UNIVERSIDAD POLITÉCNICA DE MADRID

Departamento de Ingeniería Telemática y Electrónica

Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación

APORTACIONES METODOLÓGICAS PARA EL DISEÑO DE DESCODIFICADORES DE VÍDEO DE ÚLTIMA GENERACIÓN SOBRE PLATAFORMAS MULTI-DSP.

TESIS DOCTORAL

Autor: Miguel Chavarrías Lapastora

Máster en Ingeniería de Sistemas y Servicios para la Sociedad de la Información

Directores:

Fernando Pescador del Oso

Doctor Ingeniero de Telecomunicación

Matías J. Garrido González

Doctor Ingeniero de Telecomunicación

Junio 2017

TÍTULO: Aportaciones metodológicas para el diseño de descodificadores de vídeo de última generación sobre plataformas Multi-DSP

AUTOR: D. Miguel Chavarrías Lapastora
DIRECTORES: D. Fernando Pescador del Oso
D. Matías Javier Garrido González

El Tribunal nombrado con fecha XX de XX de 2017 por el Mgfco. Excmo. Sr. Rector de la Universidad Politécnica de Madrid, compuesto por los doctores:

Presidente:	D.
Vocal:	D.
Vocal:	D.
Vocal:	D.
Secretario:	D.
Suplente:	D.
Suplente:	D.

realizado el acto de lectura y defensa de la tesis en la Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación de la UPM, acuerda otorgar la calificación de

_____

Madrid, a XX de XX de 2017

EL PRESIDENTE EL SECRETARIO

LOS VOCALES

A mis padres Mercedes y Pascual,

por todo su apoyo y cariño

y porque gracias a ellos hice realidad mis ilusiones.

AGRADECIMIENTOS

En primer lugar quiero agradecer a mis directores de tesis Fernando y Matías todo su apoyo, su ayuda y la guía que han sido para mí durante estos últimos años. Estoy seguro de que este trabajo que ahora cierro no habría sido algo de lo que poder sentirme orgulloso sin su confianza.

También quiero agradecer a todos los miembros de mi grupo de investigación, el GDEM, el apoyo recibido al hacerme sentir un miembro más del equipo, y en especial a César, Edu y Rubén por todo lo que de vosotros aprendo dentro y fuera del laboratorio.

A mi departamento, el DTE, donde me han abierto las puertas desde el primer día que empecé a perderme por sus pasillos. Hoy tengo la suerte de formar parte de este departamento ya como profesor. Espero seguir aprendiendo para llegar a ser un buen docente No faltan, cerca de mí, grandes ejemplos. No quiero olvidarme de agradecer su dedicación a Marisa y a Rosa, cuyo trabajo muchas veces no se ve, pero sin las cuales, estoy seguro, no saldríamos adelante.

Durante el desarrollo de mi tesis he tenido la suerte de poder realizar mi segunda estancia en el grupo IETR del INSA de Rennes. Quiero agraceder especialmente a Maxime, su atención y cercanía, pero sobre todo esas largas charlas sobre *dataflow* con las que tanto he aprendido, y también a todas las personas que allí trabajan o con las que he compartido mi vida esos meses en el extranjero: Alexandre, Khaled, Karol, Clementine y Avae, *merci beaucoup pour me montrer une région exceptionnelle!*

Quiero dar también las gracias a todos los alumnos que tuvieron el valor de hacer su proyecto fin de carrera, grado o máster conmigo: Pablo R., Fernando, Adrián, Julio, Pablo C., Estefanía, Álvaro, Rafa, Luis y Daniel. Gracias por confiar en mí y por poner vuestra parte en este trabajo que ahora termino, una parte de todo esto es vuestra.

Quiero agradecer a mis padres Mercedes y Pascual, y a mi hermano Guillermo, su apoyo incondicional desde que decicí empezar este camino. Siempre me habéis animado a dedicarme a aquello que más me gustaba, al tiempo que me ayudábais con los problemas, y a veces las dudas, que la vida te va poniendo por delante.

No me olvido de mi primo Pablo y de mis amigos María, Miguel, Óscar, Laura, Fer, Mario, Javi, Alex, Elena, y a mis "Erasmus" de Lavapiés… gracias a todos por sacarme de mi rutina y por reservarme todos esos momentos de fiesta que me he perdido durante la tesis.

Hay una persona muy especial con la que comparto mi vida casi desde el mismo día en el que me matriculé de la tesis. Gracias por aguantarme la eterna escritura de la tesis, mis explicaciones incomprensibles sobre *"de qué va mi tesis",* las frustraciones tras semanas de bloqueo… pero también por los viajes con mochila y sin ruta fija descubriendo sitios increíbles, por hacerme reír tanto los días buenos como los malos y por tu confianza en todo lo que he necesitado. Gracias por hacerme feliz y por compartir tu vida conmigo, te quiero Miguel.

ÍNDICE GENERAL

RELACIÓN DE TABLAS

RELACIÓN DE FIGURAS

RESUMEN

Esta tesis doctoral se enmarca dentro de las líneas de investigación centradas en la codificación y descodificación de vídeo digital que durante los últimos años desarrolla en la Universidad Politécnica de Madrid el Grupo de Diseño Electrónico y Microelectrónico. Dentro de estas líneas de investigación se han llevado a cabo varias tesis doctorales, donde se han propuesto distintas metodologías para la implementación de codificadores y descodificadores de vídeo sobre arquitecturas *hardware* específicas, así como para su optimización en tiempo de ejecución. Uno de los objetivos principales de estas metodologías es la reducción de los tiempos de desarrollo. Sin embargo, todas estas propuestas están centradas en arquitecturas integradas por un solo procesador. Tanto el incremento en la complejidad de los algoritmos que integran los nuevos estándares de codificación, como la utilización de procesadores multinúcleo, hacen necesaria la concepción de nuevas metodologías de diseño que aporten soluciones flexibles para la implementación de los nuevos descodificadores sobre este tipo de plataformas.

Durante la última década se ha generalizado el uso de todo tipo de arquitecturas y plataformas multinúcleo en los terminales multimedia. Los Procesadores Digitales de Señal (DSP) multinúcleo (Multi-DSP) están específicamente diseñados para mejorar el procesamiento de algoritmos como los incluidos en los descodificadores de vídeo. Sin embargo, la programación de aplicaciones para procesadores Multi-DSP no es una tarea trivial. Esto se debe principalmente a que, por una parte, el código de referencia de los descodificadores suele estar optimizado para otras arquitecturas y debe migrarse a este entorno, lo que implica largos procesos de desarrollo y, por otra, el soporte software para el desarrollo de las aplicaciones es escaso, ya que este tipo de plataformas no disponen de un Sistema Operativo (SO) que facilite la tarea del programador. Así, resulta necesario buscar soluciones metodológicas que permitan aprovechar las ventajas que las plataformas DSP ofrecen en el procesamiento de algoritmos como los que se integran en las aplicaciones multimedia manteniendo tiempos de desarrollo razonables.

El principal objetivo de esta tesis es el desarrollo de una metodología de diseño que permita implementar descodificadores de vídeo de última generación sobre plataformas Multi-DSP.

Para lograr este objetivo, en primer lugar, se ha realizado un profundo estudio sobre el estado del arte en codificación y descodificación de vídeo. En este sentido, el recientemente estandarizado *High Efficiency Video Coding* (HEVC) es el códec más novedoso de entre los existentes. De forma complementaria a los distintos estándares de codificación, el estándar *Reconfigurable Video Coding* (RVC) permite el desarrollo de los mismos en base a una descripción de alto nivel mediante unidades funcionales y flujos de datos. Dicho estándar dota de un elevado grado de flexibilidad y modularidad al proceso de diseño e implementación de descodificadores, exponiendo un elevado grado de paralelismo que facilita su ejecución en entornos multinúcleo.

Seguidamente, se ha llevado a cabo un análisis de las distintas tecnologías utilizadas en la implementación de descodificadores de vídeo y se han evaluado las distintas metodologías para el diseño de aplicaciones en entornos multinúcleo. La metodología que se propone en esta tesis tiene dos características que no poseen las metodologías que se han evaluado: que está específicamente pensada para facilitar el diseño con tecnologías DSP multinúcleo (aunque es lo suficientemente general como para ser de utilidad con otros tipos de arquitecturas) y que está muy orientada al desarrollo de codecs de vídeo (porque está basada en RVC).

Para el desarrollo de descodificadores sobre plataformas Multi-DSP ha sido necesario evaluar, seleccionar y utilizar distintas herramientas y tecnologías que dan soporte a aspectos clave como la paralelización del código, la posibilidad de optimizar alguno de los algoritmos que integran los descodificadores, o la generación automática del código fuente. Las distintas soluciones adoptadas, así como su justificación, han constituido también una parte importante de este trabajo.

A partir de la experiencia de diseño obtenida en la implementación de diferentes descodificadores en plataformas Multi-DSP se ha sintetizado una metodología de diseño, que además se ha automatizado incorporándola a *Open RVC Cal compiler* (Orcc), una herramienta de código abierto que permite la generación automática de código a partir de modelos RVC-CAL. Es más, la metodología propuesta es lo suficientemente general como para permitir la realización de implementaciones sobre plataformas multinúcleo basadas en Procesadores de Propósito General (GPPs).

ABSTRACT

This Ph.D. work is integrated into the digital video coding and decoding research lines of the Electronic and Microelectronic Research Group (GDEM), Universidad Politécnica de Madrid. Previous dissertations were mainly focused in the implementation of video codecs over specific hardware architectures, specifically in performance optimization and development time reduction. However, these proposals centered their attention on single-core architectures. Nowadays, the increase in the algorithmic complexity of new coding standards, together with the massive adoption of multicore processors, is pushing towards the conception of new design methodologies that improve designer's efficiency. Succeeding in this goal, would help in providing more flexible solutions for the implementation of new video decoders over multicore platforms.

During the last decade, the use of a large range of multicore architectures in multimedia devices has become widespread. The Multicore Digital Signal Processors (Multi-DSP) are specially designed to improve the performance of algorithms like those used on video decoders. However, programming this kind of processors has posed two main difficulties. On one side, the reference code of video codecs is typically optimized for other, more generic platforms, so it needs to be efficiently migrated, which involves long development times. Besides, software support for these platforms is reduced, as proven by the fact, among others, that no operating system is usually provided to both, help the development tasks and support the applications to take advantage of the hardware resources. Therefore, the proposal of new methodological solutions becomes necessary to take advantage of the Multi-DSP benefits while reducing the time to market.

The main objective of this thesis is the development of a design methodology that allows the implementation of state-of-the-art video decoders for Multi-DSP platforms.

First, a detailed study about the state-of-the-art on video codecs has been done. In this regard, the High Efficiency Video Coding (HEVC) standard is the newest one and it has been recently standardized. Complementary to the coding tools, the Reconfigurable Video Coding (RVC) standard allows the development of video codecs from a high-level description based on the paradigm of dataflow models of computation. The use of RVC-based solutions introduces flexibility, modularity and reusability of video codecs implementations. Also, it eases the runtime management and the parallel execution of the different actor, i.e., functional units, of the dataflow graph on the available computational resources.

Secondly, a review of video coding implementation technologies has been done too, mainly focused on DSPs and GPPs platforms. Besides, an analysis of the state-of-

the-art on multicore programming methodologies has been carried out. As a result, it turned that none of them had addressed the main target of this work: to contribute to improve the design phase for Multi-DSP technologies. Although the proposal contained herein is highly focused on the development of video codecs, as it is based on the RVC standard, it is general enough to be useful with other types of architectures and applications.

During the development of this work, it has been necessary to evaluate, select and use different tools and technologies in order to develop video decoders for Multi-DSP platforms under the analyzed conditions. These tools support key aspects such as code parallelization, algorithm optimization features (for the decoders), as well as automatic source code generation.

Finally, the proposed design methodology is based on the experience obtained during the practical implementation of different video decoders over Multi-DSP platforms. The methodology and tools developed in this work have been incorporated into Open RVC CAL Compiler (Orcc), an open source tool that allows the automatic source code generation from RVC-based models. Moreover, the proposed methodology is general enough to allow the implementation of generic multicore applications on GPP-based platforms.

PRIMERA PARTE: Introducción, objetivos y metodología

En esta primera parte de la memoria se realiza la introducción a la temática de la tesis doctoral, se resumen los objetivos de la investigación, la metodología seguida durante el desarrollo del trabajo de investigación y se define la estructura del resto de la memoria.

1. Introducción, objetivos y metodología

1.1. Introducción

La constante evolución de los estándares de codificación de vídeo y las sucesivas modificaciones que se efectúan sobre los mismos, hace que el tiempo disponible para el desarrollo de nuevas versiones de los codificadores y descodificadores sea cada vez más corto.

La codificación y la descodificación de vídeo son aplicaciones muy exigentes que requieren soportes tecnológicos con una capacidad de cálculo elevada. En la actualidad, las diferentes alternativas tecnológicas que se utilizan para implementar este tipo de aplicaciones pueden clasificarse en tres grandes grupos: las basadas en procesadores de propósito general, las basadas en arquitecturas específicas y las basadas en procesadores digitales de señal.

Desde el año 1996, el Grupo de Diseño Electrónico y Microelectrónico (GDEM) de la E.T.S.I.S. de Telecomunicación [1] (ETSIST) perteneciente a la Universidad

[1] Antigua Escuela Universitaria de Ingeniería Técnica de Telecomunicaciones (EUITT), campus sur UPM (Madrid).

Politécnica de Madrid (UPM) tiene una línea de investigación en arquitecturas *hardware* orientadas a la codificación de vídeo digital. En el marco de esta investigación se desarrollaron dos tesis doctorales [San98] [Gar04] en las que se propusieron arquitecturas específicas avanzadas para la realización de codificadores de vídeo.

Sin embargo, a partir de 2004, algunas compañías comenzaron a comercializar Procesadores Digitales de Señal (DSP) de muy altas prestaciones, con una capacidad de cómputo elevada, lo que permite implementar aplicaciones de codificación y descodificación de vídeo sobre ellos. Adicionalmente, estos procesadores incorporan una serie de periféricos (puertos de vídeo de entrada y salida, convertidores digitales-analógicos o acceso a la red local) que facilitan las interfaces propias de este tipo de aplicaciones, permitiendo implementar sistemas completos en un solo chip (System on Chip o SoC).

Las ventajas de estos procesadores respecto a las arquitecturas *hardware* específicas se encuentran en la facilidad para modificar el estándar que implementan, ya que sólo supone una sustitución del *software* que ejecuta el procesador. Esta versatilidad permite disponer de sistemas de codificación/descodificación con tiempos de desarrollo menores que los obtenidos con implementaciones basadas en arquitecturas *hardware*. Como contrapartida, tanto el precio como el consumo de los DSPs son superiores a los de los sistemas diseñados específicamente para implementar un estándar de codificación.

Otro tipo de SoC son aquellos integrados por Procesadores de Propósito General (GPPs). Si bien históricamente los GPPs han estado asociados a dispositivos de altas prestaciones en terminales fijas, los fabricantes de chips ya comercializan dispositivos con menores prestaciones que los GPPs clásicos destinados a integrarse en terminales móviles.

Dentro del proyecto PccMuTe [PCC] y empleando DSPs de última generación como soporte tecnológico, se ha desarrollado dentro del grupo de investigación una tesis doctoral [Pes11] en la que se han propuesto metodologías para la optimización del tiempo de ejecución de descodificadores de vídeo compatibles con diferentes estándares (MPEG-2, MPEG-4, H.264 y H.264/SVC) utilizando tecnología DSP.

Las metodologías propuestas en la tesis mencionada son válidas para arquitecturas con un único DSP. En la actualidad existen dos factores que impiden implementar descodificadores de vídeo funcionando en tiempo real con este tipo de arquitecturas. Por una parte los nuevos estándares de codificación como *High Efficieny Video Coding* (HEVC) [ISO15] [SOHW12] demandan una capacidad de cálculo más elevada que sus antecesores y, por otra, las resoluciones espaciales y temporales de las imágenes cada vez son mayores, lo que implica también un coste computacional mayor.

Para abordar la implementación de estos nuevos estándares y soportar elevadas resoluciones espaciales, es necesario emplear más de un procesador; de esta forma, la

carga computacional del proceso de descodificación puede repartirse entre varios procesadores. En este sentido, a comienzos de 2012 algunos de los fabricantes más representativos han comercializado dispositivos que integran varios DSPs (en adelante, procesadores Multi-DSP). Estos dispositivos poseen una elevada capacidad de cómputo para un consumo reducido lo que les hace, además, especialmente adecuados para aplicaciones que deben trabajar autónomamente empleando baterías.

El particionado de la carga computacional de un algoritmo que se ejecuta en un sistema multinúcleo es un tema que ha sido ampliamente abordado en sistemas complejos que integran Sistemas Operativos (SOs) como pueden ser Linux o Windows. En la actualidad ya existen grupos de investigación trabajando en metodologías para optimizar de forma automática la paralelización de algoritmos entre varios procesadores [YSJ+14] [PSB14], si bien hasta la fecha estos trabajos se han focalizado en procesadores de propósito general.

Sin embargo, los Sistemas Operativos en Tiempo Real (RTOS) [SYS] que emplean los procesadores multi-DSP no incorporan las facilidades proporcionadas por los SOs de altas prestaciones, por lo que es tarea del diseñador el realizar la división de forma manual. Esto implica un trabajo laborioso y poco flexible puesto que los cambios en el reparto de la carga computacional entre los procesadores requieren modificaciones significativas y un elevado coste en términos de tiempo de desarrollo.

En los últimos años, el GDEM ha orientado su investigación en la línea de arquitecturas para la codificación de vídeo hacia las metodologías para la implementación de descodificadores de vídeo de última generación con procesadores multinúcleo. En este contexto, el GDEM ha participado en los proyectos H2B2VS [H2B], MR-UHDTV [MRU] e IVME [IVME].

