el tamaño de la dimensión  $Y$ , ya que en las técnicas de división en rectángulos aplicadas el tamaño de la dimensión  $X$  se mantiene constante, es decir, 512 pixels en cada una de las configuraciones.

En primer lugar, podemos observar que la descomposición basada en el dominio de los datos implementada con dos hilos, donde cada hilo procesa un bloque independiente de 64 imágenes de  $512 \times 512$  pixels en un procesador con tecnología *Hyper-Threading*, introduce un beneficio significativo, ya que se alcanzan unos *speed-ups* entre 1,05 y 1,11 con respecto a la ejecución serie del codificador. Para el tamaño de bloque óptimo ( $512 \times 32 \times 16$ ), es decir, la configuración con un menor tiempo de ejecución, se obtiene un *speed-up* de 1,08. En este caso, la configuración óptima es distinta de la que se obtuvo en el capítulo 4, ya que las características del procesador y de la jerarquía de memoria son totalmente diferentes. Dicho tamaño de bloque óptimo equivale a 32 y 0,5 veces el tamaño de las caches L1 y L2, respectivamente.

Como ya comentamos en el apartado 5.4.1, el rendimiento de esta propuesta se encuentra limitado por la forma de realizar la descomposición. Cada hilo necesita acceder simultáneamente a los mismos recursos para poder procesar un bloque de imágenes por lo que se puede producir una contención en el uso de los recursos del procesador. Además, la ejecución simultánea de los dos bloques provoca que los dos hilos realicen un acceso intensivo a memoria de forma casi simultánea, provocando una mayor presión sobre la jerarquía de memoria que en la versión secuencial del codificador. Sin embargo, la carga de trabajo se encuentra correctamente balanceada entre los dos procesadores lógicos (50 %), ya que cada uno de los procesadores lógicos del único procesador físico tiene asignado un hilo. En cuanto a la programación, ésta es sencilla de llevar a cabo ya que cada hilo trabaja sobre una parte independiente de los datos sin la necesidad de utilizar semáforos para proteger secciones críticas.

En cuanto a la ejecución paralela del esquema multi-hilo en el multiprocesador, se obtienen *speed-ups* entre 1,18 y 1,38 (1,33 para el tamaño de bloque óptimo) con respecto a la ejecución serie del codificador, tal y como podemos observar en la figura 5.5. El único objetivo de la realización de las pruebas en el multiprocesador consiste en la determinación de la importancia de la contribución del esquema multi-hilo en un sólo procesador con tecnología *Hyper-Threading*, ya que establecen un límite máximo.

### 5.5.3. Descomposición basada en una división funcional

En este caso, se mide el tiempo de ejecución de la compresión de la secuencia de vídeo médico *Corazón* compuesta de 64 *frames* y una resolución de  $512 \times 512$  pixels codificada en escala de grises (8 bits por pixel).

Los resultados para el esquema basado en la pre-búsqueda de datos y la transformada de wavelet 3D no se muestran debido a que las diferentes alternativas implementadas obtienen unos *speed-ups* insignificantes (1,01 o 1,02), o incluso peores que la versión secuencial del codificador. En este esquema multi-hilo, el hilo maestro se encarga de realizar el cálculo computacional, por lo que esta activo la mayor parte del tiempo (aproximadamente el 100 % del tiempo), mientras que el hilo de ayuda es el responsable de llevar a cabo las instrucciones de pre-búsqueda en el momento determinado. Por lo tanto, los resultados dependen del equilibrio de los siguientes factores:

- El hilo maestro dispone de una menor cantidad de recursos del procesador que en la versión secuencial para realizar la misma tarea. Por lo tanto, el esquema propuesto implica un aumento en el tiempo de ejecución del hilo maestro debido a la contención en el uso de los recursos del procesador.
- La ejecución sucesiva de las diferentes instrucciones de pre-búsqueda reducen la latencia de acceso a memoria para obtener los datos de la aplicación, por lo que el tiempo de ejecución del hilo maestro disminuye gracias a la correcta ejecución de dichas instrucciones de pre-búsqueda.
- El procesador Intel Xeon dispone de un mecanismo propio de pre-búsqueda basado en hardware que realiza de forma automática instrucciones de pre-búsqueda. El hardware de pre-búsqueda examina el patrón de acceso de datos de la aplicación en curso e intenta adelantar 256 bytes, es decir, dos líneas de la cache L2. Por lo tanto, el procesador ejecuta automáticamente instrucciones de pre-búsqueda para mejorar el rendimiento de la aplicación y ocultar la latencia de acceso a memoria. Este hecho implica una atención especial por parte del programador para introducir las instrucciones de pre-búsqueda en el hilo de ayuda, ya que se puede producir un conflicto entre las instrucciones de pre-búsqueda hardware y las software. Por ello, las diferentes alternativas propuestas realizan las pre-búsquedas correspondientes a una distancia superior a 256 bytes para no interferir con la pre-búsqueda por hardware. Además, el patrón de acceso a memoria del algoritmo de la transformada de wavelet es bastante regular, por lo que el hardware de pre-búsqueda obtiene un buen rendimiento.

- El ancho de banda del bus es limitado,  $3,2 \frac{\text{Gbytes}}{\text{seg}}$  en el procesador Pentium IV, por lo que hay que tener cuidado para que las diferentes peticiones a memoria entre las que se encuentran el acceso a los datos actuales por parte de la aplicación y las instrucciones de pre-búsqueda por hardware o por software, no produzcan una saturación en el uso del bus en un instante determinado.

En todos los casos implementadas y comentados en la sección 5.4.2.1, los resultados mejoran ligeramente o tienen un comportamiento peor que la versión secuencial del codificador. Por lo tanto, a partir de estas propuestas hemos realizado varias modificaciones para disminuir el número de pre-búsquedas o el tiempo que transcurre entre una pre-búsqueda y la siguiente. Sin embargo, los resultados obtenidos no son lo suficientemente satisfactorios debido a una saturación en el uso del bus o a conflictos con las pre-búsquedas realizada por el propio procesador a través del hardware correspondiente. De esta forma, podemos deducir que la inclusión del hilo de pre-búsqueda tendría un mayor sentido en algoritmos con un patron de acceso irregular (listas, árboles), donde el mecanismo hardware de pre-búsqueda no pudiera actuar con eficacia.

La figura 5.6 muestra los *speed-ups* obtenidos por el codificador basado en la transformada de wavelet 3D para la compresión de la secuencia de vídeo médico *Corazon* de 64 frames de  $512 \times 512$  pixels. Se presentan resultados para dos de los tres esquemas propuestos en la sección 5.4.2: la transformada de wavelet 3D, la cuantificación y la codificación entrópica que aparece en la figura como *Alta&Baja*, y la transformada de wavelet 3D, la cuantificación y la codificación entrópica alta y baja que se presenta en la figura como *Alta*.

En cuanto a la propuesta denominada *Alta&Baja*, en la figura 5.6, podemos observar que la versión del codificador ejecutada sobre un procesador HT obtiene *speed-ups* entre 1,12 y 1,14 con respecto a la versión secuencial. La configuración que obtiene el menor tiempo de ejecución, es decir, el tamaño de bloque óptimo ( $512 \times 32 \times 16$ ), obtiene un *speed-up* de 1,12. Estos resultados son mejores que los que se obtienen con una descomposición basada en el dominio de los datos, ya que en una descomposición funcional no es necesario reservar memoria para procesar y almacenar dos bloques de forma independiente y no se produce una contención en el uso de los recursos del procesador. De hecho, las dos principales tareas van a estar la mayor parte del tiempo utilizando unidades de ejecución diferentes, ya que el hilo wavelet usa principalmente la unidad de ejecución de punto flotante, mientras que el hilo cuantificador va a disponer de la unidad de ejecución de enteros para la emisión de la mayoría de sus instrucciones. Sin embargo, la carga de trabajo no se encuentra correctamente balanceada entre los dos procesadores lógicos, ya que el hilo cuantificador ocupa tres cuartas partes del tiempo de CPU, mientras que el hilo wavelet solo dispone de una cuarta parte del tiempo de CPU. Debido a este desbalanceo de la carga, y como hemos comentado