1.2. Objetivos de la investigación

Para flexibilizar el particionado de tareas entre los diferentes procesadores se ha investigado en metodologías que automaticen el proceso de reparto y resuelvan los problemas de sincronización y movimiento de datos entre los núcleos disponibles.

El trabajo realizado en esta tesis se centra en aprovechar las metodologías desarrolladas en otros grupos de investigación [BEP+08] [PRP+08] para procesadores de propósito general, así como la experiencia acumulada en el GDEM en la optimización de algoritmos de descodificación de vídeo para DSPs con el objetivo de proponer una metodología que reduzca el tiempo de desarrollo de algoritmos de codificación de vídeo con elevados requisitos de cálculo en plataformas Multi-DSP. Además, la metodología propuesta debe ser automatizada y validada de forma práctica utilizando tecnologías *hardware* y algoritmos de codificación que estén en el estado del arte.

De esta forma quedan definidos los tres objetivos principales de esta tesis:

1) La formalización de una metodología de trabajo que facilite el diseño de descodificadores de vídeo sobre plataformas multinúcleo basadas en tecnología DSP.

2) La creación de una herramienta que permita automatizar las tareas de diseño descritas en la metodología, con el objeto de poder generar diferentes versiones de descodificadores que corran sobre diferentes plataformas (por ejemplo, con un número mayor o menor de núcleos) en un tiempo razonable.

3) La validación de la metodología propuesta y de la herramienta generada realizando diferentes versiones de un descodificador[2] de vídeo conforme al estándar HEVC, utilizando tecnología DSP (Multi-DSP). Esta implementación debe superar con ventaja las realizaciones con un solo DSP, permitiendo incrementar la resolución espacial y/o temporal de las imágenes empleando varios núcleos de procesamiento.

Asociados a los objetivos anteriores hay una serie de objetivos secundarios que han surgido durante la realización del trabajo de investigación:

1) La integración de algoritmos de asignación dinámica de carga computacional y de tecnologías multinúcleo basadas en GPPs en la herramienta anteriormente mencionada.

2) La realización de un estudio sobre el consumo energético en un descodificador HEVC basado en tecnología Multi-DSP, teniendo en cuenta la relación entre el número de núcleos utilizados, el incremento del rendimiento y el consecuente aumento de energía consumida.

1.3. Metodología de trabajo

La metodología de trabajo que se ha empleado contempla cuatro fases:

1) Estudio del estado del arte en la implementación de algoritmos de procesamiento en plataformas multiprocesador. Se ha realizado un estudio detallado de otras contribuciones que se han publicado en la literatura científica sobre aplicaciones y herramientas que facilitan la tarea de distribución de la carga computacional entre los núcleos de procesamiento. Este estudio no se ha limitado a aplicaciones de procesamiento de vídeo, ni a sistemas basados en procesadores Multi-DSP, si no que se ha generalizado a otro tipo de aplicaciones y empleando otras plataformas para tener una visión más general respecto a la implementación de algoritmos basados en sistemas multiprocesador.

2) Implementación de un descodificador de vídeo sobre dos plataformas Multi-DSP. Con objeto de realizar los experimentos que han permitido poner de manifiesto los diferentes problemas, ensayar soluciones y, en definitiva, extraer las conclusiones

[2] La elección de un descodificador, en lugar de un codificador, para validar la metodología propuesta responde a consideraciones prácticas relacionadas tanto con el tiempo disponible como con la trayectoria investigadora previa del GDEM.

que han alimentado el desarrollo de la metodología de trabajo propuesta, se ha implementado un descodificador compatible con el estándar HEVC en una plataforma Multi-DSP. Para ello, se ha seleccionado una plataforma de desarrollo adecuada entre las existentes en el mercado ([TMS0] [TMS1], entre otras). A partir de una implementación de referencia del descodificador se ha realizado el proceso de portado a la tecnología DSP y la división de la carga computacional entre los núcleos de procesamiento, entre otras tareas necesarias.

3) Síntesis de la metodología. A partir de la experiencia obtenida durante la implementación del descodificador HEVC, se han extraído conclusiones metodológicas que han permitido definir una metodología de trabajo que permite reducir los tiempos de desarrollo de estas aplicaciones, y que ayuda en la flexibilización de la implementación en sistemas Multi-DSP con un número variable de procesadores.

4) Creación de una herramienta de diseño. Los diferentes pasos de la metodología de diseño propuesta se han integrado en una herramienta que permite su aplicación de forma automática. La herramienta ha sido verificada con el diseño de varios descodificadores HEVC sobre varias plataformas basadas en DSPs, pero también en GPPs.

Estas cuatro fases no se han desarrollado de manera completamente secuencial. El esfuerzo principal de la primera fase se realizó al comienzo de la tesis, pero ha sido necesario seguir actualizando la información recopilada con las nuevas contribuciones que se han seguido publicando durante todo el desarrollo de la misma. La segunda y la tercera fase han discurrido en paralelo y se han realimentado mutuamente durante gran parte del desarrollo de la tesis. La cuarta fase se ha llevado a cabo en la parte final de la tesis, una vez quedaron fijados los diferentes pasos de la metodología de diseño.

1.4. Estructura de la memoria

La presente memoria se ha estructurado en 5 partes, compuestas por 13 capítulos, y 6 anexos.

La primera parte se compone del capítulo primero exclusivamente; aquí se realiza la introducción y se establece el marco del trabajo de investigación desarrollado dentro de las tareas que viene realizando el GDEM de la UPM. En esta primera parte también se presentan los objetivos de la tesis doctoral.

En la segunda parte de la memoria se reúne la información relevante sobre el estado del arte en técnicas de codificación de vídeo, tecnologías de implementación y metodologías de diseño. Esta parte contiene tres capítulos:

En el capítulo 2 se introducen las últimas técnicas en codificación de vídeo digital, y se presentan los principios de los estándares HEVC y *Reconfigurable Video Coding* (RVC) [MRJ13], claves en el desarrollo de la tesis.

El capítulo 3 versa sobre las principales tecnologías utilizadas para la implementación de codificadores y descodificadores de vídeo. En este sentido se analizan las principales características de las arquitecturas de aquellas plataformas utilizadas durante la fase de desarrollo de este trabajo. También se analiza el estado del arte de las implementaciones llevadas a cabo, y publicadas en la literatura científica, por parte de otros grupos de investigación.

El capítulo 4 cierra la segunda parte de la memoria y aborda el estado del arte en las metodologías para el diseño de *software* sobre plataformas multinúcleo. En este capítulo se presenta la metodología basada en flujos de datos que ha servido como punto de partida para la consecución de los objetivos planteados en este trabajo.

En la tercera parte de la memoria se presentan las distinas tecnologías que se han utilizado durante el desarrollo de la tesis doctoral, en cuatro capítulos:

El capítulo 5 presenta el *software* del descodificador de vídeo OpenHEVC [HRD14b]. Este *software* está escrito en lenguaje de programación C y ha sido principalmente desarrollado por investigadores del INSA de Rennes [INSA].

En el capítulo 6 se presenta la Interfaz de Programación de Aplicaciones (API) OpenMP. OpenMP es un estándar *de facto* que facilita la programación paralela sobre plataformas con memoria compartida.

A lo largo del capítulo 7 se presentan las características de las plataformas utilizadas para la realización de pruebas y toma de resultados durante la fase de desarrollo.

El capítulo 8 detalla las particularidades del entorno de trabajo del compilador Orcc [ORC]. Ésta es una herramienta fundamental en el desarrollo de la tesis doctoral, ya que sobre parte de ella se implementarán las modificaciones necesarias que den soporte a la nueva metodología propuesta.

En la cuarta parte de la memoria se presenta la metodología propuesta para el diseño de descodificadores de vídeo sobre plataformas Multi-DSP. También se presentan las distintas implementaciones llevadas a cabo para validar la nueva metodología. Esta parte contiene dos capítulos:

En el capítulo 9 se detallan los distintos pasos seguidos para desarrollar la metodología propuesta. Aquí se presentan, pormenorizadamente, todos los aspectos que deben tenerse en cuenta a la hora de abordar el diseño de descodificadores de vídeo sobre plataformas Multi-DSP. También se incluyen las mejoras incluidas en la fase de diseño relativas a las técnicas de reducción de consumo energético y optimización del rendimiento mediante el *software* de OpenHEVC.

El capítulo 10 presenta el banco de pruebas utilizado para validar las distintas técnicas presentadas en el capítulo 9. Además, se detallan las particularidades de las implementaciones desarrolladas y se presentan los resultados obtenidos.

Para finalizar la memoria, en la quinta parte se presentan los resultados y las aportaciones de la tesis doctoral, en tres capítulos:

En el capítulo 11 se hace un repaso de los objetivos de la tesis y se resumen los resultados conseguidos. En el capíutlo 12 se detallan las publicaciones científicas llevadas a cabo durante la tesis así como otros méritos relacionados con el desarrollo de la misma. Por último, en el capítulo 13, se presentan las distintas propuestas de trabajos futuros a desarrollar a partir del trabajo realizado en la presente tesis doctoral.

La memoria concluye con el listado de referencias bibliográficas, los acrónimos utilizados y los anexos:

En el Anexo I se facilita la configuración establecida en el fichero *config.cfg* para la plataforma Multi-DSP. El Anexo II incluye los diagramas completos de las tres versiones de descodificadores HEVC-RVC utilizados. En el Anexo III se realiza un análisis de la utilización de módulos optimizados sobre implementaciones HEVC-RVC. El Anexo IV detalla la implementación de un descodificador HEVC en base al código de referencia del estándar (HM) sobre una plataforma DSP. El Anexo V contiene el código relativo a la gestión de interrupciones con el procesador TMS320C6678. Por último, el Anexo VI es un resumen en lengua inglesa de esta memoria.

SEGUNDA PARTE: Antecedentes

En esta parte de la memoria se presentan los antecedentes que dan lugar al trabajo desarrollado. Aquí se agrupan los conceptos tecnológicos que, en su mayoría, dan soporte a la parte teórica de la tesis doctoral.

Así, el capítulo 2 presenta los elementos principales de la codificación de vídeo digital y los últimos estándares de codificación desarrollados. En el capítulo 3 se realiza un estudio sobre el estado del arte en la implementación de descodificadores HEVC. Finalmente en el capítulo 4 se lleva a cabo un estudio sobre las distintas metodologías en el diseño de aplicaciones sobre plataformas multinúcleo.

2. Codificación de Vídeo Digital

En este capítulo se describen los conceptos más relevantes de la codificación de vídeo digital, introduciendo en primer lugar tanto los fundamentos como las técnicas que permiten la compresión de la información. Seguidamente se hace un breve repaso de los principales estándares de codificación, desde los años 80 del siglo pasado hasta la actualidad. Finalmente, se introducen los estándares HEVC y RVC que han sido utilizados en la tesis.

Así, el capítulo queda estructurado de la siguiente manera: en el apartado 2.1 se introducen los conceptos básicos y comunes a todos los estándares de codificación. También se realiza un repaso en orden cronológico de los principales estándares, destacando aquellos hitos de mayor relevancia. Seguidamente, en el apartado 2.2 se introduce con mayor detalle el estándar de codificación HEVC utilizado durante la realización de esta tesis doctoral. Finalmente, en el apartado 2.3 se presenta el estándar RVC, clave en el desarrollo de nuevos descodificadores de vídeo y que ha sido una pieza central en la investigación realizada.

2.1. Introducción a la codificación de vídeo digital

En este apartado se describen brevemente los fundamentos de la codificación de vídeo digital empleados en el desarrollo de los estándares de codificación de vídeo digital durante las últimas décadas. Gran parte de la nomenclatura utilizada en los posteriores capítulos de esta memoria se introduce en este capítulo.

2.1.1. Origen de la codificación de vídeo digital

Desde que a finales del siglo XIX comenzaran las primeras reproducciones de vídeo, hasta nuestros días, las tecnologías multimedia han experimentado grandes avances animadas por el éxito obtenido entre los consumidores de contenidos y una industria próspera y en constante expansión.

El rápido desarrollo experimentado, principalmente en los últimos años, ha permitido la diversificación de la industria multimedia, que hoy en día ofrece servicios como la televisión digital inteligente de alta definición o la reproducción bajo demanda de contenidos en terminales móviles o sistemas de videoconferencia 3D, entre otros. Buena parte del éxito y de la rápida expansión de estas tecnologías durante el último cuarto de siglo se han conseguido gracias a la digitalización de las señales de televisión. La era digital facilitó el almacenamiento y la difusión de contenidos, su edición y montaje, y abrió la puerta a una trepidante fase de desarrollo de nuevas tecnologías multimedia.

El primer formato para la digitalización y codificación de vídeo digital surgió en el año 1984 de la mano de la Unión Internacional de Telecomunicaciones (ITU) [ITU0]. En este primer estándar se define una frecuencia de muestreo fija tanto para sistemas PAL[3] como NTSC[4], al mismo tiempo que en la señal transmitida se incluyen impulsos para la sincronización de la imagen. Bajo este primer estándar el área útil del vídeo es de 625 líneas y 25 imágenes por segundo en el sistema PAL y de 525 líneas y 30 imágenes por segundo en NTSC. Además, se decide establecer una representación del vídeo basada en la utilización de tres componentes por cada *píxel*: una para la luminancia (Y) y dos para las crominancias o diferencias de color (C_R y C_B). Existen otras formas de representación de la señal de vídeo, una de las más destacables es el formato RGB, en el que se recogen tres componentes de color por pixel: rojo, verde y azul (en inglés *Red, Green, Blue*).

Tanto el régimen binario necesario para la transmisión de la señal de vídeo como la cantidad de memoria necesaria para su almacenamiento exceden las capacidades de los sistemas existentes. Por ejemplo, una secuencia de vídeo de alta definición requeriría un régimen binario de 1.24 Gbps[5]. Para almacenar únicamente 5 minutos de vídeo sería

[3] *Phase Alternating Line* (línea de fase alternada), este sistema de codificación analógica era el utilizado en la mayoría de países de Europa, Asia, África y Oceanía.
[4] *National Television System Committee* (Comité Nacional de Sistema de Televisión), sistema de codificación mayoritariamente utilizado en América y en países como Japón o Filipinas.
[5] Este régimen binario se calcula considerando una resolución de 1920x1080 píxeles, y según la fórmula: (1920x1080) píxeles/imagen * 25 imágenes/segundo * 8 bits/*píxel* * 3 componentes = 1.24 Gbps.

necesario un dispositivo capaz de albergar 373,2 Gbits, el equivalente a un disco Blu-Ray[6].

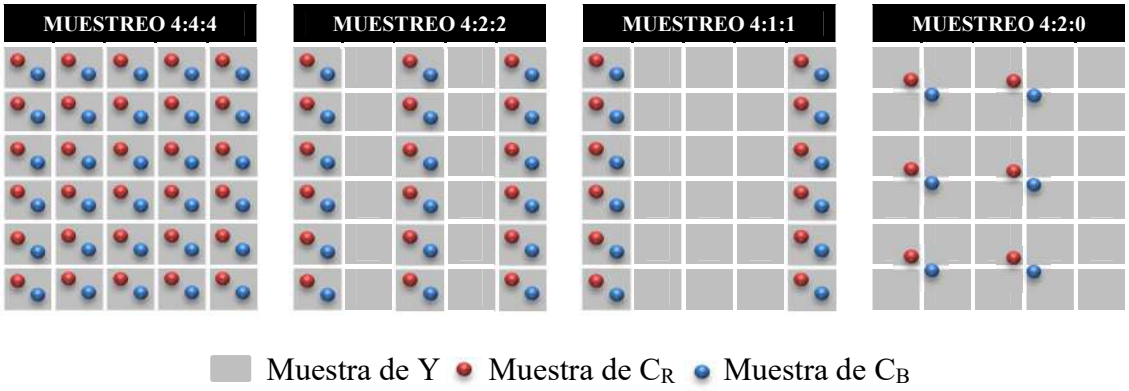
Afortunadamente, la señal de vídeo posee información redundante, y esta redundancia se puede reducir utilizando algoritmos de compresión. Normalmente, estos algoritmos permiten reducir el régimen binario de la señal de vídeo en dos órdenes de magnitud, aunque en el proceso se introducen pérdidas. Con el paso de los años se han estandarizado numerosos algoritmos de codificación, los cuales se valen de distintas técnicas de muestreo de imagen y compresión de información. Los avances en las tecnologías de compresión de vídeo han posibilitado el almacenamiento y transmisión de vídeo con resoluciones espaciales y temporales cada vez más elevadas, con regímenes binarios más reducidos y con una mayor calidad.

2.1.2. Fundamentos de la codificación de vídeo

A la hora de realizar la digitalización de una secuencia de vídeo se pueden utilizar distintos esquemas de muestreo en función de la calidad deseada, de acuerdo con las especificaciones dadas para la posterior difusión del contenido multimedia.

El muestreo de mayor calidad es aquel que en cada muestra registra, por cada píxel de la imagen, una componente de luminancia y dos de crominancias, o muestreo 4:4:4 (Figura 1-a). Sin embargo, existen otros esquemas de muestreo en los que se reduce la proporción de componentes de crominancia aprovechando que el sistema de visión humana es más sensible a los cambios en el brillo que a los matices de color. De esta forma se puede aplicar un muestreo 4:2:2 (Figura 1-b) en el que por cada dos componentes de luminancia se obtienen únicamente dos componentes de crominancia, una C_R y otra C_B. Esta proporción se ve todavía más reducida en los esquemas 4:1:1 (Figura 1-c) y 4:2:0 (Figura 1-d), en los que se registran dos muestras de crominancia C_R C_B por cada cuatro muestras de luminancia. La diferencia entre estos dos últimos esquemas radica en que en el primero de ellos el submuestreo se realiza únicamente en el eje horizontal, mientras que en el segundo se realiza tanto en el horizontal como en el vertical.

[6] Considerando que la capacidad media de un disco Blu-Ray es de 50 Gbytes.



a) Muestreo 4:4:4 b) Muestreo 4:2:2 c) Muestreo 4:1:1 d) Muestreo 4:2:0

Figura 1. Esquemas de muestro.

Ya en los primeros estándares MPEG[7]-2 ISO 13818 [ISO94] las imágenes que se codifican se dividen en áreas de píxeles, denominadas *macrobloques*. Estos macrobloques se componen de 16x16 muestras de luminancia y un número variable de muestras de crominancia, que depende del esquema de muestreo utilizado. En el estándar H.264/AVC se modificó la estructura del macrobloque, quedando éste dividido en bloques de 8x8 píxeles. Posteriormente, en el estándar HEVC (ver apartado 2.2), el tamaño del macrobloque se aumentó hasta las 64x64 muestras. La Figura 2 muestra la estructura de un macrobloque para distintos esquemas de muestreo.

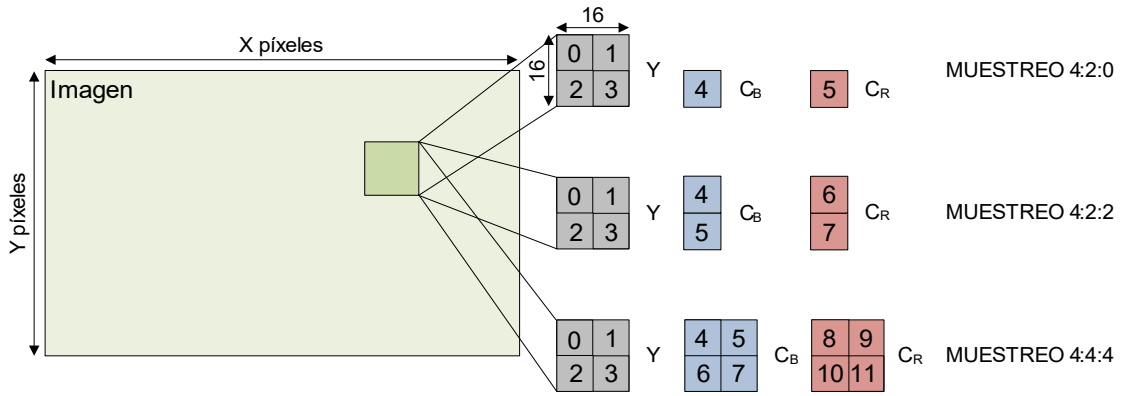


Figura 2. Estructura de un macrobloque según distintos esquemas de muestreo.

2.1.3. El esquema de codificación híbrido

Para realizar la compresión de una secuencia de vídeo se aprovechan diferentes propiedades que en mayor o menor medida permiten obtener una buena ratio de compresión con unas pérdidas de calidad aceptables. La mayoría de las bases para la compresión del vídeo se establecieron en el estándar MPEG-2, dichas bases han servido como punto de partida al resto de estándares desarrollados.

Realizando un análisis estadístico se puede observar que se dan correlaciones tanto entre los píxeles próximos entre sí dentro de cada imagen como entre grupos de

[7] *Moving Pictures Experts Group* [MPEG].

píxeles pertenecientes a diferentes imágenes. Las primeras son consecuencia de la llamada redundancia espacial de las imágenes, mientras que las segundas son consecuencia de la redundancia temporal entre imágenes.

La redundancia espacial puede reducirse mediante la implementación de un proceso de transformación al dominio de la frecuencia, donde se realiza una compactación de la información en el rango de las bajas frecuencias espaciales; posteriormente se aplican procesos de cuantificación y codificación de entropía. Todo ello permite reducir la cantidad de información de la secuencia a cambio de una reducción en su calidad que depende, básicamente, del grado de cuantificación empleado.

La reducción de la redundancia temporal se consigue mediante la aplicación de técnicas de predicción basadas en la estimación de movimiento. Para cada macrobloque de la imagen que se va a codificar se busca una predicción dentro de imágenes previamente codificadas. Los algoritmos de estimación de movimiento realizan esta búsqueda comparando el macrobloque que se va a codificar con diferentes zonas del mismo tamaño dentro de las imágenes ya codificadas utilizando un criterio de parecido, normalmente la Suma Absoluta de Diferencias. La reducción de la redundancia temporal se obtiene al codificar la diferencia entre cada macrobloque y su predicción, en lugar de codificar el macrobloque.

Adicionalmente se deben tener en cuenta las particularidades del sistema de visión humano, para reducir aquellas partes de información que dicho sistema no percibe o que percibe con poca precisión, como son las componenetes espaciales de alta frecuencia. Este hecho se conoce como redundancia de percepción.

La mayoría de los codificadores de vídeo estandarizados durante los últimos 30 años utilizan un esquema de codificación híbrido que permite reducir la redundancia espacial y la redundancia temporal de manera simultánea. La Figura 3 representa el diagrama de bloques básico de un codificador híbrido de MPEG-2. Como se observa en el diagrama, en la parte superior están los bloques o módulos más relacionados con la reducción de la redundancia espacial, y en la parte inferior aquellos más relacionados con la reducción de la redundancia temporal.

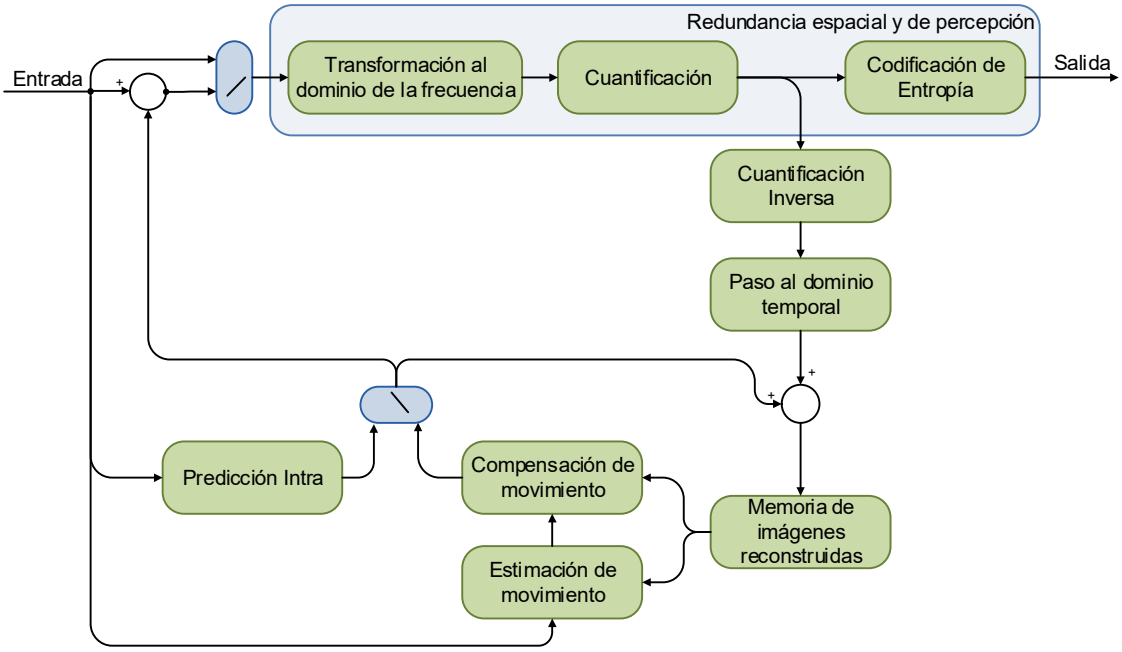


Figura 3. Diagrama de bloques de un codificador híbrido.

El codificador híbrido realiza la codificación macrobloque a macrobloque y, dentro de cada macrobloque, bloque a bloque.

En primer lugar, para cada bloque de la imagen, se realiza una transformación al dominio de las frecuencias espaciales. El resultado de esta transformación es un conjunto de coeficientes que representan las componentes de frecuencia de cada bloque. Seguidamente, estos coeficientes son cuantificados con el objetivo de reducir el rango de valores de las muestras, permitiendo así su codificación con un número de bits menor. Dependiendo del nivel de cuantificación utilizado, habitualmente se elimina una parte de las componentes asociadas a las altas frecuencias espaciales, lo que permite una reducción de la cantidad de información a costa de una pérdida moderada de calidad. Finalmente se utiliza una codificación de entropía mediante la que se asignan códigos de menor longitud a las combinaciones más probables, a este procesado se le conoce como *Context-Adaptive Variable-Length Coding* (CAVLC). Dentro de todo este proceso se producen pérdidas, fundamentalmente debidas al proceso de cuantificación; estas pérdidas no podrán ser compensadas en el proceso de descodificación.

Por otro lado, las imágenes así codificadas son sometidas a un proceso de descodificación análogo al que ocurrirá en el descodificador. El resultado de este proceso son las llamadas *imágenes reconstruidas*, que son las mismas que se han codificado, pero incluyendo las pérdidas introducidas durante el proceso de codificación. Estas imágenes son almacenadas en una memoria o *buffer* de imágenes reconstruidas. Para reducir la redundancia temporal, el codificador realiza una predicción del bloque de la imagen que debe codificar en cada momento, basada en las imágenes previamente codificadas y almacenadas en la memoria de reconstrucción. Para ello, buscará entre las imágenes ya codificadas un bloque que sirva de predicción;

una vez obtenido se codificará únicamente la diferencia entre el bloque que se va a codificar y su predicción.

El proceso de búsqueda de las predicciones se realiza mediante el módulo de estimación de movimiento. Aquellos macrobloques necesarios para el procesamiento de las predicciones pasan a la estimación de movimiento junto con las imágenes almacenadas en el *buffer*. Posteriormente, en la fase de descodificación, las imágenes pueden recuperarse sin pérdidas mediante un módulo de compensación de movimiento.

Para cada macrobloque, el codificador puede elegir entre realizar una codificación con o sin predicción. Los macrobloques que se codifican sin predicción son macrobloques intracodificados, o simplemente macrobloques *intra*, y son identificados en la trama de bits como macrobloques tipo I. Los macrobloques que se codifican con predicción son macrobloques intercodificados o macrobloques *inter*. Aquellos macrobloques *inter* para los que la predicción ha sido calculada en base a una sola imagen de entre las previamente codificadas se identifican como macrobloques tipo P (con Predicción). Si la predicción ha sido calculada en base a la información de dos imágenes previamente codificadas, entonces se identifican como macrobloques tipo B (con predicción Bidireccional).

A partir de los distintos tipos de macrobloques que integran una imagen, éstas quedan definidas como imágenes de tipo I, B o P:

- Las imágenes tipo I se componen únicamente de macrobloques tipo I. Dado que se codifican sin predicción, la compresión que puede obtenerse es pequeña. Son, por tanto, las que más tamaño ocupan y se utilizan cada cierto tiempo para evitar que los errores derivados de las pérdidas introducidas durante el proceso de codificación se vayan acumulando, con la consecuente pérdida de calidad. Adicionalmente, facilitan el acceso aleatorio a la trama de bits codificada, la recuperación frente a errores en la transmisión y la realización de búsquedas rápidas hacia adelante o hacia atrás (*fast forward* o *rewind*).

- Las imágenes tipo P deben contener al menos un macrobloque tipo P, el resto deben ser tipo I. Permiten una compresión moderada de la información a cambio de un coste de procesamiento también moderado.

- Finalmente, las imágenes tipo B deben contener al menos un macrobloque tipo B. El resto, en su caso, pueden ser de tipo P o I. Con este tipo de imágenes se consiguen las mejores ratios de compresión con un mayor coste de procesamiento.

La Figura 4 muestra una secuencia típica del ordenamiento de imágenes con los tres tipos descritos, mediante las flechas se indican las dependencias entre las imágenes. En esta secuencia la cadencia entre los diferentes tipos de imágenes es periódica, cada imagen tipo P se predice en base a la imagen P o I inmediatamente anterior, y cada imagen tipo B se predice en base a la imagen I o P inmediatamente anterior y a la

imagen I o P inmediatamente posterior. Para que un descodificador sea capaz de descodificar tal secuencia, es necesario que el codificador codifique las imágenes en un orden diferente al de presentación (visualización). En la Tabla 1 se muestra la diferencia entre el orden de presentación y el orden de codificación de las imágenes de la secuencia de la figura. Por ejemplo, para descodificar las imágenes 2 y 3, el descodificador deberá haber descodificado previamente las imágenes 1 y 4.

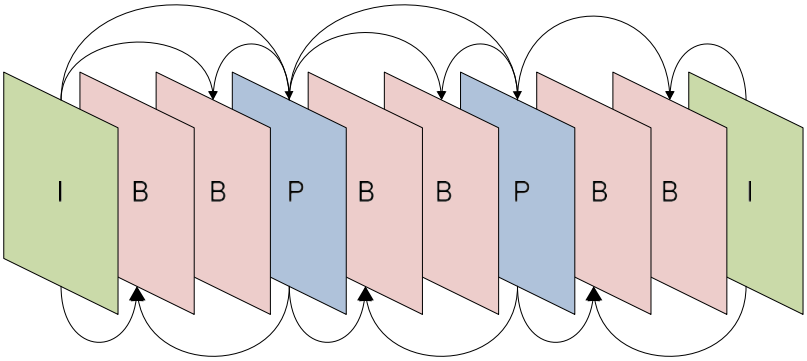


Figura 4. Representación de una secuencia con imágenes I, P y B.

Orden de presentación	1	2	3	4	5	6	7	8	9	10
	I	B	B	P	B	B	P	B	B	I
Orden de codificación	1	4	2	3	7	5	6	10	8	9
	I	P	B	B	P	B	B	I	B	B

Tabla 1. Diferencia entre los órdenes de presentación y codificación de las imágenes.

La secuencia de la Tabla 1 sigue un orden típico de codificadores de baja complejidad como MPEG-2 [ISO94]. Los codificadores más recientes como MPEG-4 Parte 10 [ISO03] o HEVC (ISO 23008-2) [ISO15] [SOHW12] generan típicamente secuencias más complejas, en las cuales, por ejemplo, la predicción de un macrobloque puede involucrar información contenida en varias imágenes codificadas además en cualquier instante previo.

2.1.4. Estándares de codificación de vídeo

Desde el desarrollo del primer estándar de codificación a principios de los años ochenta, y hasta el día de hoy, dos organismos internacionales han liderado los procesos de desarrollo y estandarización de nuevos sistemas de codificación de vídeo. Por una parte la ITU (*International Telecommunication Union)* desarrolló los estándares H.261 y H.263 entre 1984 y 1998. Mientras que la ISO (*International Organization for Standardization*) desarrolló los estándares MPEG-1, MPEG-2 y MPEG-4 en los años 1990, 1995 y 1999 respectivamente.

Sin embargo, hacia el año 2003, ambas organizaciones, la ITU y la ISO aunaron esfuerzos y desarrollaron conjuntamente el estándar H.264 en nomenclatura ITU o MPEG-4 parte 10 en nomenclatura ISO. Este es uno de los estándares más extendidos y

es ampliamente utilizado en sistemas de difusión de televisión digital o de vídeo en *streaming,* entre muchos otros.

Finalmente hacia el año 2010 se comenzó a trabajar, por parte de ambas organizaciones, en un nuevo estándar denominado H.265 (según la ITU) o HEVC (*High Efficiency Video Coding* – según la ISO). El proceso de estandarización de HEVC culminó en 2012; desde entonces HEVC se considera el sucesor natural de H.264.

La Figura 5 resume gráficamente el desarrollo de los distintos estándares de codificación durante los últimos 30 años por parte de la ITU y la ISO.

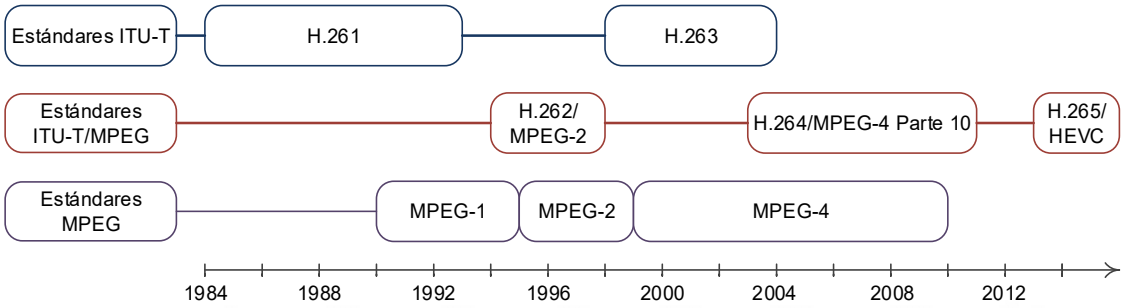


Figura 5. Representación cronológica del desarrollo de los principales estándares de codificación de vídeo.

El estándar HEVC se ha utilizado como referencia durante la realización de esta tesis doctoral, por esta razón, en el siguiente apartado se analizan sus particularidades, con especial énfasis en aquellos detalles más relevantes para este trabajo. La información relativa a los estándares de MPEG puede consultarse en [MPEGS], y en el sitio web de la ITU-T se encuentran disponibles los documentos de las distintas normas que afectan a sus estándares [ITU].

2.2. High Efficiency Video Coding

En este apartado se presenta un breve resumen del estándar de codificación de vídeo HEVC. Para ampliar esta información, en la literatura científica pueden consultarse numerosas referencias entre las que cabe destacar el documento de la ITU-T relativo al estándar [ISO15], el resumen de sus principales características publicado en [SOHW12] o el análisis de una primera implementación sobre dos plataformas basadas en GPPs [BBSF12].