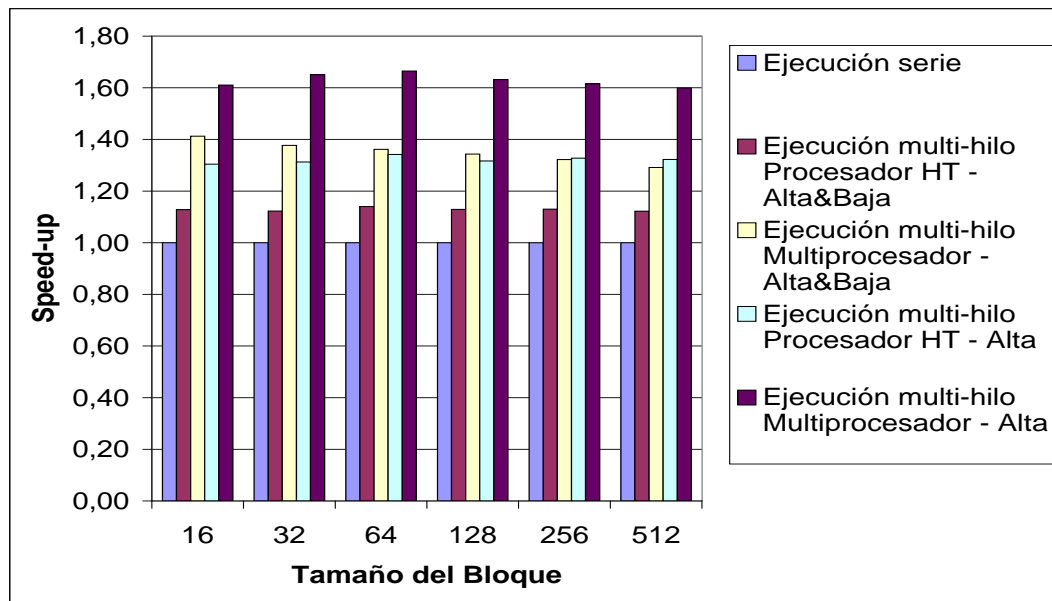


Figura 5.6: *Speed-ups* para la descomposición multi-hilo basada en una división funcional

en la sección 5.4.2.2, los resultados que genera el hilo wavelet deben ser guardados en un lugar diferente de memoria a aquel donde se realiza el cálculo de la transformada de wavelet, para que el hilo wavelet pueda procesar el siguiente bloque sin tener que esperar a que el hilo cuantificador termine de realizar las operaciones correspondientes sobre el bloque en curso.

Debido al desbalanceo de la carga entre el hilo wavelet y el hilo cuantificador, se propone el esquema descrito en la sección 5.4.2.3 que aparece en la figura 5.6 como Alta. En este caso, el hilo wavelet ocupa un 60 % del tiempo de CPU, mientras que el hilo cuantificador dispone del 40 % del tiempo de CPU. Con esta división de la carga, la versión ejecutada sobre el procesador con tecnología *Hyper-Threading* obtiene *speed-ups* entre 1,30 y 1,34 con respecto a la versión secuencial del codificador, mientras que el *speed-up* para el tamaño de bloque que genera el menor tiempo de ejecución es 1,31. Estos resultados son sustancialmente mejores que las propuestas anteriores y se deben a que el uso de los recursos del procesador se reparte de forma eficiente entre los dos hilos. Además, y al contrario que en la propuesta anterior, no es necesario tener que realizar una copia de los resultados del hilo wavelet en un lugar diferente al que se calcula la transformada para poder procesar el siguiente bloque.

En la figura 5.6, también podemos observar los *speed-ups* para las propuestas Alta&Baja y Alta sobre un multiprocesador y con respecto a la versión secuencial. Obviamente, los resultados son mejores que en el caso de un sólo procesador SMT, ya que los *speed-ups* se sitúan desde 1,29 a

1,41 para la propuesta Alta&Baja, y desde 1,60 a 1,67 para la propuesta Alta. Los *speed-ups* para los tamaños de bloque óptimos son 1,38 y 1,65 respectivamente. Estos resultados confirman que el aumento de la productividad de los procesadores con tecnología *Hyper-Threading* es significativo con respecto al número de recursos que es necesario duplicar, ya que la principal ventaja de dicha tecnología consiste en el aumento de la utilización de los recursos del procesador mediante la planificación dinámica de las unidades funcionales entre varios hilos, como se ha demostrado en la última propuesta basada en una descomposición funcional.

En cuanto a la programación de las propuestas basadas en una descomposición funcional, los dos hilos van a estar ejecutándose la mayor parte del tiempo de forma simultánea, por lo que es necesario realizar cambios significativos sobre el codificador secuencial. Dichas modificaciones no son triviales y requieren un análisis y un conocimiento profundo del funcionamiento del codificador.

## 5.6. Conclusiones

En este capítulo, hemos investigado el comportamiento de un procesador de propósito general con características de una arquitectura SMT como es el procesador Intel Xeon con tecnología *Hyper-Threading* para obtener beneficios en las aplicaciones multimedia. Para ello, hemos desarrollado varias propuestas sobre nuestro codificador basado en la transformada de wavelet 3D para evaluar el rendimiento de la citada tecnología.

En primer lugar, hemos propuesto una descomposición basada en el dominio de los datos en la que dos hilos ejecutan de forma simultánea cada una de las fases del codificador sobre partes de datos totalmente independientes. En este caso, un procesador con tecnología *Hyper-Threading* obtiene unos *speed-ups* entre 1,05 y 1,11 con respecto a un procesador sin dicha tecnología. Aunque los resultados aportan una mejora sustancial con respecto a un procesador sin HT, se encuentran limitados por la propia descomposición.

En segundo lugar, hemos desarrollado varias propuestas basadas en una descomposición funcional en la que dos hilos ejecutan de forma simultánea tareas diferentes del codificador sobre un único conjunto de datos. Según este tipo de descomposición, hemos desarrollado las siguientes técnicas:

- Pre-búsqueda de datos y la transformada de wavelet 3D. Hemos propuesto un esquema multi-hilo en el que un hilo maestro se encarga de llevar a cabo el cálculo de la transformada de wavelet 3D y un hilo de ayuda tiene que realizar las pre-búsquedas de los pixels necesarios

para conseguir el cálculo de la transformada. El objetivo consiste en reducir la latencia de acceso a memoria del codificador. Sin embargo, hemos implementado varias propuestas, las cuales, se diferencian en la distancia entre el dato actual y el dato a pre-buscar, y el tiempo de espera para realizar las pre-búsquedas consecutivas, en las que los resultados no mejoran significativamente el rendimiento del codificador en un procesador sin tecnología *Hyper-Threading*.

- La transformada de wavelet 3D, la cuantificación y la codificación entrópica. En este caso, se realiza una división funcional básica en la que un hilo se encarga de calcular la transformada de wavelet y otro hilo realiza la cuantificación y la codificación entrópica. Los resultados sobre un procesador HT obtienen unos *speed-ups* entre 1,12 y 1,14 con respecto a un procesador sin tecnología HT. Por lo tanto, el rendimiento es mayor que en una descomposición basada en el dominio de los datos.
- La transformada de wavelet 3D, la cuantificación y la codificación entrópica alta y baja. En esta última división funcional se parte de la propuesta anterior para optimizar algunos aspectos como el balanceo de la carga y el paralelismo entre los dos hilos descritos. El primer hilo calcula la transformada de wavelet 3D sobre la secuencia de vídeo, la cuantificación y la codificación entrópica sobre la banda baja, mientras que el segundo hilo realiza la cuantificación y la codificación entrópica sobre la banda alta. Esta última propuesta obtiene los mejores resultados sobre un procesador HT, ya que consigue unos *speed-ups* de 1,30 y 1,34 con respecto a un procesador sin tecnología HT.

Por lo tanto, el uso de la tecnología *Hyper-Threading* aporta una mejora significativa al rendimiento del codificador basado en la transformada de wavelet 3D. Las propuestas que se basan en una descomposición funcional de las tareas a realizar y que requieren un mayor conocimiento del comportamiento de la aplicación obtienen un mayor rendimiento que aquellas propuestas que se basan en una descomposición en el dominio de los datos. Los compiladores actuales tienen grandes dificultades para extraer de forma automática el paralelismo más adecuado para cada aplicación. En este caso, hemos utilizado los hilos POSIX del sistema operativo Linux para indicar al compilador como realizar la paralelización y obtener el máximo beneficio. Con el objetivo de disminuir el trabajo del programador, se podría obtener un modelo equivalente en el que se siguiera ayudando al compilador [Fernández *et al.*, 2004], que consistiría en la utilización de las directivas de OpenMP [ope, 2002] que nos permiten realizar una paralelización a más alto nivel sin tener que modificar el algoritmo secuencial en la mayoría de aplicaciones. Finalmente, hay que destacar que las distintas conclusiones obtenidas en este capítulo se pueden generalizar y llevar a

cabo en aplicaciones multimedia similares como la codificación de vídeo MPEG-2 y MPEG-4.



## Capítulo 6

### Conclusiones y trabajo futuro

---

En este capítulo vamos a resumir las principales conclusiones que hemos obtenido a lo largo de los diferentes capítulos que componen la presente tesis doctoral, así como las publicaciones obtenidas y la financiación disfrutada para tal fin. Por último, describiremos algunas líneas de trabajo futuro que pueden complementar el trabajo realizado.

#### 6.1. Conclusiones

En esta tesis se ha propuesto un nuevo codificador de vídeo médico basado en el uso de la transformada de wavelet 3D, capaz de ejecutarse en una arquitectura monoprocesador en tiempo real.