2.2.1. Introducción al estándar HEVC

El estándar para la codificación de vídeo digital HEVC ha sido desarrollado conjuntamente por las organizaciones ISO/IEC [ISO] y MPEG [MPEG] mediante el consorcio *Joint Collaborative Team on Video Coding* (JCT-VC) [JCT]. Dicho consorcio surge a finales de los años 90 con el ánimo de desarrollar nuevos estándares de codificación de vídeo. Frutos de esta colaboración fueron los estándares H.262/MPEG-2 y H.264/MPEG-4 parte 10, detallados en el apartado 2.1.4. La primera versión oficial del estándar HEVC fue estandarizada en 2012 bajo la norma ISO/IEC 23008-2. HEVC

surge con el doble objetivo de mejorar la capacidad de descodificar secuencias de vídeo con mayores resoluciones espaciales, incluyendo ultra alta definición (*Ultra High Definition* – UHD[8]), al mismo tiempo que se consigue reducir el régimen binario de una secuencia de vídeo codificada con su predecesor H.264/AVC perfil alto, sin reducir con ello la calidad del vídeo. Todo esto se consigue incrementando considerablemente la complejidad de los códecs, aumentando con ello también el coste computacional de sus implementaciones.

Debido al incremento en la complejidad de los algoritmos que integran las implementaciones HEVC se hace necesaria la inclusión de herramientas que faciliten su paralelización. En este sentido, se incluyen en el estándar técnicas que permiten paralelizar el procesamiento de las imágenes a nivel de *slices,* de imágenes y por filas de CTUs (llamado *Wavefront Parallel Processing* – WPP).

2.2.1.1. Particionado de imágenes

El estándar HEVC no presenta una estructura global disruptiva con respecto a su predecesor, sino que se apoya en numerosas pequeñas mejoras efectuadas sobre los distintos bloques funcionales del codificador. Estas pequeñas mejoras consiguen una mejora global de hasta el 50% en el régimen binario necesario para transmitir una secuencia de vídeo codificado con la misma calidad respecto a su predecesor H.264.

Sin embargo, HEVC sí introduce algunos elementos novedosos con respecto a sus predecesores. Algunas de estas novedades se centran en el modelo de elementos que se utilizan a la hora de realizar el particionado de las imágenes. Antes de la llegada de HEVC el elemento utilizado para realizar el particionado durante el proceso de codificación era el macrobloque. Cada macrobloque se compone, típicamente utilizando un muestreo 4:2:0, de un total de 16x16 muestras de luminancia y 8x8 muestras de crominancia. De distinta forma, el estándar HEVC se apoya sobre un modelo más flexible basado en una serie de estructuras más complejas:

- Unidades en estructura de árbol (*Coding Tree Units* – CTU) y Bloques en estructura de árbol (*Coding Tree Blocks* – CTB). Al contrario que sus predecesores, HEVC utiliza como elemento principal de particionado el CTU, cuyo tamaño se especifica en la codificación y puede ser mayor que el del macrobloque. Cada CTU se compone un número de CTBs correspondientes a luminancias y crominancias. Además, el tamaño del CTB es variable, pudiendo ser de 16x16, 32x32 ó 64x64 muestras. HEVC también contempla un particionado menor de los CTBs mediante el modelo *quad-tree* [Sam84]. La Figura 6 representa gráficamente esta subdivisión. En la Figura 6 (a) se observa el particionado realizado sobre un CTB, y en la Figura 6 (b) la correspondiente estructura según el modelo quad-tree.

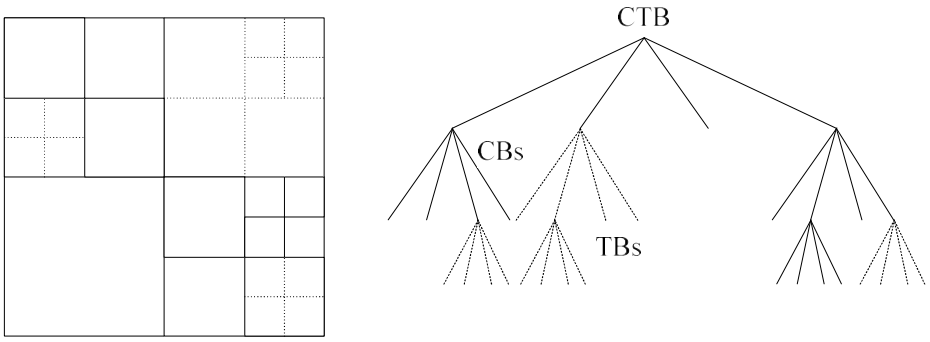
- Unidades de codificación (*Coding Units* – CUs) y bloques de codificación (*Coding Blocks* CBs). Cada CTB contiene una CU, aunque ésta puede a su vez dividirse para

[8] UHD – 4K tiene una resolución de 3840x2160 *píxeles* por imagen.

formar múltiples CUs. A su vez, una CU está formada generalmente por un CB de luminancia y dos CBs de crominancia. La relación entre una CTU y el tamaño y posición de las CBs viene dado por el índice del modelo quad-tree (ver Figura 6). Finalmente cada CU tiene asociado un sub-particionado en otros dos elementos menores: Unidades de Predicción (Prediction Units – PUs) y Unidades de Transformación (*Transform Units* – TUs).

Las decisiones a la hora de realizar predicciones tanto inter como intra se toman a nivel de CU. De esta forma, el particionado en estructuras PU viene también dado a partir del nivel de particionado de CUs. En función de las decisiones aplicadas en las predicciones, las componentes de luminancia y crominancia de los CBs pueden separarse a su vez en Bloques de Predicción (*Prediction Blocks* – PBs). Bajo el estándar HEVC el tamaño de los PBs es variable y puede tomar valores desde las 4x4 muestras hasta 64x64 muestras.

Tanto los TBs, como las TUs se integran en la estructura de árbol mostrada en la Figura 6 y tienen como referencia las CUs. Un CB de luminancia residual puede ser igual a su correspondiente TB de luminancia o bien puede dividirse en TBs de luminancia de menor tamaño. El mismo procedimiento puede aplicarse con los CBs de crominancia. Los TBs pueden tener un tamaño de 4x4, 8x8, 16x16 ó 32x32.



(a) (b)

Figura 6. Subdivisión de un CTB en CBs y TBs. Las líneas sólidas representan subdivisiones en CBs y las punteadas en TBs.

Esencialmente, los nuevos elementos introducidos en HEVC permiten la codificación de regiones homogéneas de mayor tamaño que el alcanzable con sus estándares predecesores. La Figura 7 muestra un ejemplo de la división en bloques realizada en una imagen por un codificador H.264 (imagen izquierda) y otro HEVC. Esta característica permite una reducción importante del *bitrate* necesario para codificar el vídeo [OSS+12]. Así, el conjunto de mejoras introducidas en HEVC permiten reducir hasta en un 50% el *bitrate* necesario para codificar la misma secuencia (misma resolución y factor QP) respecto del necesario con H.264.

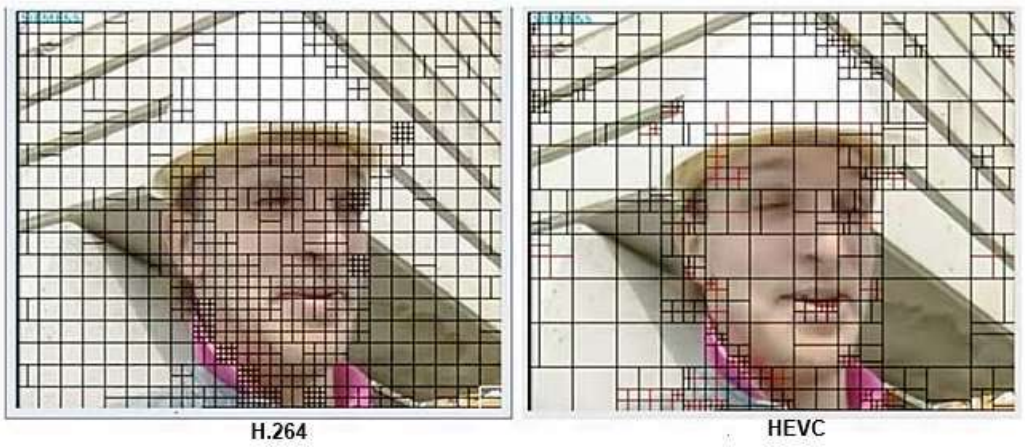


Figura 7. Ejemplo de la división de bloques realizada en una imagen por un codificador H.264 (izquierda) y por un codificador HEVC (derecha).

2.2.1.2. Sintaxis de alto nivel

Además de las modificaciones introducidas en el modelo utilizado para realizar el particionado de las imágenes, también se han llevado a cabo una serie de cambios en la sintaxis de alto nivel mediante la que se gestiona la transmisión del vídeo codificado. Los cambios introducidos tienen como objetivo mejorar la adaptación del estándar a distintos entornos de transmisión de datos para ampliar su aplicabilidad.

Los datos listos para transmitirse a la salida del codificador se organizan jerárquicamente formando una trama cuyo protocolo está bien definido en el estándar. El codificador incluye en la trama la información necesaria para la sincronización y la descodificación de las imágenes codificadas encapsuladas en la trama. Cada paquete de la trama viene precedido por una cabecera en la que se incluye la información necesaria para una correcta interpretación del subsiguiente paquete de datos. En el lado del descodificador se procesa en primer lugar cada cabecera antes de comenzar la descodificación de cada imagen. La Figura 8 representa gráficamente una trama a la salida del codificador; como se puede ver, en primer lugar se transmiten varios paquetes relativos a la información necesaria para proceder a la descodificación, seguidamente se envían secuencialmente las imágenes codificadas. A continuación se describen las cabeceras y tipos de paquetes más relevantes que se pueden encontrar en una trama codificada con el estándar HEVC.

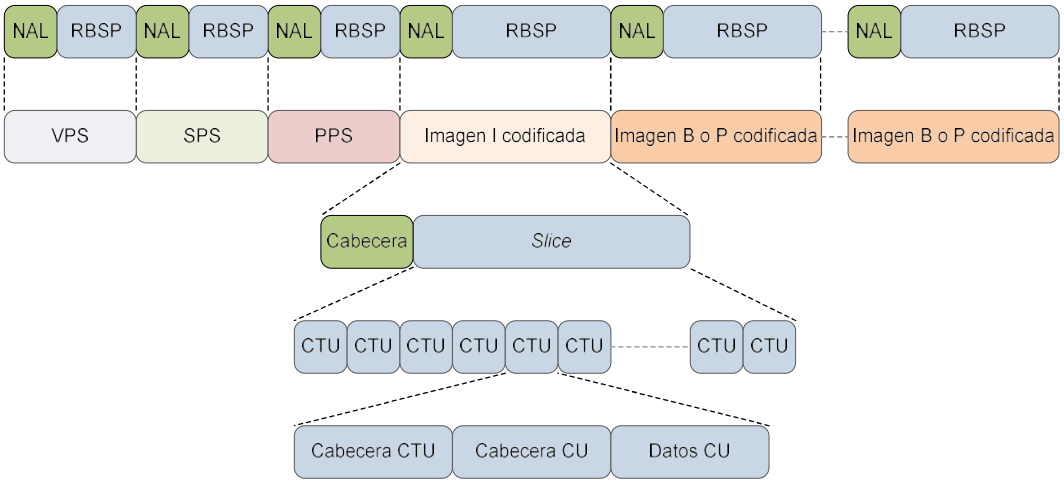


Figura 8. Representación gráfica de los niveles jerárquicos de una trama de vídeo codificado bajo el estándar HEVC.

- Cabecera NAL (*Network Abstraction Layer* – Capa de abstracción de red): todo paquete transmitido viene precedido de la cabecera NAL de dos *bytes*, mediante la que se informa del propósito del paquete de datos transmitido a continuación. Se han definido distintos tipos de cabeceras NAL en función de la información encapsulada, la Tabla 2 lista todos ellos. Como se puede observar, existen numerosos tipos de cabeceras que permiten una rápida identificación de la información que se transmite por parte del codificador. El detalle de cada uno de los tipos de cabecera NAL se puede consultar en [ITU15].

Tipo	Significado	Clase	Tipo	Significado	Clase
0, 1	*Slice* ordinario de imagen	VCL	32	Paquete VPS	No VCL
2, 3	*Slice* de imagen TSA	VCL	33	Paquete SPS	No VCL
4, 5	*Slice* de imagen STSA	VCL	34	Paquete PPS	No VCL
6, 7	*Slice* de imagen RADL	VCL	35	Delimitador de acceso a unidad	No VCL
8, 9	*Slice* de imagen RASL	VCL	36	Fin de secuencia	No VCL
10~15	Reservado para futuras ampliaciones	VCL	37	Fin del *bitstream*	No VCL
16~18	*Slice* de imagen BLA	VCL	38	Datos de relleno	No VCL
19, 20	*Slice* de imagen IDR	VCL	39, 40	Mensaje SEI	No VCL
21	*Slice* de imagen CRA	VCL	41~47	Reservado para futuras ampliaciones	No VCL
22~31	Reservado para futuras ampliaciones	VCL	48~63	No especificado (disponible para uso del sistema)	No VCL

Tabla 2. Listado de los distintos tipos de cabeceras NAL en el estándar HEVC.

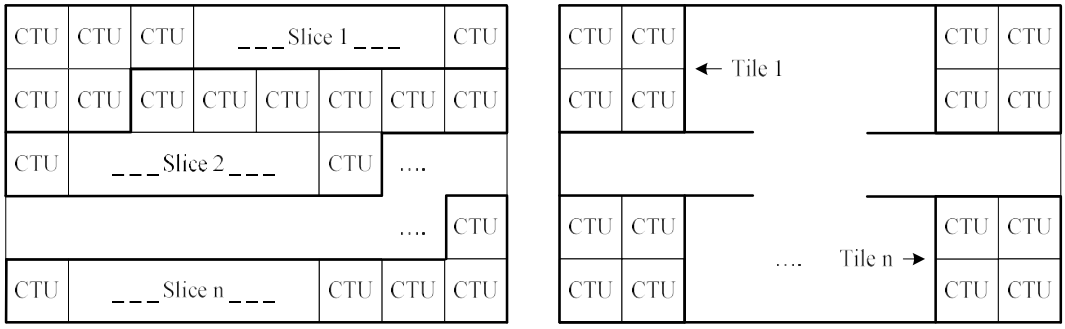
- El primer paquete que se transmite contiene información común a toda la trama de vídeo codificado, este paquete es una extensión del mismo utilizado en el estándar H.264 y se denomina VPS (*Video Parameter Set* – conjunto de parámetros del vídeo).

- Tras el VPS se transmite un paquete SPS (*Sequence Parameter Set* – conjunto de parámetros de secuencia), donde se incluye información común a toda la secuencia de vídeo, como el perfil y el nivel utilizados en la codificación.

- Finalmente, y antes de comenzar el envío de las imágenes codificadas, se adjunta un paquete PPS (*Picture Parameter Set* – conjunto de parámetros de imagen). Este paquete se utiliza para transmitir información relativa a la codificación, similar al paquete SPS, pero de una o varias imágenes individuales.

- *Slice* (rebanada o porción): una *slice* es el conjunto de datos codificados correspondientes a una parte, o a la totalidad, de una imagen. Cada *slice* puede ser descodificada de forma independiente de otra en términos de codificación de entropía, predicción o reconstrucción de la señal. De esta forma los errores producidos en descodificación en caso de pérdidas en la transmisión se ven reducidos. Cada *slice* está compuesta de un número variable de CTUs. La Figura 9 (a) muestra un ejemplo de cómo una imagen puede subdividirse en *slices*.

- *Tiles* (mosaicos): bajo el estándar HEVC se ha definido un subnivel adicional cuyo propósito es facilitar el procesamiento en paralelo de distintas regiones de imágenes. Los *tiles* se corresponden a regiones rectangulares de una imagen y pueden ser descodificados de forma independiente, además pueden compartir la misma información de una cabecera del *slice* en el que se encuentren. La Figura 9 (b) muestra un ejemplo de subdivisión en *tiles*.



(a) (b)

Figura 9. Subdivisión de una imagen en *slices* (a) y en *tiles* (b).

2.2.1.3. Perfiles y niveles de HEVC

Como es habitual en los estándares de codificación de vídeo MPEG, en HEVC se han definido una serie de perfiles y niveles que ofrecen distintos grados de funcionalidad y calidad. Éstos permiten establecer diversos grados de operatividad de cara a la implementación del estándar en distintos campos de aplicación. Los perfiles representan subconjuntos de la sintaxis de la trama de bits de salida del codificador, mientras que los niveles especifican, dentro de cada perfil, un conjunto de restricciones a aplicar sobre los distintos parámetros de esa trama.

A continuación se detallan todos aquellos perfiles que han sido incluidos de forma oficial en el estándar; aunque para la realización de este trabajo se han utilizado distintas versiones de descodificadores HEVC (capítulo 10), el perfil seleccionado ha sido en todos los casos *Main*, ya que su uso es el más extendido y maximiza las oportunidades de comparación de los resultados de esta investigación con los de otros trabajos. La Tabla 3 muestra las principales características de los distintos perfiles existentes para el estándar HEVC.

Perfiles / Características	Main	Main 10	Main 12	Main 4:2:2 10	Main 4:2:2 12	Main 4:4:4	Main 4:4:4 10	Main 4:4:4 12	Main 4:4:4 16 Intra
Profundidad de color	8	8~10	8~12	8~10	8~12	8	8~10	8~12	8~16
Formatos de muestreo	4:2:0	4:2:0	4:2:0	4:2:0/ 4:2:2	4:2:0/ 4:2:2	4:2:0/ 4:2:2/ 4:4:4	4:2:0/ 4:2:2/ 4:4:4	4:2:0/ 4:2:2/ 4:4:4	4:2:0/ 4:2:2/ 4:4:4
4:0:0 (Monocromo)	No	No	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Predicción de alta precisión ponderada	No	No	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Predicción de componentes cruzados	No	No	No	No	No	Sí	Sí	Sí	Sí
Desactivación de suavizado intra	No	No	No	No	No	Sí	Sí	Sí	Sí
Adaptación Rice persistente	No	No	No	No	No	Sí	Sí	Sí	Sí
RPDCM implícita / explícita	No	No	No	No	No	Sí	Sí	Sí	Sí
Transformación de tamaños de bloque mayores de 4x4	No	No	No	No	No	Sí	Sí	Sí	Sí
Transformar salto de contexto / rotación	No	No	No	No	No	Sí	Sí	Sí	Sí
Procesamiento de precisión extendida	No	No	No	No	No	No	No	No	Sí

Tabla 3. Listado de perfiles incluidos en el estándar HEVC.

La Tabla 4 muestra los valores límite para determinados parámetros que se han definido en los diferentes niveles dentro del perfil *Main*. Así mismo el perfil *Main* incluye dos clasificaciones, *Main Tier* (clasificación principal) y *High Tier* (clasificación alta), éstas se diferencian únicamente en el régimen binario y la capacidad del *buffer* de imágenes en el descodificador.

Nivel	Número máximo de muestras Y	Ratio máximo de muestras Y (muestras/segundo)	*Bitrate* máximo para el nivel *Main*	*Bitrate* máximo para el nivel *High*	Ratio de compresión mínimo
1	36.864	552.960	128	-	2
2	122.880	3.686.400	1.500	-	2
3	552.960	16.588.800	3.000	-	2
4	2.228.224	66.846.720	12.000	30.000	4
5	8.912.896	267.386.880	25.000	100.000	6
6	35.651.584	1.069.547.520	60.000	240.000	8

Tabla 4. Valores límite para los principales niveles del perfil *Main*.

2.2.2. Modelo de un descodificador HEVC

En este apartado se describen, una vez introducidas las principales características del estándar de codificación HEVC, los módulos que integran un descodificador de vídeo HEVC genérico. La información relativa a la funcionalidad de cada etapa de procesado, así como la implementación de cada una de ellas, es crucial a la hora de establecer técnicas que permitan explotar el procesamiento paralelo de las mismas y tomar decisiones sobre la optimización del código relativo a cada sección. El diagrama de la Figura 10 muestra los distintos bloques que deben integrar un descodificador HEVC. Seguidamente se resume la funcionalidad de cada uno de los bloques.

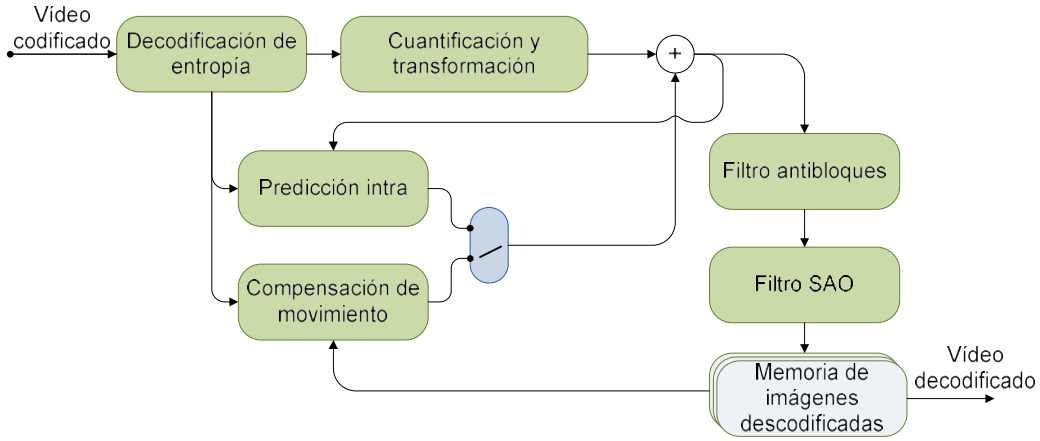


Figura 10. Diagrama de bloques simplificado de un descodificador HEVC.

- Descodificación de entropía: En el estándar HEVC el método utilizado para la codificación/descodificación de entropía es el llamado *Context-based Adaptive Binary Arithmetic Coding* (CABAC) [MSW03]. A partir de la trama de bits codificada, el descodificador de entropía extrae la información sobre el tipo de codificación empleada (inter/intra), los coeficientes de cada CU (transformados y cuantificados) y los vectores de movimiento (en su caso).

- Cuantificación y transformada inversa: una vez se han extraído los datos del vídeo codificado mediante la descodificación de entropía se realiza la decuantificación de los coeficientes transmitidos, dichos coeficientes deben ser entonces transformados al dominio temporal mediante la Transformada Discreta del Coseno Inversa (IDCT), obteniendo así los bloques de píxeles que conforman la imagen.

- Reconstrucción de imágenes: las CTUs se reconstruyen mediante la suma de las predicciones inter/intra y las CTUs salientes del proceso de cuantificación y transformación.

- Predicción Intra: en el caso de que una CU se haya codificado con predicción intra-*frame* se procede a calcular su predicción a partir de la información de los PBs vecinos. Además, en este caso los bloques intra descodificados pasarán directamente a la memoria de imágenes.

- Predicción Inter: en este caso el bloque de compensación de movimiento obtiene la predicción a partir de los vectores de movimiento y de las imágenes previamente almacenadas en la memoria de imágenes. Este procesado se realiza a nivel de PUs y puede utilizarse una compensación de movimiento con precisión de hasta un cuarto de píxel.

- Filtrado antibloques y filtrado SAO: tras la obtención de la imagen se aplica un primer filtrado antibloques (DBF) [SOHW12] consistente en suavizar los flancos existentes entre los distintas PUs y TUs. Este filtrado se realiza a nivel de CU y se aplica en bloques de 8x8 píxeles. Seguidamente, se aplica un segundo filtrado (SAO, *Sample Adaptative Offset*) [SOHW12]. Este filtrado no está acotado a los bordes de los bloques que integran las imágenes, y consiste en añadir un valor de base (*offset*) a algunos píxeles ya reconstruidos. Los valores a añadir se eligen en función de una tabla de valores transmita y generada por el codificador. Ambos filtros se aplican en el bucle correspondiente a la predicción de imágenes inter, y sus efectos de mejora sobre la imagen descodificada se incrementan al utilizarse de forma conjunta.

- Memoria de imágenes descodificadas: las imágenes descodificadas se almacenan en memoria con un doble objetivo: disponer de las imágenes utilizadas durante la predicción inter ya que son necesarias para su correcta descodificación, así como para poder reordenar correctamente las imágenes en su orden natural ya descodificadas como se explicó en el apartado 2.1.3.

2.3. MPEG Reconfigurable Video Coding (RVC)

En este apartado se presentan las principales características del estándar de Codificación de Video Reconfigurable (RVC, de sus siglas en inglés *Reconfigurable Video Coding*). El estándar ha sido desarrollado por el grupo MPEG y actualmente se encuentra estandarizado por la ISO. El estándar RVC ofrece una solución al desarrollo clásico y monolítico de descodificadores de vídeo. Esto es posible mediante la combinación de un nuevo lenguaje para desarrollar aplicaciones y una serie de bibliotecas que incluyen los algoritmos de codificación utilizados en los códecs de MPEG.

2.3.1. Introducción al estándar RVC

El desarrollo del estándar RVC comenzó en el año 2005 con el ánimo de ofrecer nuevas soluciones en la implementación de códecs de vídeo digital. Dichos códecs se sirven de algoritmos cuya complejidad ha ido creciendo a la vez que se han ido consiguiendo mayores tasas de compresión. Además, el desarrollo de nuevos dispositivos multimedia, como *tablets* o *smartphones,* y la implantación de la televisión digital, requieren de nuevas implementaciones más versátiles. La necesidad de poder reutilizar elementos ya desarrollados, y de integrar en un solo descodificador diversos códecs, permitiendo que el mismo se adapte dinámicamente al flujo de datos entrante, hace de RVC el estándar idóneo para afrontar los retos del momento.

Con el desarrollo de RVC, MPEG realizó un trabajo de reorganización de los estándares ya existentes, de esta forma MPEG-1, 2 y 4 se integraron en MPEG-A (formatos de aplicaciones multimedia), MPEG-B (Sistemas tecnológicos), MPEG-C (tecnologías de vídeo) y MPEG-H (para la difusión en medios heterogéneos).

El nuevo *framework* MPEG RVC fue estandarizado como MPEG-B parte 4 y MPEG-C parte 4 bajo las normas ISO/IEC 23001-4 [ISO09] e ISO/IEC 23002-4 [ISO10] respectivamente, con el objetivo de reemplazar las referencias *software* de los descodificadores, escritas en lenguaje C o HDL, por especificaciones de la descodificación de vídeo a nivel de bibliotecas de componentes. La primera de estas normas ofrece un nuevo lenguaje para la descripción de códecs de vídeo; la segunda, una biblioteca de herramientas utilizada por los algoritmos de descodificación de vídeo en los estándares de MPEG. De esta forma, y bajo el estándar RVC, las antiguas referencias *software* son sustituidas por representaciones Abstractas de Modelos de Descodificadores (*Abstract Decoder Models* – ADMs). Los detalles que ilustran este modelo se describen en siguiente apartado y en la Figura 11.

Uno de los principales objetivos de MPEG RVC es el desarrollo de descodificadores de vídeo digital a partir de una serie de especificaciones de alto nivel basadas en modelos de flujos de datos (*dataflow*). De esta forma, un ADM representa el modelo de un descodificador como un diagrama de flujos de datos. Este tipo de representaciones ofrecen una serie de ventajas claves en el estándar RVC, las cuales se enumeran a continuación:

- Un lenguaje de programación abstracto, que permite a los desarrolladores abstraerse de los detalles de bajo nivel de la arquitectura y poner el énfasis en la mejora y optimización de los algoritmos de compresión.

- Un paralelismo escalable, gracias a que los nuevos métodos de programación se basan en modelos de flujos de datos, lo cual facilita un mayor paralelismo de los programas desarrollados.

- La reutilización de módulos y algoritmos es más fácil, ya que existen numerosas similitudes entre los módulos utilizados en distintas implementaciones. Además, el *framework* MPEG RVC facilita el desarrollo de algoritmos que puedan reutilizarse en futuros estándares.

2.3.2. Descripción de descodificadores de vídeo con MPEG RVC

En esta subsección se introducen los principales conceptos necesarios para entender el funcionamiento y la implementación de los módulos que integran una aplicación basada en MPEG RVC. La mayoría de estos conceptos son clave para entender el desarrollo de esta tesis doctoral.

Hoy en día los estándares MPEG-B y MPEG-C, sobre los que se apoya MPEG RVC, siguen evolucionando con el objetivo de incluir nuevos algoritmos sobre los que

se puedan desarrollar futuros estándares de descodificación de vídeo. Como se introducía en la subsección anterior, los ADMs pueden representar las especificaciones de un descodificador de vídeo basado en alguno de los estándares existentes de MPEG.

Desde un punto de vista técnico, MPEG RVC queda definido mediante una biblioteca de herramientas de codificación, una descripción sintáctica del descodificador y una descripción del flujo de datos entre las unidades funcionales que integran las aplicaciones. Cada uno de estos elementos queda integrado en el ADM como una Biblioteca de Herramientas de Vídeo (*Video Tool Library* – VTL), un Lenguaje Funcional de unidad de Red (*Functional unit Network Language* – FNL) [ISO11] y un Lenguaje para la Descripción Sintáctica del *Bitstream* (*Bitstream Syntax Description Language* – BSDL) respectivamente. La Figura 11 ilustra gráficamente el modelo basado en ADMs.

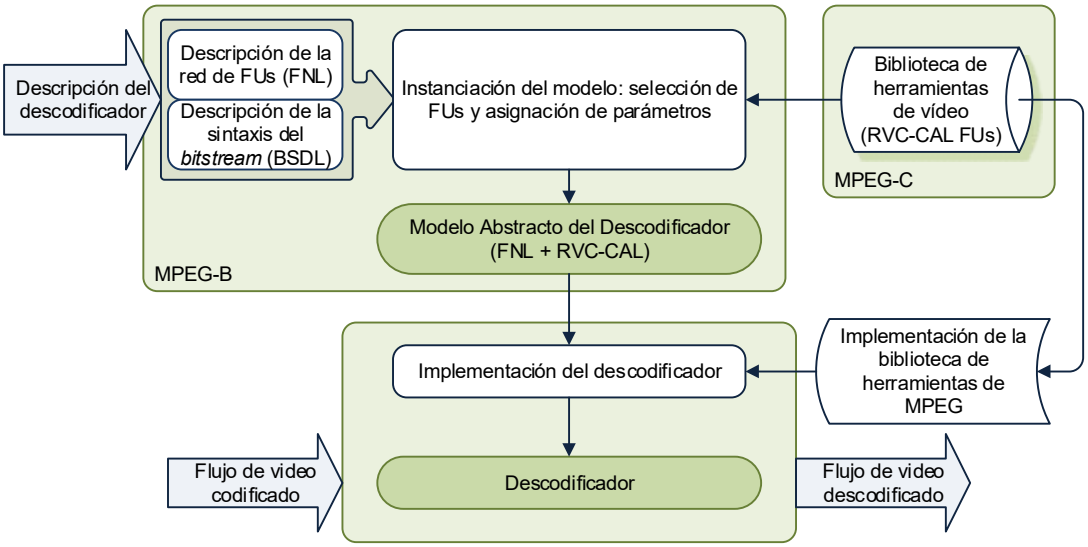


Figura 11. Modelo de ADMs utilizado en el estándar RVC.

Los algoritmos que implementan la funcionalidad se encapsulan en entidades independientes llamadas Unidades Funcionales (*Functional Units* – FUs), estas entidades conforman una serie de herramientas de codificación proporcionadas por la VTL. La funcionalidad de cada FU es implementada a su vez por otra entidad denominada *actor*. De esta forma una FU puede contener a su vez otras FUs, sin embargo, en el nivel más bajo de la jerarquía cada FU debe contener, al menos, un actor. La programación de los actores se realiza en el lenguaje de referencia RVC-CAL Actor Language (RVC-CAL) [CAL] [BEJ+09]. Los conceptos básicos, tanto de la programación interna, como del funcionamiento de los actores, se describen someramente en la subsección siguiente.

El FNL permite realizar la interconexión entre las FUs. En el estándar RVC, una aplicación queda definida mediante una serie de instancias de FUs o actores, representadas gráficamente mediante cajas, y la interconexión de las mismas mediante flechas. Las flechas representan el flujo de datos entre los actores y se conectan a éstos mediante puertos de entrada/salida, la implementación de los flujos de datos se realiza

mediante colas de datos tipo FIFO (*First In First Out*, primero en entrar, primero en salir). Existen tres tipos de conexiones definidas en FNL:

- *Input* (entrada): entre un puerto de entrada de la red y una instancia.
- *Output* (salida): entre la salida de una instancia y un puerto de salida de la red.
- *Connection* (conexión): entre la salida de una instancia y la entrada de otra instancia.

La Tabla 5 ilustra los distintos tipos de elementos que pueden darse en este tipo de modelos utilizando FNL a partir del diagrama de la Figura 12.

Figura 12. Diagrama básico de un modelo basado en FNL.

Instancias	<Instance id=" FU_1 "> <Class name =" Algo_Ejemplo_1 "/> </ Instance > <Instance id=" FU_2 "> <Class name =" Algo_Ejemplo_2 "/> </ Instance >
Connections	<Connection src= » FU_1 « src –port = »B » dst= » FU_2 « dst –port = »D »/> <Connection src= » FU_1 « src –port = »C » dst= » FU_2 « dst –port = »E »/>
Inputs	<Input src=" FU_1 " src –port ="A"/>
Outputs	<Output src= » FU_2 « src –port = »F »/>

Tabla 5. Elementos existentes en el diagrama FNL de la Figura 12.

Finalmente se incluyen en el *framework* las herramientas necesarias para realizar el proceso de análisis y traducción sintáctica de *bitstreams* mediante RVC-BSDL. El BSDL es capaz de realizar la traducción sintáctica de cualquier *bitstream* compatible con el estándar y está definido en la norma ISO/IEC 23001-5 [ISO08].

2.3.3. RVC-CAL Actor Language, conceptos básicos

Con el objetivo de hacer esta memoria lo más autocontenida posible, este apartado presenta únicamente aquellos conceptos principales sobre el lenguaje de programación RVC-CAL Actor Language. Para tener un conocimiento más detallado de la especificación CAL Actor Language se recomienda acudir al informe realizado por la Universidad de California en Berkeley donde fue desarrollado [CAL].

RVC-CAL Actor Language es un lenguaje de programación de alto nivel utilizado en la descripción del comportamiento de los actores que integran las aplicaciones basadas en el estándar MPEG RVC. Los actores se componen de interfaces, puertos de entrada/salida, variables de estado y otros parámetros. A lo largo de esta subsección se van a utilizar pequeños extractos de código para que sirvan de ejemplo.

El primer elemento de un actor es su cabecera, en la cual se deben declarar los puertos de entrada/salida necesarios. La Figura 13 muestra la parte de código relativa a la cabecera, los puertos de entrada (Y, U y V) y de salida (YUV), las acciones y la FSM del actor MergeYUV integrado en un descodificador HEVC-RVC.

```
Package org.ietr.mpegh.part2.common;

actor MergeYUV () //Entradas al actor
      int(size=8) Y,
      int(size=8) U,
      int(size=8) V

      ==>  //Salidas del actor
      int(size=8) YUV :

      // Action, select the Y blocks
            Read_Y: action Y:[ y ] repeat 4096 ==> YUV:[ y ] repeat 4096
      end
      (...) //Resto de acciones (Read_U y Read_V)

      fsm Y :
            Y ( Read_Y ) ==> U;
            U ( Read_U ) ==> V;
            V ( Read_V ) ==> Y;
      end
end
```

Figura 13. Cabecera y cuerpo en lenguaje RVC-CAL del actor MergeYUV del proyecto *Research*.