En primer lugar, se ha realizado una descripción pormenorizada de los tres procesos básicos que componen un codificador de este tipo: transformación de los datos, cuantificación de los coeficientes transformados y codificación entrópica de los coeficientes cuantificados. Para cada una de las fases mencionadas se han mostrado diferentes alternativas u opciones. En la transformada matemática se han presentado la transformada de Fourier, la transformada discreta del coseno y la transformada de wavelet. Hemos dedicado especial atención al funcionamiento y uso de los algoritmos 1D-FWT, 2D-FWT y 3D-FWT, así como a los fundamentos matemáticos de la transformada de wavelet. En la cuantificación se han descrito las principales formas de cuantificación existentes: uniforme, no uniforme y adaptativa. Por último, en la codificación entrópica se ha introducido la codificación de Shanon-Fano, Huffman y aritmética, así como el funcionamiento de la codificación *Run-Length*.

A continuación, se ha llevado a cabo una descripción de las principales características y del

funcionamiento de los estándares basados en la DCT (JPEG, JPEG-LS, MPEG-1, MPEG-2 y MPEG-4, H.261 y H.263) y en la DWT (JPEG-2000 y MPEG-4). Asimismo, se han introducido los principales métodos basados en la DWT para la compresión de imágenes y secuencias de vídeo. Debido a la importancia de la DWT en los últimos años, se ha realizado un resumen de los principales usos de la misma (no orientados a la compresión), entre los que se encuentran la eliminación de ruido, la detección de comportamientos autosimilares en series de tiempo, la recuperación de imágenes basadas en el contexto y el diseño de un sistema de seguridad basado en un reconocimiento facial.

El estudio y análisis de los conocimientos anteriores nos ha permitido proponer un nuevo codificador de vídeo médico con pérdida de información basado en la 3D-FWT. El codificador obtiene unos excelentes resultados desde el punto de vista de la tasa de compresión y la calidad de las secuencias de vídeo reconstruidas. Para ello, se han analizado diversos factores como la función wavelet madre utilizada, el número de veces que se debe aplicar dicha función y el orden para aplicar la transformada en las diferentes dimensiones, todo ello en la fase de aplicación de la transformada de wavelet. La umbralización se lleva a cabo mediante la combinación de la técnica del percentil y la eliminación de un número determinado de bits menos significativos. En cuanto a la cuantificación, se ha implementado un cuantificador que asigna un número de bits a cada coeficiente wavelet, dependiendo del número de veces que se aplica la transformada en la sub-banda en la que se encuentra el citado coeficiente. El codificador entrópico se compone de un codificador *Run-Length* binario y un codificador de Huffman.

La construcción del codificador paso a paso nos ha permitido conocer y poder realizar nuevas propuestas para superar las limitaciones del mismo. Por tanto y con el objetivo de mejorar las técnicas existentes, se han propuesto varias optimizaciones sobre el cuantificador y el codificador entrópico. Dichas propuestas tienen las propiedades comunes de aprovechar mejor las características propias de un codificador 3D-FWT, mejorar la tasa de compresión y no degradar la calidad de las secuencias de vídeo. El codificador *Run-Length* binario 3D inteligente propone comprimir las cadenas de ceros y unos consecutivos siguiendo el orden impuesto por la dimensión en el tiempo. Dicho orden provoca una mayor cantidad de ceros y unos consecutivos, y por tanto una mayor compresión. El cuantificador asigna un número de bits a un coeficiente dependiendo del sub-cubo al que pertenece dicho coeficiente y teniendo en cuenta el número de veces que se ha aplicado la transformada en el mencionado sub-cubo. Dicha asignación permite realizar un mejor ajuste del número de bits que se asigna a cada coeficiente, una vez aplicada la 3D-FWT. La codificación hexadecimal permite la codificación de cadenas de longitud ilimitada de ceros o unos consecutivos a la hora de realizar el *Run-Length*. Asimismo, se proponen técnicas para

codificar de forma eficiente las cadenas de longitud desde uno hasta siete ceros consecutivos y las cadenas de unos consecutivos. Por último, la utilización de la codificación aritmética con 16 símbolos permite realizar una mayor compresión que en el caso del codificador de Huffman. La integración de las propuestas comentadas han supuesto que el codificador resultante obtenga un mejor comportamiento, en lo que se refiere a tasa de compresión y calidad de las secuencias de vídeo reconstruidas, que nuestro codificador base, el estándar MPEG-2 y el codificador EZW. En los tres casos, cuando la calidad de las secuencias de vídeo reconstruidas es la misma para los distintos codificadores, la tasa de compresión del codificador mejorado supera notablemente al resto de codificadores en un rango que varía desde un 40 % hasta un 70 % con relación al codificador base, desde un 21 % hasta un 54 % con respecto a EZW y desde un 51 % hasta un 119 % con respecto al estándar MPEG-2.

La siguiente aportación ha consistido en la implementación de la compresión y transmisión en tiempo real de la transformada de wavelet 3D para la codificación de vídeo médico en arquitecturas monoprocesador. Para conseguir tal fin, se han propuesto una serie de técnicas, cuya integración ha supuesto un primer paso hacia la convergencia entre las aplicaciones multimedia y las extensiones multimedia. Por todo ello, se ha propuesto el uso de las técnicas de *blocking* teniendo en cuenta las particularidades del cálculo de la transformada de wavelet y con el objetivo de explotar la jerarquía de memoria. El reuso de operaciones en punto flotante en las regiones limítrofe de los diferentes sub-cubos ha permitido reducir el número de instrucciones en punto flotante ejecutadas y la presión sobre la jerarquía de memoria al producirse una disminución en el número de accesos a memoria. El uso de las extensiones SIMD a través de una paralelización manual, ya que la paralelización automática no obtiene los beneficios esperados, ha explotado el paralelismo a nivel de datos mediante la realización en paralelo de varias operaciones en punto flotante. El desenrollado manual del bucle de la dimensión en el tiempo y la inserción de instrucciones de pre-búsqueda ha originado un descenso en el número de fallos de cache y una mejor explotación del paralelismo a nivel de instrucción. Por último, la propuesta denominada vectorización por columnas ha determinado una utilización más óptima de las extensiones multimedia y ha vuelto a reducir el número de instrucciones ejecutadas en punto flotante. Para cada una de las técnicas mencionadas se ha realizado la evaluación correspondiente y se han analizado sus ventajas e inconvenientes. Dicha evaluación determina que la integración de las técnicas propuestas consiguen una aceleración del tiempo de ejecución de 4,25 con respecto al cálculo original de la transformada de wavelet 3D. Todas estas propuestas se han desarrollado teniendo en cuenta su fácil y sencilla implementación por lo que todas ellas podrían generalizarse y aplicarse en aplicaciones multimedia con características similares como los estándares de codificación de vídeo MPEG-2 o MPEG-4.

La introducción de las nuevas arquitecturas paralelas de un único chip CMP y SMT nos ha llevado a realizar un estudio pormenorizado de la forma de implementar el concepto SMT por parte de los procesadores *Pentium IV Xeon* mediante la tecnología *Hyper-Threading*, así como a la realización de un resumen de las principales características, ventajas e inconvenientes que proporciona dicha tecnología. Asimismo, se ha realizado una descripción de los principales trabajos que se han llevado a cabo haciendo uso de la mencionada tecnología HT.

Los conocimientos anteriores sobre SMT y la tecnología *Hyper-Threading* nos han permitido realizar varias propuestas basadas en un esquema multi-hilo que aprovechan las ventajas de un procesador HT. Las distintas técnicas se han diferenciado en la forma de descomponer la aplicación original. Por un lado, tenemos una descomposición basada en el dominio de los datos, en la que varios hilos ejecutan simultáneamente la misma aplicación sobre una parte independiente de los datos. En particular, se ha propuesto una descomposición en la que los dos hilos ejecutan simultáneamente cada uno de los procesos del codificador sobre un conjunto de datos independiente. Por otro lado, podemos realizar una descomposición funcional, en la que la aplicación se divide en varios procesos o tareas que se ejecutan de forma concurrente sobre un mismo conjunto de datos. Para esta última descomposición hemos desarrollado varias propuestas que utilizan un esquema multi-hilo y que se aprovechan del conocimiento del funcionamiento del codificador de vídeo médico 3D-FWT. Una primera propuesta divide el trabajo de cada uno de los hilos entre el cálculo de la transformada de wavelet 3D y la realización de instrucciones de pre-búsqueda para llevar a cabo el cálculo anterior. La segunda propuesta realiza una división funcional básica en la que un hilo calcula la transformada de wavelet 3D y otro hilo procesa las fases de cuantificación y codificación entrópica. Por último, la propuesta que obtiene los mejores resultados realiza una división de tareas de tal forma que un hilo se encarga de llevar a cabo la transformada de wavelet sobre la totalidad de la secuencia de vídeo y de realizar los procesos de cuantificación y codificación entrópica sobre la banda baja, mientras que un segundo hilo se encarga de ejecutar los procesos de cuantificación y codificación entrópica sobre la banda alta. La consecución de un mejor balanceo de la carga y un mayor paralelismo entre los dos hilos ha permitido que se obtengan en un procesador *Hyper-Threading* unos *speed-ups* entre 1,30 y 1,34 con respecto a un procesador sin la mencionada tecnología.

En resumen, en esta tesis se ha presentado un codificador de vídeo médico basado en la transformada de wavelet 3D que aumenta la tasa de compresión entre un 40 % y un 70 % para una misma calidad de las secuencias de vídeo reconstruidas, con respecto al codificador base. Las diferentes propuestas presentadas han conseguido una aceleración del tiempo de ejecución de un 363 %.

## 6.2. Resultados de la tesis

Durante el desarrollo de la presente tesis doctoral se han llevado a cabo diferentes publicaciones en distintas revistas y congresos. A continuación, exponemos brevemente y en orden cronológico los principales datos de cada una de ellas.

- Gregorio Bernabé, José González, José M. García y José Duato. “**A New Lossy 3D Wavelet Transform for High Quality Compression of Medical Video**”. IEEE EMBS International Conference on Information Technology Applications in Biomedicine (ITAB-ITIS 2000), IEEE Networking the World, ISBN: 0-7803-6449-X, Washington-EEUU, Noviembre 2000.

En este artículo se propone el codificador base presentado en la sección 3.3. Se evalúan las secuencias de vídeo *Corazon* y *Cateter* y se realiza una comparación del rendimiento con MPEG-2.

- Gregorio Bernabé, José González, José M. García y José Duato. “**Applying the 3-D Wavelet Transform to Transmit Medical Video in Telemedicine**”. 5th World Congress on the Internet in Medicine (MEDNET 2000), IOS Press, ISBN: 1-58603-146-5, Bruselas-Belgica, Noviembre 2000.

Este trabajo consiste en un *abstract* en el que se introduce la propuesta comentada en el artículo anterior.

- Gregorio Bernabé, José González, José M. García y José Duato. “**Enhancing the Entropy Encoder of a 3D-FWT for High-Quality Compression of Medical Video**”. IEEE International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS 2001), IEEE Signal Processing Society, ISBN: 0-9716692-0-1, Nashville-EEUU, Noviembre 2001.

Este trabajo presenta las diferentes optimizaciones sobre el cuantificador y el codificador entrópico presentadas en la sección 3.4. La evaluación de las secuencias de vídeo *Corazon* y *Mano* determinan un mejor comportamiento del codificador mejorado con respecto al codificador base, MPEG-2 y EZW.

- Jorge A. Vélez, Andrés A. Navarro, Luis E. Múnuera y Gregorio Bernabé. “**A Software Architecture for Virtual Simulation of Endoscopic Surgery**”. Telemedicine Journal and e-Health. Vol. 8, Nº 2, ISSN: 1530-5627, Verano 2002.

Se trata de un *abstract* en el que se introduce una nueva arquitectura software apropiada para Internet y para la simulación 3D siguiendo los pasos de la cirugía otorrinolaringológica. Las técnicas de compresión utilizadas son las propuestas en la presente tesis doctoral.

- Gregorio Bernabé, José González, José M. García y José Duato. “**Memory Conscious 3D Wavelet Transform**”. 28th EUROMICRO Conference (EUROMICRO 2002), IEEE Computer Society, ISBN: 0-7695-1787-0, Dortmund-Alemania, Septiembre 2002. También publicado en las Actas de las XIII Jornadas de Paralelismo, Universidad de Lleida, ISBN: 84-8409-159-7, Lleida-España, Septiembre 2002.

En este artículo se desarrollan las técnicas de división en bloques (cubo y rectangular), así como el reuso de operaciones en punto flotante para solventar el problema de la gran cantidad de memoria que necesita el algoritmo de la transformada de wavelet 3D para la codificación y decodificación de vídeo médico. La parte de evaluación analiza la mejora de las técnicas propuestas y el comportamiento de la memoria cache.

- Gregorio Bernabé, José M. García y José González. “**Reducing 3D Wavelet Transform Execution Time through the Streaming SIMD Extensions**”. 11th EUROMICRO Conference on Parallel, Distributed and Network-Based Processing (EUROMICRO-PDP 2003), IEEE Computer Society, ISBN: 0-7695-1875-3, Genova-Italia, Febrero 2003.

Este artículo completa las diferentes propuestas presentadas en el capítulo 4 de la tesis. Se propone el uso de las extensiones multimedia, desenrollado de bucles, pre-búsqueda de datos y la vectorización por columnas para permitir la compresión y transmisión de la 3D-FWT en tiempo real en arquitecturas monoprocesador.

- Jorge A. Vélez, Andrés A. Navarro, C. A. De La Roche, A. Kopec, L.E. Munera, Gregorio Bernabé, C. Bermudez y J. F. Jimenez. “**A Virtual Surgical Telesimulations in Micrographic Dermatologic Surgery (MOHS)**”. 17th International Congress on Computer Assisted Radiology and Surgery (CARS 2003), Elsevier Science, ISBN: 0-444-51387-6, Londres-Reino Unido, Junio 2003.

Este artículo propone una arquitectura software para el aprendizaje a distancia de la cirugía dermatológica. La transmisión y compresión de imágenes y vídeo se basa en la transformada de wavelet y en los métodos propuestos en esta tesis.

- Andrés A. Navarro, Jorge A. Vélez, J.E. Satizabal, Luis E. Munera y Gregorio Bernabé. “**Virtual Surgical Telesimulations in Ophthalmology**”. 17th International Congress

on Computer Assisted Radiology and Surgery (CARS 2003), Elsevier Science, ISBN: 0-444-51387-6, Londres-Reino Unido, Junio 2003.

En este trabajo se describe una arquitectura basada en una serie de módulos interactivos para el tratamiento quirúrgico a distancia de operaciones oftalmológicas.

- Andrés A. Navarro, Jorge A. Vélez, J. E. Satizabal, Luis E. Múnuera, Gregorio Bernabé, C. Bermudez y J.F. Jimenez. “**Telemedicine Services and Virtual Surgical Telesimulation in Ophthalmology**”. Telemedicine Journal and e-Health. Volumen 9, Suplemento I, ISSN: 1530-5627, 2003.

Este *abstract* define y presenta los primeros resultados de una arquitectura software apropiada para Internet y para la simulación 3D, a través de un sistema de telemedicina que sigue los pasos de la cirugía oftalmológica.

- Gregorio Bernabé, José M. García y José González. “**An Efficient 3D Wavelet Transform on Hyper-Threading Technology**”. Enviado al 13th EUROMICRO Conference on Parallel, Distributed and Network-Based Processing (EUROMICRO-PDP 2005). Technical Report UM-DITEC-2004-02.

En este trabajo se estudia el comportamiento de un procesador SMT (*Pentium IV Xeon Hyper-Threading*) y se desarrollan diferentes propuestas sobre el codificador 3D-FWT basadas en un esquema multi-hilo, presentadas en la sección 5.4, para aprovechar eficazmente este tipo de arquitecturas.

- Ricardo Fernández, José M. García, Gregorio Bernabé, Manuel E. Acacio. “**Optimizing a 3D-FWT Video Encoder for SMPs and HyperThreading Architectures**”. Enviado al 13th EUROMICRO Conference on Parallel, Distributed and Network-Based Processing (EUROMICRO-PDP 2005).

En el trabajo anterior la propuesta basada en una descomposición funcional obtiene el mejor comportamiento en una arquitectura SMT. En este artículo se evalúan varias alternativas para implementar dicha propuesta usando paralelización automática, las directivas de OpenMP y los POSIX, desde el punto de vista del tiempo de ejecución, facilidad de programación y mantenimiento del código obtenido.

Asimismo, hemos enviado un artículo a una revista especializada, el cual se encuentra aceptado y pendiente de publicación.

- Gregorio Bernabé, José M. García y José González. “**Reducing 3D Fast Wavelet Transform Execution Time Using Blocking and the Streaming SIMD Extensions**”. Aceptado y pendiente de publicación en el Journal of VLSI Signal Processing.