El procesamiento de un actor se describe mediante una secuencia de pasos que se ejecutan de forma atómica, de manera similar a como lo hacen las funciones escritas en otros lenguajes de programación, como el lenguaje C. Estas secuencias de pasos se denominan acciones. La funcionalidad de cada actor debe describirse como parte del código de cada acción. A la ejecución de una acción se le denomina disparo (*firing*); en cada ejecución el actor consume elementos (*tokens*) disponibles en sus entradas, produce *tokens* en los puertos de salida y modifica las variables de estado. Este proceso ocurre para cualquier actor y es el mismo en cualquier proceso de disparo. En el ejemplo de la Figura 13 se muestra la acción *Read_Y* junto con el código de la misma y sus acciones sobre los puertos de E/S. La descripción del actor concluye con la especificación de la FSM, en este caso tras ejecutar la acción Read_Y se continuará con Read_U, y seguidamente Read_V, para volver entonces al punto de partida.

Un actor puede estar compuesto por una o varias acciones. Pueden existir situaciones no determinísticas si las condiciones de ejecución de cada una de las acciones no han sido especificadas para cualquier situación posible. En la Figura 14 se ejemplifica un caso no determinístico básico, ya que ambas acciones pueden ejecutarse al no haberse descrito condiciones de ejecución; en este caso la acción que será finalmente ejecutada es indeterminada.

```
action Input1: [x] ➔ [x] end
action Input2: [x] ➔ [x] end
```

Figura 14. Ejemplo de sección de código no determinística en RVC-CAL.

La ejecución de una acción se produce cuando hay datos disponibles en las entradas, espacio suficiente disponible en los puertos de salida afectados y las condiciones de *guarda* se cumplen, si existen. Las condiciones de guarda pueden referirse al valor de los datos de entrada, al estado interno del actor o a ambos. La Figura 15 ilustra un ejemplo: mientras el valor de la entrada 'sel' sea TRUE el dato en 'A' pasa al puerto de salida, en caso contrario se selecciona el dato presente en el puerto 'B'. Dependiendo del propio estado del actor, las condiciones de guarda pueden utilizarse para decidir las próximas acciones a ejecutar. Toda la gestión de la ejecución de las acciones se realiza mediante una Máquina de Estados Finita (*Finite State Machine* – FSM), donde se pueden tener en cuenta los valores de los *tokens* de entrada, las condiciones de guarda y las prioridades.

```
  Read_CodingUnit_end_sendQp.blk4x4:action QpData:[length, qp_y]➔Qp:[qp_y]
guard //Condición de guarda para que se pueda ejecutar la acción precedente
  length = 0
end
```

Figura 15. Ejemplo de código en RVC-CAL de una acción del actor QpGen con condición de guarda.

2.3.4. Funcionamiento interno de los actores

El código fuente que implementa cada uno de los actores es el resultado de las transformaciones realizadas sobre el código CAL de los mismos. Además, se genera el código relativo a la gestión de entrada/salida de datos mediante las FIFOs y la FSM que controla el comportamiento interno de cada actor.

Todos los actores disponen además de dos funciones: la función de inicialización, que establece el estado inicial de la FSM y realiza una primera lectura de *tokens* en las FIFOs de entrada del actor, y la función del planificador, la cual es llamada secuencialmente por el hilo de ejecución y que controla la ejecución de las funciones internas del actor mediante la FSM. La Figura 16 esquematiza el funcionamiento de los actores, cuando un actor finaliza su ejecución (no hay más *tokens* en sus FIFOs de entrada) el planificador de acciones llamará a ejecución al siguiente actor en ese hilo de ejecución (estos conceptos se explican en detalle en el apartado 4.5.2).

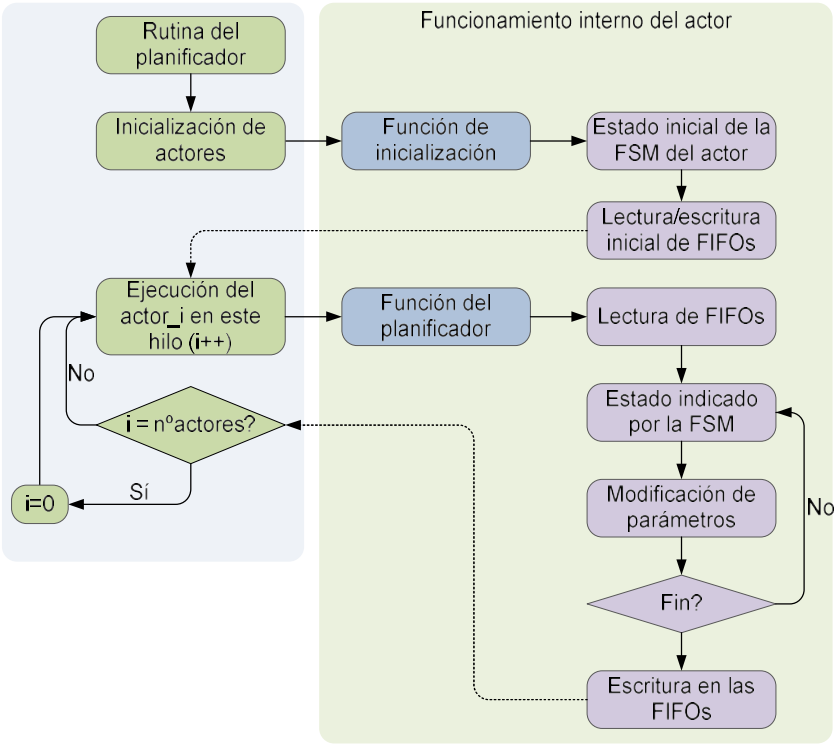


Figura 16. Diagrama del funcionamiento interno de los actores.

2.3.5. Implementaciones de descodificadores de vídeo basadas en MPEG RVC

Desde que en el año 2002 comenzara el desarrollo de MPEG RVC, diferentes grupos de investigación han llevado a cabo numerosas aplicaciones, tanto para la descodificación de vídeo digital, como dentro de otros campos de estudio. Algunos de los grupos más destacados son: IETR (*Institut d'Électronique et de Télécommunications de Rennes*) [IETR] e IRISA (*Institut de Recherche en Informatique et Systèmes Aléatoires*) [IRISA], ambos dentro de la red de centros de investigación del INSA (*Institut National des Sciences Appliquées*) de Rennes (Francia), y l'EPFL (*École Polytechnique Fédérale de Lausanne*) [EPFL] en Suiza. Cabe destacar que el GDEM, donde se ha desarrollado esta tesis doctoral, ha realizado varias publicaciones relacionadas con MPEG RVC [JRR+13] [RCS+13] [JRW+13]. Existe una estrecha relación entre los cuatro grupos de investigación mencionados, y en varias ocasiones se han realizado intercambios de personal docente e investigador con el fin de fomentar la colaboración internacional y la superación conjunta de algunos de los retos planteados por MPEG RVC.

Tanto para el desarrollo de esta tesis doctoral, como en la mayoría de proyectos impulsados por los distintos grupos de investigación anteriormente mencionados, se ha utilizado un repositorio web [ORC] donde se encuentran disponibles numerosas aplicaciones basadas en RVC. En el repositorio se pueden encontrar los siguientes descodificadores de vídeo digital basados en MPEG RVC:

- Descodificador MPEG-4 parte 2 SP.
- Descodificador MPEG-4 parte 10 (AVC o H.264), perfiles PHP y CBP.
- Descodificador MPEG-H parte 2 (HEVC o H.265).

Durante la elaboración del Trabajo Fin de Máster del doctorando en 2012 se realizó un estudio detallado de los descodificadores MPEG-4 parte 2 y MPEG-4 parte 10. En este trabajo se analizó el rendimiento de estas implementaciones, tanto sobre plataformas mono-núcleo como multinúcleo en arquitecturas tipo PC y ARM. La memoria está disponible en [Cha12].

El descodificador HEVC ha sido el utilizado como aplicación de referencia para el desarrollo de esta tesis doctoral. Cabe destacar que hay desarrolladas distintas versiones de este descodificador. En el apartado 10.1 se describen pormenorizadamente cada una de estas versiones y su importancia en el desarrollo de este trabajo.

- Descodificador HEVC-RVC de referencia (REF)*:* primera versión estable que se desarrolló. En esta primera versión el grado de paralelismo alcanzable está muy limitado, sin embargo se incluyen ya las etapas de filtrado SAO y antibloques, y está preparada para resoluciones de hasta 4K.
- Descodificador HEVC-RVC YUV: versión con el procesado dividido en FUs independientes de las componentes Y, U y V. Esta versión ofrece un mayor grado de paralelismo que el alcanzable con la versión de referencia. Las versiones de descodificadores MPEG-4 parte 10 con RVC ya incluían implementaciones basadas en el procesado en paralelo de luminancias y crominancias.
- Descodificador HEVC-RVC FBn: ésta es una de las últimas versiones desarrolladas, y en este caso se explota el paralelismo a nivel de imagen (*frame-based)*. El procedimiento consiste en replicar tantos descodificadores como imágenes en paralelo se deseen procesar al mismo tiempo. Una de las principales ventajas de esta versión radica en que de esta manera se puede paralelizar el módulo *Parser*, el cual no tiene paralelismo con el resto de unidades funcionales.

En el repositorio también se encuentran proyectos de aplicaciones no relacionados con el procesado de vídeo digital basados en RVC. A continuación se enumeran algunos de los más relevantes:

- *Crypto:* descripción de algoritmos para tareas de encriptación.
- *DigitalFiltering:* implementaciones de filtros FIR e IIR.
- *JPEG y Jpeg2000:* codificador/descodificador para JPEG.
- *Stereo:* Implementación de algoritmos de *stereo matching.*

2.4. Resumen

A lo largo de este capítulo se han presentado los principios de la codificación de vídeo digital, incluyendo un breve repaso a la evolución de los estándares de codificación en los últimos años.

Seguidamente se ha introducido el estándar H.265/HEVC. La característica más destacable de dicho estándar es la mejora de hasta un 50% en la tasa binaria necesaria para la transmisión de una secuencia de vídeo respecto de su antecesor H.264/AVC para un mismo nivel de calidad. Se han presentado las principales características del estándar, perfiles y niveles, así como lo principales bloques funcionales que integran los códecs.

Por último, se ha presentado el estándar de vídeo reconfigurable RVC. RVC permite realizar descripciones de alto nivel de descodificadores de vídeo e integra un conjunto de bibliotecas de MPEG con diversas funcionalidades necesarias en la implementación de este tipo de aplicaciones. Este estándar supone una revolución en el desarrollo de descodificadores de vídeo, ya que posibilita llevar a cabo diseños con una alta modularidad al tiempo que permite reutilizar elementos previamente desarrollados, reduciendo así los tiempos de desarrollo de las implementaciones. Además, permite llevar a cabo particionados funcionales de las FUs que integran los descodificadores en entornos multinúcleo. Esto se hace asociando la ejecución de cada una de los actores a cualquiera de los núcleos disponibles en el procesador.

3. Tecnologías de implementación

En este capítulo se realiza una revisión del estado del arte en las tecnologías de implementación de códecs de vídeo con procesadores multinúcleo. Se han considerado tres tipos de tecnologías: GPPs de altas prestaciones, GPPs orientados a sistemas empotrados y DSPs. Las tecnologías basadas en arquitecturas *hardware* específicas no se han tenido en cuenta porque las metodologías que se utilizan en su diseño están fuera del marco de esta tesis. A lo largo del capítulo se describen las características más relevantes de las tecnologías de implementación mencionadas y, para cada una de ellas, se recopilan las implementaciones más destacadas, poniendo el foco en los descodificadores de vídeo basados en el estándar HEVC.

En el apartado 3.1 se enumeran e introducen brevemente las tecnologías de implementación relevantes para el desarrollo de esta tesis. En el apartado 3.2 se muestra la arquitectura típica de los chips multinúcleo basados en GPPs de altas prestaciones, se analizan sus ventajas e inconvenientes y se recopilan las implementaciones más destacadas de códecs HEVC realizadas en base a ese tipo de tecnología, haciendo hincapié en las metodologías y herramientas utilizadas. En los apartados 3.3 y 3.4 se hace lo mismo para los chips multinúcleo basados en GPPs para sistemas empotrados y para los basados en DSPs, respectivamente.

3.1. Introducción a las tecnologías de implementación

Durante la última década los fabricantes de procesadores de todo tipo han venido desarrollado sistemas multinúcleo como alternativa a seguir aumentando la frecuencia de reloj de los chips para incrementar sus prestaciones.

A lo largo de los siguientes apartados se describen las principales características y ventajas ofrecidas por tres tipos de sistemas utilizados en el desarrollo de descodificadores de vídeo. En primer lugar, en el apartado 3.2, se estudian las particularidades de los sistemas integrados por procesadores tipo GPP de altas prestaciones. Estos sistemas, típicamente utilizados en los ordenadores personales (PCs), conforman dispositivos multipropósito típicamente utilizados en las fases de desarrollo y prueba de aplicaciones. Son los que ofrecen mayores prestaciones, pero su coste es elevado y su consumo energético es alto, requiriendo habitualmente el uso de disipadores activos y sistemas de ventilación.

Sin embargo, también existen dispositivos integrados por procesadores tipo GPP con menores prestaciones que son adecuados para ser utilizados en terminales móviles. Dichos sistemas conforman típicamente un sistema empotrado que integra, además de los GPPs, aquellos periféricos, memorias o dispositivos complementarios necesearios para el funcionamiento de una amplia variedad de diseños. Estos dispositivos se revisan en el apartado 3.3.

Por último, en el apartado 3.4 se detallan las particularidades e implementaciones con procesadores tipo Multi-DSP, que como se ha dicho a lo largo de esta memoria son el principal objetivo de esta tesis doctoral.

Cada tipo de arquitectura presenta distintas ventajas que marcan la idoneidad de ésta para su utilización en entornos diversos, tales como el consumo energético o las facilidades disponibles a la hora de realizar la programación sobre un sistema multinúcleo[9]. Sin embargo la utilización de técnicas como *Single Instruction, Multiple Data*[10] (Una Instrucción, Múltiples Datos – SIMD) o la sustitución de fragmentos de código por transcripciones directas de éstos en lenguaje ensamblador[11] pueden reportar importantes mejoras en el rendimiento, si bien su integración en el *software* puede ser costosa. La complejidad en la utilización de estas técnicas sobre las distintas plataformas no es idéntica entre ellas, así como la mejora derivada de este proceso.

3.2. Tecnologías de implementación basadas en GPPs de altas prestaciones

3.2.1. GPPs de altas prestaciones

Los procesadores más potentes están formados por varios núcleos GPP de altas prestaciones como los fabricados por Intel [INT] o AMD [AMD]. Estos procesadores soportan SOs como Windows o Linux, su programación está muy extendida y pueden desarrollar capacidades de cómputo muy elevadas. Además, las plataformas basadas en este tipo de procesadores (ordenadores personales, portátiles,…) tienen relativamente bajo coste y son muy accesibles. Por todas estas razones, las primeras implementaciones de un estándar de codificación que se desarrollan en el entorno científico suelen utilizar este soporte tecnológico. Es sobre este tipo de plataformas donde se encuentran las implementaciones capaces de descodificar en tiempo real secuencias de mayor calidad y resolución. Sin embargo, este tipo de sistemas tienen un elevado consumo de recursos, energía y espacio, por lo que no resultan ser eficaces para el desarrollo de sistemas

[9] Estas características dependen de diversos factores, como la compatibilidad de la arquitectura con APIs de gestión de multitareas, como OpenMP (capítulo 6).

[10] Las técnicas SIMD dependen de la arquitectura, y permiten conseguir un mayor paralelismo en el procesamiento de datos. Las instrucciones SIMD se basan en utilizar una sola operación sobre un conjunto amplio de datos siendo éstos procesados de forma simultánea por distintas unidades de procesamiento. Para un conocimiento más exhaustivo se recomienda consultar [DJL+06].

[11] El nivel de optimización del código máquina generado por un compilador no siempre es lo suficientemente elevado, por lo que éste puede ser reescrito por el desarrollador pudiendo obtener así importantes mejoras en el rendimiento de la aplicación. Este tipo de tareas son costosas y requieren de largos periodos de desarrollo.

empotrados. Tampoco su coste es lo suficientemente bajo para que resulten competitivos en este tipo de aplicaciones.

Otra ventaja de este titpo de plataformas, y de los SOs capaces de funcionar sobre ellas, es la disponibilidad del conjunto de herramientas *software* necesarias para los grupos de trabajo de desarrollo de aplicaciones. El hecho de que todas las fases de diseño, prueba y depuración puedan llevarse a cabo sobre la misma plataforma facilita y acorta los tiempos de diseño del *software*.

Además, la mayoría de GPPs de altas prestaciones, como los de las familias más recientes de los fabricantes Intel y AMD, disponen de los repertorios de instrucciones SIMD como SSE (SSE) [SSE] y *3Dnow!* (3DN) [3DN] respectivamente. Dichos repertorios de instrucciones pueden ser utilizados para optimizar el código de una aplicación, consiguiendo importantes mejoras en el rendimiento. Un ejemplo de su uso sobre descodificadores de vídeo HEVC se puede consultar en los trabajos publicados por Khaled et al. [JRS+14] y [JYS+16].