El trabajo muestra las diferentes propuestas realizadas para conseguir la compresión y transmisión en tiempo real de la 3D-FWT en arquitecturas monoprocesador. Las técnicas descritas son: *blocking*, reuso de operaciones en punto flotante, uso de las extensiones multimedia, desenrollado de bucles, pre-búsqueda de datos y vectorización por columnas

### 6.3. Financiación disfrutada

El trabajo expuesto se ha desarrollado en el marco de los siguientes proyectos de investigación:

- **TIC97-0897-C04-02**. “Desarrollo de una Red de Estaciones de Trabajo de Altas Prestaciones y Bajo Coste”. Proyecto subvencionado por la Comisión Interministerial de Ciencia y Tecnología (CICYT), desde el 1 de agosto de 1997 hasta el 31 de julio de 2000. Entidades participantes: Universidad de Murcia, Universidad de Castilla La Mancha y Universidad Politécnica de Valencia. Investigadores responsables: Dr. José Duato Marín (proyecto coordinado) y Dr. José Manuel García Carrasco (subproyecto de la Universidad de Murcia). Cantidad financiada: 46,000,000 ptas. en total, de las cuales 8,395,000 ptas. corresponden al subproyecto de la Universidad de Murcia.
- **TIC2000-1151-C07-03**. “Mejora de las Prestaciones y Servicios Ofrecidos por las Redes de Computadores Personales. Desarrollo de Aplicaciones Multimedia Distribuidas”. Proyecto subvencionado por la Comisión Interministerial de Ciencia y Tecnología (CICYT), desde el 28 de diciembre de 2000 hasta el 27 de diciembre de 2003. Entidades participantes: Universidad de Murcia, Universidad de Castilla La Mancha, Universidad Politécnica de Valencia, Universidad de Valencia y Universidad Jaime I. Investigadores responsables: Dr. José Duato Marín (proyecto coordinado) y Dr. José Manuel García Carrasco (subproyecto de la Universidad de Murcia). Cantidad financiada: 100,240,000 ptas. en total, de las cuales 13,720,000 ptas. corresponden al subproyecto de la Universidad de Murcia.
- **TIC2003-08154-C06-03**. “Mejora de las Prestaciones y Servicios Ofrecidos por las Redes de Computadores Personales (II)”. Proyecto subvencionado por la Comisión Interministerial de Ciencia y Tecnología (CICYT), desde diciembre de 2003 hasta diciembre de 2006. Entidades participantes: Universidad de Murcia, Universidad de Castilla La Mancha, Universidad



Politécnica de Valencia, Universidad de Valencia y Universidad Jaime I. Investigadores responsables: Dr. José Duato Marín (proyecto coordinado) y Dr. José Manuel García Carrasco (subproyecto de la Universidad de Murcia). Cantidad financiada: 985,000 euros en total, de las cuales 309,000 euros corresponden al subproyecto de la Universidad de Murcia.

## 6.4. Trabajo futuro

Las diferentes propuestas desarrolladas en el capítulo 5 para aprovechar las arquitecturas SMT se han implementado mediante los hilos POSIX del sistema operativo Linux. Como ya hemos comentado, la realización de un modelo equivalente mediante las directivas de OpenMP [ope, 2002] puede descargar de trabajo al programador, ya que se realiza una programación a más alto nivel en la que no es necesario realizar grandes modificaciones sobre el código secuencial para realizar la paralelización [Fernández *et al.*, 2004]. Relacionado con este mismo tema, sería interesante evaluar los diferentes modelos construidos con OpenMP y POSIX en distintos compiladores, que contengan soporte para ambas opciones y para la programación de las extensiones multimedia, para determinar la implementación que obtiene mayores beneficios.

Una opción muy interesante para aprovecharse de las ventajas de la paralelización automática, incluso en los casos en los que el compilador no dispone de la suficiente información, sería la paralelización especulativa [Rauchwerger y Padua, 1999][Chen y Olukotun, 2003][Prabhu y Olukotun, 2003]. Dicha técnica resulta especialmente atractiva en arquitecturas donde los costes derivados de las comunicaciones son pequeños como en el caso de las arquitecturas CMP. La creación de modelos basados en la ejecución de hilos especulativos y su análisis supondría una buena contrapartida para los modelos desarrollados en este trabajo.

En el capítulo 4 se han presentado varias propuestas basadas en el uso de las extensiones multimedia para reducir el tiempo de ejecución del codificador de vídeo médico 3D-FWT. El diseño y creación de nuevas instrucciones multimedia basadas en el conocimiento de tareas específicas y comunes de las aplicaciones multimedia y en la arquitectura del procesador, podría suponer una reducción en el número de ciclos necesarios para ejecutar dichos procesos. Desde un punto de vista hardware, se podrían proponer nuevos paradigmas y arquitecturas multimedia específicas para ejecutar el codificador de vídeo médico 3D-FWT y enfrentar los resultados con los que se consiguen en arquitecturas de propósito general.

Como ha quedado patente a lo largo del desarrollo de la presente tesis doctoral, uno de los trabajos complementarios a realizar consistiría en la generalización de las técnicas propuestas en aplicaciones multimedia de características similares. La creación de una librería gráfica que tuviera

como objetivo la optimización y la mejora de dichas aplicaciones significaría la culminación de tal fin.

El modelo computacional del procesador de una tarjeta gráfica (GPU) se centra en los datos y no en las instrucciones, evitando el cuello de botella que constituye la memoria en los procesadores de propósito general. La utilización de la potencia computacional de las tarjetas gráficas puede ayudar a la unidad central de proceso a ejecutar aplicaciones y descargar de trabajo a esta última. El estudio y análisis de la ejecución del codificador de vídeo médico 3D-FWT sobre una GPU sería una línea de trabajo a desarrollar que sirviera de base para otras aplicaciones multimedia.

## Referencias

---

- «OpenMP Archicecture Review Board». *OpenMP C and C++ Aplication Program Interface, version 2.0*, 2002.
- «The Colombian Telemedicine Centre», 2004.  
Disponible en <http://www.colombiantelemed.org>
- ACHEROY, M. y GRANDJEAN, S.: «METEOSAT Image Compression using the Wavelet Transform». *Informe técnico*, Royal Military Academy, 1994.
- ANDERSON, E.; BAI, Z.; BISCHOF, C.; DEMMEL, J.; DONGARRA, J.; CROZ, J. DU; GREENBAUM, A.; HAMMARLING, S.; KENNEY, A. MC y SORENSEN, D.: «LAPACK: A portable Linear Algebra Library for High-Performance Computers». *Tech. Report CS-90-105, (LAPACK Working Note #20)*, Univ. Of Tennessee, Knoxville, 1990.
- ANDERSON, MARK CHARLES: *Task-Oriented Lossy Compression of Magnetic Resonance Images*. Tesina o Proyecto, Simon Fraser University, 1995.
- ANH, LAN y TEGGINMATH, SHOBA: «Applications of Multimedia in Library and Information Services». *Workshop Information Netwoks for the future*, 2002.
- ANTONINI, MARC; BARLAUD, MICHEL; MATHIEU, PIERRE y DAUBECHIES, INGRID: «Image Coding Using Wavelet Transform». *IEEE Transactions on Image Processing*, 1992, **1(2)**, pp. 205–220.
- ASBUN, EDUARDO; SALAMA, PAUL; SHEN, KE y DELP, EDWARD J.: «Very Low Bit Rate Wavelet-Based Scalable Video Compression». *IEEE International Conference on Image Processing*, 1998.
- BANERJEE, UTPAL; EIGENMANN, RUDOLF; NICOLAU, ALEXANDRU y PADUA, DAVID A.: «Automatic Program Parallelization». *Proceedings of the IEEE*, 1993, **81(2)**, pp. 211–243.

- BASITH, SHANAWAZ A. y DONE, STEPHEN R.: «Digital Video, MPEG and Associated Artifacts». *Informe técnico*, Department of Computing and Electrical Engineering, Imperial College, 1996.
- BATTISTA, STEFANO; CASALINO, FRANCO y LANDE, CLAUDIO: «MPEG-4: A Multimedia Standard for the Third Millenium, Part 1». *IEEE Multimedia*, 1999, **6(4)**, pp. 74–83.
- BATTISTA, STEFANO; CASALINO, FRANCO y LANDE, CLAUDIO: «MPEG-4: A Multimedia Standard for the Third Millenium, Part 2». *IEEE Multimedia*, 2000, **7(1)**, pp. 76–84.
- BERNABÉ, GREGORIO; GARCÍA, JOSÉ M. y GONZÁLEZ, JOSÉ: «Reducing 3D Wavelet Transform Execution Time through the Streaming SIMD Extensions». *Proceedings of the 11th Euromicro Conference on Parallel Distributed and Network based Processing*, 2003.
- BERNABÉ, GREGORIO; GARCÍA, JOSÉ M. y GONZÁLEZ, JOSÉ: «An Efficient 3D Wavelet Transform on Hyper-Threading Technology». *Enviado al 13th Euromicro Conference on Parallel Distributed and Network based Processing. Technical Report UM-DITEC-2004-02*, 2004.
- BERNABÉ, GREGORIO; GONZÁLEZ, JOSÉ; GARCÍA, JOSÉ M. y DUATO, JOSÉ: «Applying the 3-D Wavelet Transform to Transmit Medical Video in Telemedicine». *5th World Congress on the Internet in Medicine (MEDNET 2000)*, 2000a.
- BERNABÉ, GREGORIO; GONZÁLEZ, JOSÉ; GARCÍA, JOSÉ M. y DUATO, JOSÉ: «A New Lossy 3-D Wavelet Transform for High-Quality Compression of Medical Video». *Proceedings of IEEE EMBS International Conference on Information Technology Applications in Biomedicine*, 2000b, pp. 226–231.
- BERNABÉ, GREGORIO; GONZÁLEZ, JOSÉ; GARCÍA, JOSÉ M. y DUATO, JOSÉ: «Enhancing the Entropy Encoder of a 3D-FWT for High-Quality Compression of Medical Video». *Proceedings of IEEE International Symposium for Intelligent Signal Processing and Communication Systems*, 2001.
- BERNABÉ, GREGORIO; GONZÁLEZ, JOSÉ; GARCÍA, JOSÉ M. y DUATO, JOSÉ: «Memory Conscious 3D Wavelet Transform». *Proceedings of the 28th Euromicro Conference. Multimedia and Telecommunications*, 2002.
- BIK, AART; GIRKAR, MILIND; GREY, PAUL y TIAN, XINMIN: «Efficient Exploitation of Parallelism on Pentium III and Pentium IV Processor-Based Systems». *Intel Technology Journal*, *Q1*, 2001.