La Figura 17 muestra el diagrama de bloques genérico de un procesador de la familia i7 de última generación del fabricante Intel. Como se puede observar este tipo de procesadores integran, desde hace varios años, varios núcleos en su arquitectura. Además disponen de un nivel típicamente adicional de memoria caché (L3) con una capacidad elevada. También incorporan el acceso a aquellos periféricos y subsistemas externos de comunicación, visualización y gestión de memoria externa de alta capacidad. Más recientemente, este tipo de procesadores incorporan facilidades *hardware*, como el *HyperThreading* del fabricante Intel [INT], que permiten paralelizar la ejecución de los procesos emulando dos procesadores sobre un único núcleo físico, obteniendo rendimientos más elevados de las aplicaciones desarrolladas.

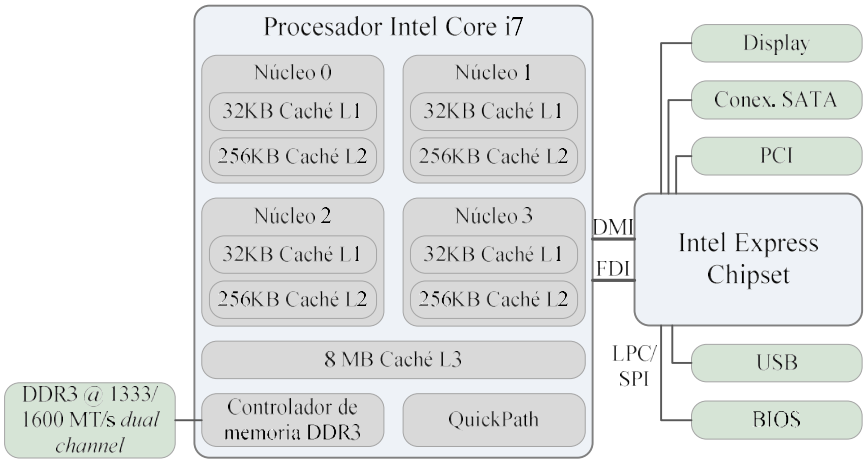


Figura 17. Diagrama de bloques de un procesador de la familia Intel i7.

A continuación se presenta una relación de algunas implementaciones desarrolladas sobre este tipo de arquitecturas.

3.2.2. Implementaciones desarrolladas sobre plataformas tipo GPP de altas prestaciones

En [YDSG12] investigadores del Instituto de Informática y Tecnología de la Universidad de Pekín presentan un descodificador HEVC implementado sobre un Intel i5 con optimizaciones SIMD. Tras identificar los módulos del descodificador de mayor carga computacional, en su caso la compensación de movimiento, el filtrado antibloques y la transformada inversa, proceden a optimizar el código de referencia del descodificador (versión HM 4.0) de dichos módulos mediante el uso de instrucciones SIMD para arquitecturas x86[12]. Con estas mejoras, el tiempo de ejecución de los módulos optimizados se ve reducido hasta en un 86% para la compensación de movimiento y el filtro antibloques, y hasta un 70% para la transformada. El número de imágenes por segundo (FPS) descodificadas se ve globalmente incrementado en 3,14 veces, respecto al rendimiento conseguido con el código del estándar, en el peor de los casos y hasta por 5,23 veces para la secuencia que presenta los mejores resultados. Sin embargo, no se hace ninguna mención a la utilización de varios núcleos de los dos disponibles en un Intel i5. En este trabajo se utilizan secuencias con resoluciones desde los 416x240 píxeles hasta 1920x1080 píxeles.

Posteriormente, los mismos investigadores publicaron un trabajo más extenso [DSY+14] en el que se realizan dos implementaciones de descodificadores HEVC, una sobre un Intel i7 y otra sobre un ARM Cortex-A9 (ver apartado 3.3). En este caso sí se utilizan varios núcleos de los disponibles en el procesador y los resultados se comparan con una de las implementaciones del descodificador OpenHEVC (ver capítulo 5). El uso combinado de instrucciones SIMD en ciertos módulos del descodificador, y la ejecución paralelizada del procesamiento a nivel de *frame* (*Frame Based* ver apartado 10.1.3) permiten obtener un factor de mejora medio en la tasa de imágenes por segundo de 6,32 con un procesador, respecto a los resultados con el descodificador de referencia (HM 10.0), y de 13,24 al utilizar simultáneamente los cuatro procesadores del Intel i7.

Por otra parte, uno de los grupos que lideran el desarrollo de implementaciones para aplicaciones de descodificación de vídeo es el ya mencionado IETR del INSA de Rennes (ver apartados 2.3.5 y 5.1). En algunos de sus trabajos más recientes [HRD14] y [HRD16], se presentan implementaciones del descodificador OpenHEVC (ver capítulo 5) incluyendo algunas de las extensiones incluidas en el estándar HEVC [SBCO+13], como *Scalable HEVC*[13] (SHVC) [ISO12]. En el primero de los trabajos se realiza una implementación del descodificador SHVC sobre un Intel Xeon con seis procesadores. Aplicando distintas técnicas de paralelismo contempladas en el estándar HEVC, como *Wavefront Parallel Processing* (WPP) [CHP11], se consigue la descodificación en tiempo real de secuencias de hasta 2560x1600 píxeles. En el segundo de los trabajos

[12] x86 Familia de procesadores cuyo repertorio de instrucciones se basa en la CPU del Intel 8086.
[13] La extensión SHVC del estándar HEVC contempla la implementación escalable de las características de resolución espacial, temporal y de calidad (o *Signal Noise Ratio* – SNR).

publicados se incluye la extensión *Multiview HEVC*[14] (MV-HEVC) [TWCH+14] en el código del descodificador OpenHEVC. En este caso se consigue descodificar en tiempo real una secuencia de resolución 4K a 60 FPS sobre un procesador Intel i7 de seis núcleos.

En otro trabajo [JRS+14] realizado por investigadores del IETR, se realiza una implementación de un descodificador HEVC-RVC sobre un Intel Xeon de seis núcleos. En esta ocasión se realiza una optimización del código generado por Orcc (ver capítulo 8) y se emplea la versión YUV del descodificador HEVC con el estándar RVC. Además de implementar el reparto del procesado de los actores en componentes YUV, lo que permite elevar el paralelismo de la solución, se integran en el código fuente llamadas a funciones optimizadas en lenguaje C para arquitecturas x86 incluidas en el código de OpenHEVC (ver capítulo 5). También se realiza una mejora en la implementación de las FIFOs que transmiten la información entre los actores. Todas estas mejoras permiten obtener, en el mejor caso, una ganancia de 7,6 con respecto a la implementación de referencia sin optimizaciones. Esta situación se da al utilizar cinco núcleos de los seis disponibles, pasando de una velocidad de descodificación de 6,1 FPS a 46,6 FPS. Cabe destacar que tanto este trabajo, como otras publicaciones que lo complementan [JYS+16], tienen una especial relevancia para esta tesis doctoral, como se verá en los apartados 9.4.2.1 y 9.7 de esta memoria.

3.3. Tecnologías de implementación basadas en GPPs para sistemas empotrados

3.3.1. GPPs para sistemas empotrados

Para satisfacer las necesidades de los sistemas empotrados, diferentes fabricantes han desarrollado procesadores basados en núcleos GPP de menores prestaciones que los anteriores. Estos procesadores son típicamente utilizados en terminales portátiles, tanto de telefonía móvil como de otros dispositivos multimedia cuya fuente energética es una batería. La utilización de esta alternativa tecnológica ha experimentado un incremento constante durante la última década, debido principalmente a su reducido coste, su bajo consumo energético y a una potencia de procesamiento siempre creciente. En la actualidad el fabricante ARM Holdings [ARM] es líder en este segmento de mercado, y sus procesadores Advanced RISC Machine (ARM) se utilizan por parte de fabricantes de dispositivos comerciales tan destacados como Samsung, Sony Ericsson, Apple o Nokia, entre muchos otros.

Este tipo de plataformas han experimentado un gran éxito comercial en el mercado de las telecomunicaciones durante la última década. A continuación se describen sus principales características:

- *Reduced Instruction Set Computer* (RISC – Computadora con un Conjunto de Instrucciones Reducido). Al contrario que las CPU basadas en la filosofía de diseño

[14] La extensión MV-HEVC del estándar HEVC permite la codificación de varias vistas en un solo *bitstream*.

CISC (*Complex Instruction Set Computing*), la arquitectura RISC busca un diseño de CPU que ejecute una serie de instrucciones reducidas, de tamaño fijo y reduciendo el número de accesos a memoria. Este tipo de arquitectura permite simplificar el diseño interno del procesador, reduciendo el número de transistores que lo integran. De esta manera se consigue reducir el consumo energético y la disipación.

- Consumo y disipación de calor. Debido al menor consumo energético de este tipo de plataformas la mayoría de ellas no requieren de sistemas activos para la disipación de calor, siendo suficiente una disipación pasiva mediante elementos metálicos conductores. Esto redunda en una reducción del consumo total de energía, además de reducir el espacio necesario dentro del dispositivo, así como el ruido producido por los ventiladores. En el apartado 7.3 se presenta la plataforma Odroid U3, utilizada durante la fase de desarrollo de esta tesis doctoral, la cual integra un procesador tipo ARM mulinúcleo.

3.3.2. Implementaciones desarrolladas sobre plataformas GPP para sistemas empotrados.

En [NBPM+14] investigadores del IETR del INSA de Rennes presentan una implementación del descodificador HEVC sobre la plataforma Odroid, con el SoC Exynos de Samsung (ver apartado 7.3), el cual dispone de ocho núcleos en su arquitectura, si bien sólo se utilizarán cuatro para la toma de resultados. El código escogido para la realización de pruebas es una versión del descodificador OpenHEVC con optimizaciones SIMD para esta arquitectura. El objetivo de este trabajo se centra en el ahorro de consumo energético orientado a terminales móviles mediante el uso de la técnica *Dynamic Voltage Frequency Scaling*[15] (DVFS). En este contexto presentan resultados en los que se consigue descodificar una secuencia de 1280x720 píxeles a 60 FPS en tiempo real, con un consumo aproximado de 1,1 W. Sin embargo, en los resultados publicados no se incluye el costo de la gestión y representación de las imágenes por pantalla.

Como se explicó en el apartado anterior, en [DSY+14] investigadores del Instituto de Informática y Tecnología de la Universidad de Pekín presentan dos implementaciones de descodificadores HEVC, una sobre un Intel i7 y otra sobre un ARM Cortex-A9 con dos núcleos en su arquitectura. La implementación sobre la plataforma ARM consigue una velocidad de descodificación de hasta 55 FPS para secuencias de 1280x720 píxeles, utilizando simultáneamente los dos núcleos disponibles en la plataforma. El código fuente utilizado como punto de partida es la versión HM 10.0. Por último, los autores no facilitan información sobre el consumo energético de la plataforma en este estudio.

[15] La técnica DVFS permite variar dinámicamente tanto la alimentación (tensión), como la frecuencia de funcionamiento, de un procesador con el fin de reducir su consumo energético.

En [BLRP14] investigadores de la Universidad de Genoa realizan una implementación del descodificador HEVC con optimizaciones SIMD sobre un ARM Cortex A9 con dos núcleos. En los resultados presentados se consiguen, en el mejor de los casos, tasas de descodificación de hasta 73 FPS para una secuencia con una resolución de 1280x720 píxeles. En este caso los autores tampoco facilitan información sobre el consumo energético de la plataforma.

Finalmente, en [RNH+16] investigadores del INSA de Rennes realizan una implementación del descodificador OpenHEVC (ver capítulo 5) sobre un procesador Exynos 5410 con cuatro núcleos ARM Cortex-A15 y cuatro núcleos ARM Cortex-A7. En esta implementación se han llevado a cabo optimizaciones de bajo nivel utilizando instrucciones SIMD para el procesador utilizado en aquellas secciones de código que más carga computacional suponen. También se incluyen medidas para paralelizar la ejecución del descodificador a nivel de imágenes (*frame-based*), y se implementan técnicas DVFS para la reducción del consumo energético. Los resultados muestran un factor de mejora de 3 en el mejor de los casos y al utilizar 64 hilos de ejecución, con respecto a la tasa de FPS con un núcleo. Finalmente, en el trabajo mencionado se muestra cómo la energía consumida al ejecutar una aplicación utilizando 4 hilos de ejecución se reduce en un 46% con relación a la que se consume utilizando un único hilo.

3.4. Tecnologías de implementación basadas en DSPs

3.4.1. Tecnología DSP

Finalmente, la tecnología DSP ha estado históricamente ligada al procesamiento de audio y vídeo. Debido a las limitaciones que presentan las plataformas basadas en DSP relacionadas con el soporte de SOs y de otros recursos (ver apartado 7.1) en muchas ocasiones, éstas se utilizan como aceleradores dentro de arquitecturas en las cuales un GPP realiza la gestión principal de la ejecución del *software* y despacha al DSP la ejecución de aquellos algoritmos susceptibles de ser optimizados sobre este tipo de arquitectura. Es por esto que en la literatura científica no suelen encontrarse implementaciones completas de descodificadores de vídeo sobre DSPs, sino que se suelen presentar optimizaciones de ciertos módulos que integran el descodificador.

En el caso de los DSPs, una de sus principales ventajas reside en el diseño de su arquitectura y de instrucciones específicas que permiten obtener importantes mejoras en el rendimiento, como se expuso en la tesis doctoral de Fernando Pescador [Pes11]. La arquitectura de los DSP está basada en el modelo *Super Harvard*, en dicho modelo existen memorias diferenciadas para datos e instrucciones. Estando ambas conectadas a la CPU mediante sus correspondientes *buses* de datos y direcciones. Cabe destacar que la CPU dispondrá de al menos un nivel de memoria caché, donde se copiarán de las memorias aquellos datos a procesar. Para el acceso a datos almacenados en memoria externa se dispone de un controlador de entrada/salida que gestiona la transferencia de los mismos a la memoria de datos. Por último, otra ventaja de los dispositivos DSP es

su reducido consumo energético, lo que los hace idóneos para su integración en terminales cuya fuente de alimentación sea una batería.

La Figura 18 presenta, a modo de ejemplo, un diagrama con los bloques que integran un procesador DSP multinúcleo de última generación del fabricante Texas Instruments [TI].

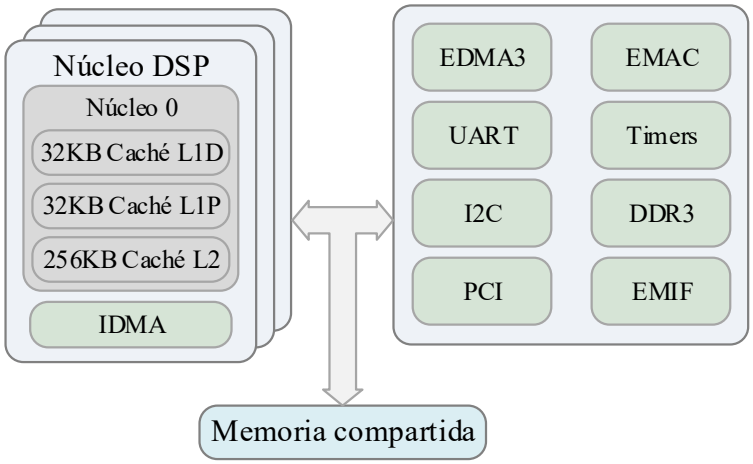


Figura 18. Diagrama de bloques de un procesador DSP multinúcleo de última generación.

El diseño del núcleo DSP se diferencia además por disponer de una unidad *Multiply-Accumulate* (MAC), capaz de realizar operaciones de multiplicación acumulación en un solo ciclo de reloj. En la Figura 19 se muestra un diagrama de una unidad MAC como las que se integran en este tipo de procesadores.

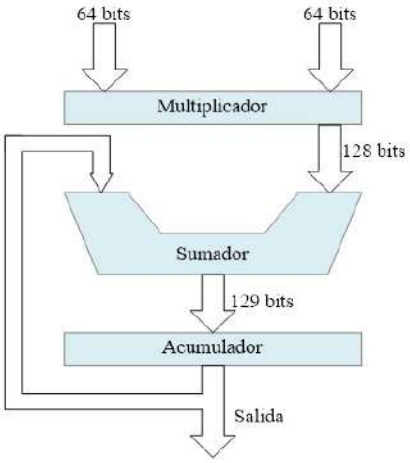


Figura 19. Diagrama de bloques de una unidad MAC en un procesador tipo DSP.

Por último, los procesadores DSP también soportan la ejecución de instrucciones SIMD sobre las unidades MAC, lo que permite a este tipo de plataformas superar en rendimiento a las plataformas GPP cuando se trata de realizar tareas de procesado numérico repetitivo.

3.4.2. Implementaciones desarrolladas sobre plataformas tipo DSP

Las plataformas DSP se emplean típicamente como aceleradores asociados a otros procesadores los cuales delegan el procesado de ciertos algoritmos a los núcleos DSP. Así por ejemplo, en [LSL+15] investigadores del centro de I+D DMC de Samsung en Suwon (Corea del Sur), realizan una implementación del descodificador HEVC en un SoC específico para televisión digital que trabaja a una frecuencia de reloj inferior a los 400 MHz. El SoC está dividido en dos unidades de procesamiento independientes que pueden funcionar de forma simultánea. Los resultados presentan que el sistema permite descodificar una secuencia de 1920x1080 píxeles en tiempo real (30 FPS).

En [PGJS13] se presenta una implementación del código de referencia (HM 5.0) del descodificador HEVC sobre el DSP de Texas Instruments TMS320DM6437, el cual dispone de un solo núcleo en su arquitectura. Los autores presentan una comparativa entre el nuevo estándar y su predecesor H.264. Se utiliza una secuencia de resolución 352x288 píxeles y QP 27 para la cual se consigue una tasa de descodificación de hasta 14 FPS. Se debe tener en cuenta que en esta implementación no se han llevado a cabo optimizaciones de bajo nivel.