- BILGIN, ALI; ZWEIG, GEORGE y MARCELLIN, MICHAEL W.: «Efficient Lossless Coding of Medical Image Volumes Using Reversible Integer Wavelet Transforms». *Data Compression Conference*, 1998, pp. 428–437.
- BLUME, WILLIAN; EIGENMANN, RUDOLF; HOEFLINGER, JAY; PADUA, DAVID; PETERSEN, PAUL; RAUCHWERGER, LRENCE y TU, PENG: «Automatic Detection of Parallelism: A Grand Challenge for High Performance Computing». *IEEE Parallel and Distributed Technology: Systems and Applications*, 1994, **2(3)**, pp. 37–47.
- BOLIEK, MARTIN: «JPEG2000 Part I Final Committee Draft Version 1.0». *ISO/IEC JTC1/SC29/WG1 N1646R*, 2000.
- BRAMBILLA, C.; VENTURA, A. DELLA; GAGLIARDI, I. y SCHETTINI, R.: «Multiresolution Wavelet Transform and Supervised Learning for Content-Based Image Retrieval». *IEEE International Conference on Multimedia Computing and Systems*, 1999.
- BRISLAWN, CRISTOPHER M.: «Fingerprints Go Digital». *Notices of the AMS*, 1994, **42(11)**.
- BUCCIGROSSI, ROBERT W. y SIMONCELLI, EERO P.: «Progressive Wavelet Image Coding Based on a Conditional Probabilistic Model». *International Conference on Acoustics, Speech and Signal Processing*, 1997.
- CALDERBANK, A. R.; DAUBECHIES, INGRID; SWELDENS, WIM y YEO, BOON-LOCK: «Wavelet Transforms that Map Integers to Integers». *Applied and Computational Harmonic Analysis*, 1998, **5(3)**, pp. 332–369.
- CARLSON, D.A.; CASTELINO, R.W. y MUELLER, R.O.: «Multimedia extensions for a 550-MHz RISC microprocessor». *IEEE Journal of Solid-State Circuits*, 1997, **32(11)**, pp. 1618–1624.
- CHANG, S. GRACE y VETTERLI, MARTIN: «Spatial Adaptive Wavelet Thresholding for Image Denoising». *International Conference on Image Processing*, 1997.
- CHANG, S. GRACE; YU, BIN y VETTERLI, MARTIN: «Image Denoising Via Lossy Compression and Wavelet Thresholding». *International Conference on Image Processing*, 1997.
- CHEN, JIANHUA; HU YU, TIAN y WU, XIAOLIN: «Non-Embedded Wavelet Image Coding Scheme». *International Conference on Image Processing*, 1997.
- CHEN, MICHAEL y OLUKOTUN, KUNLE: «The jrpm system for dynamically parallelizing java programs». *30th Annual International Symposium on Computer Architecture*, 2003, pp. 443–446.

- CHEN, YEN-KUANG; HOLLIMAN, MATTHEW; DEBES, ERIC; ZHELTOV, SERGEY; KNYAZEY, ALEXANDER; BRATANOV, STANISLAV; BELENOV, ROMAN y SANTOS, ISHMAEL: «Media Applications on Hyper-Threading Technology». *Intel Technology Journal Q1*, 2002.
- CHEN, YINGWEI y PEARLMAN, WILLIAM A.: «Three-Dimensional Subband Coding of Video Using the Zero-Tree Method». *Proc. of SPIE-Visual Communications and Image Processing*, 1996, pp. 1302–1310.
- CHOI, S. J.: *Three-Dimensional Subband/Wavelet Coding of Video with Motion Compensation*. Tesina o Proyecto, Renssalaer Polytechnic Institute, Troy, NY, 1996.
- CHUI, CHARLES K.: *An Introduction to Wavelets, volume 1 of Wavelet Analysis and Its Applications*. Academic Press, 1992.
- COLLINS, J.D.; WANG, H.; TULLSEN, D.M.; HUGHES, C.M.; LEE, Y.F.; LAVERY, D. y SHEN, J.P.: «Speculative Precomputation: Long-Range Prefetching of Delinquent Loads». *In 28th International Symposium on Computer Architecture*, 2001, pp. 14–25.
- CONTE, G.; TOMMESANI, S. y ZANICHELLI, F.: «The long and winding road to high-performance image processing with MMX/SSE». *Proceedings of the Fifth IEEE International Workshop on Computer Architectures for Machine Perception*, 2000.
- CORPORATION, INTEL: «IA-32 Intel Architecture Software Developer's Manual», 2002a.  
Disponible en <http://developer.intel.com/>
- CORPORATION, INTEL: «Intel C/C++ Compiler for Linux», 2002b.  
Disponible en <http://www.intel.com/software/products/compiler/c50/linux>
- DAUBECHIES, INGRID: *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics, 1992.
- DIEFENDORFF, KEITH; DUBEY, PRADEEP; HOCHSPRUNG, RON y SCALES, HUNTER: «AltiVec Extension to PowerPC Accelerates Media Processing». *IEEE Micro*, 2000, **20(2)**, pp. 85–95.
- DONGARRA, J.; CROZ, J. DU; DUFF, I. S. y HAMMARLING, S.: «A set of Level 3 Basic Linear Algebra Subprogram». *ACM Trans. Math. Soft*, 1988, **14**, pp. 1–17.
- DONOHU, D.: «Nonlinear Wavelet Methods for Recovery of Signals, Densities, and Spectra from Indirect and Noisy Data». *Different Perspectives on Wavelets, Proceeding of Symposia in Applied Mathematics*, 1993, **47**, pp. 173–205.

- EGGERS, SUSAN J.; EMER, JOEL S.; LEVY, HENRY M.; LO, JACK L.; STAMM, REBECCA L. y TULLSEN, DEAN M.: «Simultaneous Multithreading: A platform for next-generation process». *IEEE Micro*, 1997, pp. 12–18.
- FAVOR, G.: «AMD-K6 MMX Enhanced Processor». *Microprocessor Forum*, 1997.
- FERNÁNDEZ, RICARDO; GARCÍA, JOSÉ M.; BERNABÉ, GREGORIO y ACACIO, MANUEL E.: «Optimizing a 3D-FWT Video Encoder for SMPs and HyperThreading Architectures». *Enviado al 13th Euromicro Conference on Parallel Distributed and Network based Processing*, 2004.
- GRAPS, AMARA: «An Introduction to Wavelets». *IEEE Computational Science and Engineering*, 1995, **2(2)**.
- GROUP, TECHNICAL TASK: «Multimedia Telematics Standars: Technical Notes and Tutorial», 1997.  
Disponible en <http://www.imc.exec.nhs.uk/ttg/tutorial2/frames1.htm>
- HAMMOND, LANCE; HUBBERT, BEN; SIU, MICHAEL; PRABHU, MANOHAR; CHEN, MIKE y OLUKOTUN, KUNLE: «The Stanford Hydra CMP». *IEEE MICRO Magazine*, 2000.
- HAMMOND, LANCE; NAYFEH, BASEM A. y OLUKOTUN, KUNLE: «A Single-Chip Multiprocessor». *IEEE Computer*, 1997.
- HELLER, DON: «Rabbit: A Performance Counters Library for Intel/AMD processors and Linux», 2001.  
Disponible en <http://www.scl.ameslab.gov/Projects/Rabbit/>
- HILTON, MICHAEL L.; JAWERTH, BJÖRN D. y SENGUPTA, AYAN: «Compressing Still and Moving Images with Wavelets». *Multimedia Systems*, 1994, **2(3)**.
- HUFFMAN, D. A.: «A method for the construction of minimum-redundancy codes». *Proceedings of IRE*, 1952, **40**, pp. 1098–1101.
- IHM, INSUNG y PARK, SANGHUN: «Wavelet-Based 3D Compression Scheme for Very Large Volume Data». *Graphics Interface '98*, 1998, pp. 107–116.
- IHM, INSUNG; PARK, SANGHUN y KOO, G.: «Wavelet-Based 3D Compression Schemes for the Visible Human Dataset and Their Applications». *Visible Human Project Conference '98*, 1998.

- ITU-T: «Video Coding for Low Bit Rate Communication or POTS Videoconferencing (below 64 Kbps)». *ITU-T Recommendation H.263*, 1996.
- (JPEG/JBIG), ISO/IEC JTC1/SC29 WG1: «Lossless and near-lossless coding of continuous tone still images (JPEG-LS), FCD 14495». *IEEE Transactions on Image Processing*, 1992, **1(2)**, pp. 244–256.
- KILLIAN, EARL: «MIPS Extension for Digital Media with 3D». *Slides presented at Microprocessor Forum*, 1996.
- KIM, BEONG-JO y PEARLMAN, WILLIAM A.: «An Embedded Wavelet Video Coder Using Three-Dimensional Set Partitioning in Hierarchical Trees (SPIHT)». *Proceedings of Data Compression Conference*, 1997.
- KIM, BEONG-JO y PEARLMAN, WILLIAM A.: «Low-Delay Embedded 3-D Wavelet Color Video Coding with SPIHT». *Visual Communication and Image Processing. Proc. SPIE*, 1998, **3309**, pp. 955–964.
- KIM, YOUNGSEOP y PEARLMAN, WILLIAM A.: «Lossless Volumetric Medical Image Compression». *Proc. of SPIE, Applications of Digital Image Processing*, 1999, **3808**, pp. 305–312.
- KIM, YOUNGSEOP y PEARLMAN, WILLIAM A.: «Stripe-Based SPIHT Lossy Compression of Volumetric Medical Images for Low Memory Usage and Uniform Reconstruction Quality». *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, 2000.
- KRONANDER, T.: *Some Aspects of Perception Based Image Coding*. Tesina o Proyecto, University of Linköping, Sweden, 1989.
- LAM, MONICA S.; ROTHBERG, EDWARD E. y WOLF, MICHAEL E.: «The Cache Performance and Optimizations of Blocked Algorithms». *Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IV)*, 1991.
- LANGDON, G. G. y RISSANEN, J.: «Compression of black-white images with arithmetic coding». *IEEE Trans. Communications*, 1981, **29(6)**, pp. 858–867.
- LEE, RUBY B.: «Subword Parallelism with MAX-2». *IEEE Micro*, 1996, **16(4)**, pp. 51–59.
- LEMPEL, ODED; PELEG, ALEX y WEISER, URI: «Intel's MMX Technology - A New Instruction Set». *Proceedings of 42nd IEEE Computer Society International Conference*, 1997.



- LEVY, IAN K. y WILSON, ROLAND: «Three Dimensional Wavelet Transform for Video Compression». *IEEE International Conference on Multimedia Computing and System*, 1999.
- LEWIS, A. S. y KNOWLES, G.: «Image Compression Using the 2-D Wavelet Transform». *IEEE Transactions on Image Processing*, 1992, **1**(2), pp. 244–256.
- MAGRO, WILLIAN; PETERSEN, PAUL y SHAH, SANJIV: «Hyper-Threading Technology: Impact on Compute-Intensive Workloads». *Intel Technology Journal Q1*, 2002.
- MALLAT, STEPHANE: «A theory for multiresolution signal descomposition: The wavelet representation». *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1989, **11**(7), pp. 674–693.
- MARCELLIN, MICHAEL W.; GORMISH, MICHAEL J.; BILGIN, ALI y BOLIEK, MARTIN P.: «An Overview of JPEG-2000». *Data Compression Conference*, 2000.
- MARCUELLO, PEDRO y GONZÁLEZ, ANTONIO: «Exploiting speculative thread-level parallelism on a SMT processor». *HPCN Europe*, 1999, pp. 754–763.
- MARR, DEBORAH T.; BINNS, FRANK; HILL, DAVID L.; HINTON, GLENN; KOUFATY, DAVID A.; MILLER, J. ALAN y UPTON, MICHAEL: «Hyper-Threading Technology Architecture and Microarchitecture». *Intel Technology Journal Q1*, 2002.
- MARSH, ANDY: «EUROMED: The Creation of a global Telemedical Information Society». *Virtual Medical Worlds Magazine*, 1998.  
Disponible en <http://www.hoise.com/vmw/analysis/euromedbuildingblocks/telemed.html>
- MARTÍNEZ, JOSÉ M.: «MPEG-7 Overview (version 9)». *ISO/IEC JTC1/SC29/WG11N525*, 2003.
- MARTÍNEZ, MARIANO: «Reconocimiento de Caras usando la Transformada de Wavelet». *Proyecto fin de carrera*, Facultad de Informática. Universidad de Murcia, 2002.
- MEYER, YVES: *Wavelets: Algorithms & Applications*. SIAM Press, 1993.
- MOLLER, OLE y HEGLAND, MARKUS: «Parallel performance of Fast Wavelet Transform». *Proceedings of International Journal of High Speed Computing*, 2000, **11**(1), pp. 55–74.
- MONRO, D. M.; BASSIL, B. E. y DICKSON, G. J.: «Orthonormal Wavelets With Balanced Uncertainty». *School of Electronic and Electrical Engineering, University of Bath, England (ICIP)*, 1996.

- MPEG: «ISO/IEC 11172». *ISO/IEC JTC1/SC29/WG11*, 1993.
- MPEG: «ISO/IEC 13818». *ISO/IEC JTC1/SC29/WG11*, 1994.
- MURAKI, SHIGERU: «Approximation and rendering of volume data using wavelet transforms». *Proceedings of Visualization*, 1992, pp. 21–28.
- MURAKI, SHIGERU: «Multiscale Volume Representation by a DoG Wavelet». *IEEE Transactions on Visualization and Computer Graphics*, 1995, **1(2)**, pp. 109–116.
- NACHTERGAELE, LODE; LAFRUIT, GAUTHIER; BORMANS, JAN y BOLSENS, IVO: «Fast Software Implementation of the MPEG-4 reversible integer wavelet transform on Pentium MMX, Sharc ADSP and Trimedia TM1000». *Proceedings of Packet Video*, 2000.
- NAVARRO, ANDRÉS A.; VÉLEZ, JORGE A.; SATIZABAL, J. E.; MÚNUERA, LUIS E. y BERNABÉ, GREGORIO: «Telemedicine Services and Virtual Surgical Telesimulation in Ophthalmology». *Telemedicine Journal and e-Health*, 2003a, **9**.
- NAVARRO, ANDRÉS A.; VÉLEZ, JORGE A.; SATIZABAL, J. E.; MÚNUERA, LUIS E. y BERNABÉ, GREGORIO: «Virtual Surgical Telesimulations in Ophtalmology». *17th International Congress on Computer Assisted Radiology and Surgery (CARS 2003)*, 2003b.
- NELSON, MARK y GAILLY, JEAN-LOUP: *The Data Compression Book*. M&T Books, 2ª edición, 1996.
- OBERMAN, STUART y WEBER, GREG FAVOR FRED: «AMD 3DNow! Technology: Architecture and Implementations». *IEEE Micro*, 1999, **19(2)**, pp. 37–48.
- OHM, J.R.: «Three-dimensional subband coding with motion compensation». *IEEE Transactions on Image Processing*, 1994, **3(5)**, pp. 559–571.
- PRABHU, MANOHAR K. y OLUKOTUN, KUNLE: «Using thread-level speculation to simplify manual parallelization». *9th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2003, pp. 1–12.
- PRICEWATERHOUSECOOPERS: «HealthCast Tactics: A Blueprint for the Future», 2003.
- RABINOVITCH, IDO: «High Quality Image Compression Using the Wavelet Transform». *International Conference on Image Processing*, 1997.

- RANGANATHAN, PARTHASARATHY; ADVE, SARITA y JOUPPI, NORMAN P.: «Performance of Image and Video Processing with General-Purpose Processors and Media ISA Extensions». *International Symposium on Computer Architecture*, 1999.
- RAUCHWERGER, LAWRENCE y PADUA, DAVID: «The LRPD test: speculative run-time parallelization of loops with privatization and reduction parallelization». *IEEE Transactions on Parallel and Distributed systems*, 1999, **2(10)**, pp. 160–180.
- RISSANEN, J. y LANGDON, G. G.: «Arithmetic coding». *IBM Journal of Research and Development*, 1979, **23(2)**, pp. 149–162.
- SAHA, SUBHASIS y VEMURI, RAO: «Adaptative Wavelet Coding of Multimedia Images». *ACM Multimedia Conference*, 1999.
- SAHA, SUBHASIS y VEMURI, RAO: «Analysis-based Adaptive Wavelet Filter Selection in Lossy Image Coding Schemes». *IEEE International Symposium on Circuits and Systems*, 2000.
- SAID, AMIR y PEARLMAN, WILLIAM A.: «Image Compression Using the Spatial-Orientation Tree». *IEEE Intl. Symp. Circuits and Systems*, 1993, pp. 279–282.
- SAID, AMIR y PEARLMAN, WILLIAM A.: «An image multiresolution representation for lossless and lossy compression». *IEEE Transactions on Image Processing*, 1996a, **5(9)**, pp. 1303–1310.
- SAID, AMIR y PEARLMAN, WILLIAM A.: «A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees». *IEEE Transactions on Circuits and System for Video Technology*, 1996b, **6**.
- SANTA-CRUZ, DIEGO y EBRAHIMI, TOURADJ: «A Study of JPEG 2000 Still Image Coding versus Others Standards». *Proc. of the X European Signal Processing Conference*, 2000.
- SCARGLE, JEFFREY; STEIMAN-CAMERON, THOMAS; YOUNG, KARL; DONOHO, DAVID L.; CRUTCHFIELD, JAMES P. y IMAMURA, JAMES: «The Quasi-Periodic Oscillations and Very Low Frequency Noise of Scorpius X-1 as Transient Chaos: A Dripping Handrail?». *Astrophysical Journal*, 1993, **411**.
- SEMICONDUCTORS, PHILIPS: «An Introduction to Very-Long Instruction Word Computer Architecture», 1999.  
Disponible en <http://www.semiconductors.philips.com/acrobat/other/vliw-wp.pdf>
- SHAPIRO, JEROME M.: «Embedded Image Coding Using Zerotrees of Wavelets Coefficients». *IEEE Transactions on Signal Processing*, 1993, **41(12)**, pp. 3445–3462.

- SHEN, KE y DELP, EDWARD J.: «Wavelet Based Rate Scalable Video Compression». *IEEE Transactions on Circuits and System for Video Technology*, 1999, **19**(1), pp. 109–122.
- SHI, YUN Q. y SUN, HUIFANG: *Image and Video Compression for Multimedia Engineering. Fundamentals, Algorithms, and Standards*. CRC Press, 2000.
- SMITH, JAMES E. y SOHI, GURINDAR S.: «The Microarchitecture of Superscalar Processors». *Proceedings of the IEEE*, 1995.
- SMITH, STEVEN W.: *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997.
- SODANI, AVINASH y SOHI, GURINDAR S.: «Dynamic Instruction Reuse». *24th International Symposium on Computer Architecture*, 1997.
- STOLLNITZ, ERIC J.; DEROSE, TONY D. y SALESIN, DAVID H.: «Wavelets for computer graphics: A primer, part 1». *IEEE Computer Graphics and Applications*, 1995a, **15**(3), pp. 76–84.
- STOLLNITZ, ERIC J.; DEROSE, TONY D. y SALESIN, DAVID H.: «Wavelets for computer graphics: A primer, part 2». *IEEE Computer Graphics and Applications*, 1995b, **15**(3), pp. 75–85.
- TAUBMAN, DAVID: «High Performance Scalable Image Compression with EBCOT». *IEEE Transactions on Image Processing*, 2000, **9**(7).
- TAUBMAN, DAVID y ZAKHOR, A.: «Multirate 3-D subband coding of video». *IEEE Transactions on Image Processing*, 1994, **3**(5), pp. 572–598.
- THAKKAR, SHREEKANT y HUFF, TOM: «Internet Streaming SIMD Extensions». *IEEE Computer*, 1999, **32**, pp. 26–34.
- TREMBLAY, MARC; O'CONNOR, J. MICHAEL; NARAYANAN, VENKATESH y HE, LIANG: «VIS Speeds New Media Processing». *IEEE Micro*, 1996, **16**(4), pp. 10–20.
- TULLSEN, DEAN M.; EGGERS, SUSAN J. y LEVY, HENRY M.: «Simultaneous Multithreading: Maximizing On-Chip Parallelism». *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, 1995.
- UENO, I.; ONO, F.; YANAGIYA, T.; KIMURA, T. y YOSHIDA, M.: «Proposal of the Arithmetic Coder for JPEG2000». *ISO/IEC JTC1/SC29/WG1 N1420*, 1998.
- VIDAKOVIC, BRANI y MÜLLER, PETER: «Wavelets for Kids. A Tutorial Introduction». *Informe técnico*, Duke University, 1994.

- VÉLEZ, JORGE A.; NAVARRO, ANDRÉS A.; MÚNUERA, LUIS E. y BERNABÉ, GREGORIO: «A Software Architecture for Virtual Simulation of Endoscopic Surgery». *Telemedicine Journal and e-Health*, 2002, **8**(2).
- VÉLEZ, JORGE A.; NAVARRO, ANDRÉS A.; ROCHE, C. A. DE LA; KOPEC, A.; MÚNUERA, LUIS E.; BERNABÉ, GREGORIO; BERMUDEZ, C. y JIMENEZ, J. F.: «A Virtual Surgical Tele-simulations in Micrographic Dermatologic Surgery (MOHS)». *17th International Congress on Computer Assisted Radiology and Surgery (CARS 2003)*, 2003.
- WALDEMAR, PRATICK; RAUTH, MICHAEL y RAMSTAD, TOR A.: «Video Compression by Three-dimensional Motion-Compensated Subband Coding». *Norwegian Signal Processing Symposium*, 1995, pp. 227–232.
- WALLACE, GREGORY K.: «The JPEG Still Picture Compression Standard». *Communications of the ACM*, 1991.
- WANG, ALBER; XIONG, ZIXIANG; CHOU, PHILIP A. y MEHROTRA, SANJEEV: «Three-Dimensional Wavelet Coding of Video with Global Motion Compensation». *Data Compression Conference*, 1999.
- WANG, HONG; WANG, PERRY H.; WELDON, ROSS D.; ETTINGER, SCOTT M.; SAITO, HIDEKI; GIRKAR, MILIND; LIAO, STEVE S. y SHEN, JOHN P.: «Speculative Precomputation: Exploring the Use of Multithreading for Latency». *Intel Technology Journal Q1*, 2002.
- WANG, HOUNG-JYH y KUO, C.-C. JAY: «A Multi-Threshold Wavelet Coder (MTWC) for High Fidelity Image Compression». *International Conference on Image Processing*, 1997.
- WANG, JAMES ZE; WIEDERHOLD, GIO; FIRSCHEIN, OSCAR y WEI, SHA XIN: «Content-based image indexing and searchign using Daubechies'wavelets». *International Journal on Digital Libraries*, 1997.
- WEINBERGER, MARCELO J.; SEROUSSI, GADIEL y SAPIRO, GUILLERMO: «LOCO-I: A Low Complexity, Context-Based, Lossless Image Compression Algorithm». *Data Compression Conference*, 1996.
- WHALEY, R. C.; PETITET, A. y DONGARRA, J. J.: «Automated empirical optimizations of software and the ATLAS project». *Parallel Computing*, 2001, **27**(1-2), pp. 3–35.
- WHEELER, FREDERICK W. y PEARLMAN, WILLIAN A.: «Low-Memory Packetized SPIHT Image Compression». *Procceding of the 33st Asilomar Conference on Signal, Systems and Computers*, 1999.

- WOLF, JOE H.: «Programming Methods for the Pentium® III Processor's Streaming SIMD Extensions Using the VTune™ Performance Enhancement Environment». *Intel Technology Journal*, 1999.
- WOLFE, MICHAEL J.: *High Performance Compilers for Parallel Computer*. Addison-Wesley Publishing Company, 1996.
- XING, MINQING: *Wavelets for Image Compression and Reconstruction in a Distributed Environment*. Tesina o Proyecto, University of Regina, 1994.
- XIONG, ZIXIANG; WU, XIAOLIN; YUN, DAVID Y. y PEARLMAN, WILIAM A.: «Progressive Coding of Medical Volumetric Data Using Three-Dimensional Integer Wavelet Packet Transform». *IEEE Workshop on Multimedia Signal Processing*, 1998, pp. 553–558.
- YANG, LIHUA y MISRA, MANAVENDRA: «Coarse-Grained Parallel Algorithms for Multi-Dimensional Wavelet Transforms». *The Journal of Supercomputing*, 1998, **12(1-2)**, pp. 99–118.
- YANG, YAN y HEMAMI, SHEILA S.: «A Rate-Distorsion Optimized Motion Compensated Wavelet Video Coder». *Picture Coding Symposium*, 1999.
- ZILLES, C. y SOHI, G. S.: «Execution-based prediction using speculative slices». *In 28th International Symposium on Computer Architecture*, 2001, pp. 2–13.