Finalmente, en [BBA+15] se realiza la implementación de un codificador H.264/AVC sobre la plataforma TMS320C6678. En este trabajo, investigadores de la Universidad de Sfax presentan un codificador HD capaz de funcionar en tiempo real (26 FPS) y una resolución de 1280x720 píxeles. Este resultado se consigue aprovechando los ocho núcleos de la plataforma y realizando la codificación a nivel de imagen de forma simultánea.

3.5. Comparativa entre las distintas implementaciones

Con el fin de tener una visión más global de las implementaciones vistas en este capítulo, se ha establecido un índice de calidad que permite comparar diferentes realizaciones. Mediante este índice, cuya expresión se muestra en la Fórmula 1, se normaliza la tasa de imágenes por segundo descodificadas, FPS, ajustando el número de núcleos a uno, la frecuencia de reloj del sistema a 1GHz y una resolución estándar de 1280x720 píxeles.

$$Indice = FPS * \frac{Resolución/(1280 * 720)}{Nºnúcleos * Frecuencia\ de\ reloj}$$

Fórmula 1. Fórmula para calcular el índice de calidad de las implementaciones propuesto.

Comparar implementaciones llevadas a cabo sobre distintas plataformas con distintas versiones del *software* y secuencias de vídeo de diferente resolución y calidad siempre es complejo. Características relevantes de las plataformas, tales como las velocidades de las memorias, el tamaño de éstas, o el tiempo de desarrollo invertido, son importantes para una comparación detallada. Sin embargo, muchas veces los autores no facilitan toda la información en sus trabajos. Los datos que se resumen en la Tabla 6 representan una buena solución de compromiso entre los datos disponibles y la

calidad de la comparativa. Como se puede observar, las mejores puntuaciones se dan para aquellas implementaciones basadas en GPPs de altas prestaciones. Como se ha explicado anteriormente estas plataformas disponen de unas prestaciones muy elevadas, pero no resultan útiles en sistemas multimedia portátiles o en aquellos cuya fuente de energía sea una batería.

Referencia	Procesador	Núcleos	Frec. (GHz)	FPS	Resolución	Consumo (W/h)	SE	Índice
[YDSG12]	GPP Intel i5	2	3,4	27*	1080p	--	No	8,9
[DSY+14]	GPP Intel i7	4	3,4	122*	1080p	--	No	20,1
[HRD14]	GPP Intel Xeon	6	3,2	84**	1080p	--	No	9,8
[HRD16]	GPP Intel i7	6	3,4	60**	4K	--	No	26,4
[JRS+14]	GPP Intel Xeon	5/6	3,2	46	1080p	--	No	6,7
[NBP+14]	GPP ARM Exynos	4/8	1,6	60	720p	1,1	Sí	9,3
[DSY+14]	GPP ARM Cortex A9	2	1,2	35 ***	720p	0,5****	Sí	14,5
[BLRP14]	GPP ARM Cortex A9	2	1,2	33 ***	720p	0,5****	Sí	13,7
[PGJS13]	DSP 320DM6437	1	0,6	14	288p	--	Sí	2,5
[LSL+15]	SoC específico	2	0,35	30	1080p	--	--	96,4

** Secuencia Clase B BasketBallDrive QP 27.*
*** Se incluye extensión SHVC y/o MV-HEVC.*
**** Secuencia Clase C KristenAndSara QP 27.*
***** Consumo medio del SoC.*

Tabla 6. Comparativa de las distintas implementaciones analizadas.

De entre todas las implementaciones destaca el elevado índice obtenido por la presentada en [LSL+15]. Esto se debe a que es una implementación diseñada específicamente para un SoC industrial (ASIC), donde se realiza un aprovechamiento elevado de los recursos de la plataforma mediante un desarrollo muy optimizado. Debe tenerse en cuenta que la metodología seguida en esta implementación es radicalmente diferente a la seguida en el resto de trabajos evaluados.

3.6. Resumen

A lo largo de este capítulo se han presentado los soportes tecnológicos programables sobre los que se vienen desarrollando descodificadores de vídeo. También se han presentado las particularidades de cada arquitectura desde un punto de vista general, contraponiendo sus respectivas ventajas y desventajas. Actualmente las implementaciones sobre plataformas tipo GPP de altas prestaciones tienen un uso mayoritariamente experimental y como prueba funcional, mientras que los GPPs tipo ARM han ido tomando una relevancia cada vez mayor acorde con el desarrollo de sistemas cada vez más potentes al tiempo que mantienen un consumo energético reducido. En ambos casos la utilización de procesadores multinúcleo está plenamente instaurada y su programación eficiente sigue siendo uno de los retos en desarrollo.

Por otro lado las plataformas tipo DSP o Multi-DSP siguen desempeñando una parte importante del procesamiento de vídeo, ya sea como co-procesadores aceleradores dentro de un sistema que tenga como centro un GPP, o ya sea formando parte de un SoC específico. En este sentido su programación supone un reto mayor que en las plataformas tipo GPP, no habiéndose encontrado en la literatura científica ninguna referencia a una metodología de diseño que sirva como guía. Este hecho refuerza las motivaciones planteadas al comienzo de este trabajo de investigación.

4. Metodologías y herramientas

Durante la última década, la comunidad científica ha dedicado un considerable esfuerzo a la investigación en metodologías y herramientas para la programación de sistemas multinúcleo. En este capítulo se hará un repaso de algunos de los trabajos de investigación más relevantes que se han desarrollado en los últimos años o que se están desarrollando en la actualidad.

Así, tras unas breve introducción (apartado 4.1), en los apartados 4.2, 4.3 y 4.4 se describen los trabajos en los que se proponen metodologías y herramientas que permiten la automatización de los flujos de diseño de aplicaciones para chips multinúcleo. Finalmente, en el apartado 4.5 se describe la metodología y las herramientas utilizadas como punto de partida para el desarrollo de esta tesis.

4.1. Introducción

El diseño de aplicaciones destinadas a funcionar sobre plataformas multinúcleo plantea una serie de retos de diversa índole. En muchos casos la complejidad de los hitos a superar depende tanto de las dificultades planteadas por el, o los, sistemas sobre los que se quiere trabajar, como de las aplicaciones a las que se quiere dar soporte.

A lo largo de los últimos años se han ido desarrollando toda una amplia gama de sistemas multiprocesador para sistemas empotrados con mucho éxito en el mercado debido a su relativa alta capacidad computacional, su bajo coste y un consumo energético reducido. Este tipo de sistemas se suman a aquellos de mayor recorrido histórico, como los que integran los sistemas de los ordenadores personales o PCs, y que ofrecen un rango más elevado de prestaciones, pero con un coste mayor y consumos energéticos elevados para ser utilizados en terminales portátiles.

Por otro lado, la tendencia comercial dominante durante la última década destaca por la continua salida al mercado de nuevos terminales con soporte multimedia, como *smartphones* o tabletas, integrados por sistemas empotrados. Además, el diseño de las aplicaciones e interfaces de usuario para este tipo de plataformas ha visto acortados los tiempos de desarrollo debido a los exigentes plazos comerciales.

En este contexto, la búsqueda de metodologías que den soluciones al desarrollo de *software* compatible con los sistemas multiprocesador en tiempos cada vez más reducidos se ha convertido en el objetivo a alcanzar tanto por parte de la industria como por la academia.

4.2. Proyecto HEAP

El proyecto *Highly Efficient Adaptive multi-Processor framework* (HEAP) ha sido financiado por la Unión Europea (UE) y desarrollado entre los años 2010 y 2013 bajo la referencia FP7-ICT 247615 [HEAP] por parte de distintas empresas y universidades de la UE, entre las que destacan STMicroelectronics SRL (Italia), Thales (Francia), la Universidad Politécnica de Turín (Italia), o la Universidad de Genoa (Italia). Este proyecto surge con dos objetivos principales:

1) Diseñar un conjunto de herramientas que permitan a los programadores analizar y paralelizar secciones de código serie[16] existente.

2) A partir de los datos obtenidos de distintos análisis, mejorar el aprovechamiento de las memorias del sistema, con especial énfasis en las memorias caché, permitiendo adaptar el uso que se hace de éstas en cada aplicación a desarrollar.

Al comienzo del desarrollo de este proyecto se fijaron unos objetivos concretos de mejora: (1) reducir un 20% el tiempo de desarrollo necesario en paralelizar una aplicación ya existente, (2) reducir en otro 20% el consumo de energía derivado de las operaciones del mantenimiento de la coherencia de memoria, y (3) mejorar el rendimiento de las memorias, asegurando la coherencia de los datos almacenados, en un 20%.

HEAP utiliza el compilador Compaan [COM] [KRD00] como herramienta central. Este compilador permite generar modelos de flujos de datos en base a Redes de Procesos Kahn (*Kahn Process Networks* – KPN) [Kah74] (ver apartado 4.5.1) a partir de especificaciones en C, además se reduce la complejidad del código mediante la descripción de *Static Affine Nested Loop Programs* (SANLP). Mediante el uso de SANLPs se pueden tomar decisiones sobre el control del flujo de datos y el acceso a variables en tiempo de compilación permitiendo optimizar la solución y obtener un mayor grado de paralelismo. A partir de un diagrama KPN el compilador Compaan asignará a cada vértice del grafo un hilo de ejecución y resolverá las dependencias de datos entre hilos mediante el uso de FIFOs.

Adicionalmente a lo anteriormente descrito, se ha desarrollado un conjunto de herramientas llamado HEAP *optimistic software parallelization toolset* que da soporte a la paralelización, dentro de este entorno, a aquellas aplicaciones que requieran de modificaciones dinámicas en el uso y localización de memorias, o donde el comportamiento de sus elementos funcionales no pueda ser descrito de forma estática en

[16] El código serie se refiere a aquel cuya ejecución se va a realizar de forma secuencial por un único núcleo.

tiempo de compilación. Si bien esta herramienta accesoria complementa la funcionalidad de HEAP, debe ser el desarrollador de forma manual y en base a su experiencia, el que en estos casos extraiga el paralelismo de la aplicación.

A lo largo del proyecto se han realizado numerosas publicaciones en la literatura científica que pueden consultarse para tener un mejor conocimiento del mismo, todas ellas están disponibles en el sitio web del proyecto [HEAP], algunas de las más destacadas son [LLP+13], donde se presentan el conjutno de herramientas desarrolladas dentro del proyecto, y [RK12], donde se presenta un protocolo para el manteniemiento de la información almacenada en las memorias de los sistemas multinúcleo.

4.3. Proyecto ALMA

El proyecto ALMA (*Architecture oriented paraLlelization for high performance embedded Multicore systems using scilAb*) [GCV+13] ha sido desarrollado entre los años 2011 y 2015 por parte de varias entidades y centros de investigación de la UE, entre los que destacan Fraunhofer (Alemania), Universidad de Rennes 1 (Francia) o la Universidad del Peloponeso (Grecia) entre otros, y ha sido financiado por la UE según la referencia FP7-ICT 287733 [ALM].

El proyecto ALMA ofrece una solución para el desarrollo de aplicaciones sobre plataformas multinúcleo, abstrayéndose de la complejidad de las capas que representan las aplicaciones y las plataformas; esto se consigue mediante un conjunto de herramientas en torno al *software* Scilab[17] [SCI]. Tanto los conceptos teóricos, como las pruebas realizadas a lo largo del proyecto han sido publicados en distintos medios de difusión científica, entre las que destacan [Str13] y [BOR+14]. En estos trabajos se profundiza en los detalles del proyecto, se introducen mejoras y se difunden los resultados.

El flujo de trabajo con ALMA se presenta de forma resumida en la Figura 20. Como se puede observar, a partir de una descripción de alto nivel de la aplicación en Scilab se genera una Representación Intermedia[18] (*Intermediate Representation* – IR) de la misma, se llevan a cabo distintas optimizaciones y se trata de explotar el paralelismo de la arquitectura. Se han elegido dos plataformas como banco de pruebas de la metodología, por un lado KIT's KAHRISMA [KBS+10] con fines educativos o de investigación, y por el otro se ha elegido un SoC de *Recore Syste*ms [REC] para fines industriales y/o comerciales.

[17] Scilab (*Scientific Laboratory*) es un software similar a MATLAB, de uso libre para el procesado matemático y que dispone de un lenguaje de programación de alto nivel.

[18] Algunos de los conceptos utilizados en este apartado son comunes a las distintas metodologías estudiadas, aquellos compartidos con la metodología utilizada durante la realización de la tesis doctoral se explican con mayor detalle en apartados posteriores. En concreto los conceptos de Representación Intermedia (IR), *front-end, middle-end y back-end* se explican en el capítulo 8.

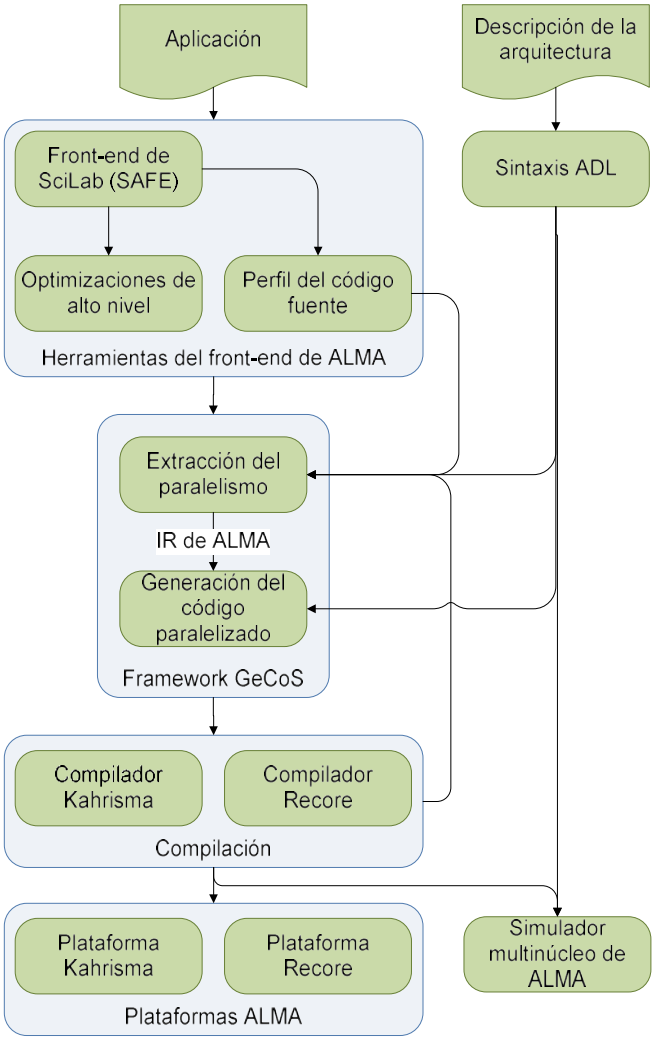


Figura 20. Diagrama del flujo de trabajo según el proyecto ALMA.

En el *front-end* de ALMA se ha incluido un perfilador (*profiler*) que realiza un primer análisis del *software* a nivel de bloques funcionales. Mediante este proceso, se realiza un primer particionado funcional que permite obtener una optimización de alto nivel. Este *profiler* puede ser utilizado en una segunda iteración para localizar puntos calientes o *hot-spots* donde deba llevarse a cabo una optimización más pormenorizada.

En cuanto a la generación de código fuente, ALMA contempla optimizaciones de alto nivel, al mismo tiempo que permite la utilización de instrucciones SIMD. Durante todo el proceso se dispone además de una descripción de alto nivel de la arquitectura, utilizada principlamente en la optimización del código a paralelizar, según la especificación *Architecture Description Language* (ADL) [SKB11]. Dicha descripción sigue un modelo jerárquico en el que se incluye el comportamiento de cada módulo disponible en la arquitectura.

4.4. PREESM

El *framework Parallel and Real-time Embedded Executives Scheduling Method* (PREESM) [PRE] es una herramienta de código abierto desarrollada por investigadores

del grupo IETR del INSA de Rennes (Francia) que facilita la simulación y generación de código fuente para sistemas empotrados y/o plataformas multinúcleo heterogéneas [PDH+14]. Mediante el uso de un lenguaje basado en flujos de datos permite extraer fácilmente el paralelismo de aplicaciones basadas en descripciones PiSDF (*Parametrized & Interfaced Synchronous Data-Flow*) [DPN+13]. PREESM está integrado por una serie de *plugins* para el IDE de Eclipse [IET14].

PREESM ha sido desarrollado con fines educativos y de investigación, y utilizado con éxito para la implementación de sistemas de telecomunicaciones, sistemas multimedia y aplicaciones de visión artificial sobre distintas plataformas heterogéneas [PAPN12] [HDN+12], entre otros.

Las aplicaciones basadas en Modelos de Computación (MoCs – ver apartado 4.5.1) y cuyos flujos de datos son de tipo PiSDF (ver apartado 4.5.1) permiten la división de una aplicación en actores, los cuales se comunican mediante FIFOs. Sin embargo, el comportamiento en la producción y consumo de *tokens* por parte de los actores puede modificarse mediante una serie de parámetros estáticos. De esta forma la descripción de los algoritmos que integran la aplicación sigue un modelo perfectamente predecible.

La Figura 21 muestra el flujo de trabajo con PREESM. Existen tres elementos sobre los que se basa este *framework*: una descripción gráfica de la aplicación (diagrama PiSDF), una descripción gráfica de la arquitectura sobre la que se desea trabajar según un modelo conocido como *System-Level Architecture Model*[19] (S-LAM), y una descripción del comportamiento global en formato *xml*[20] llamada *Scenario* (Escenario). Este escenario contiene además la información que permite enlazar los algoritmos de la aplicación con la arquitectura, pudiendo determinar los tiempos de ejecución de cada uno de los actores en cada uno de los núcleos disponibles en la arquitectura.

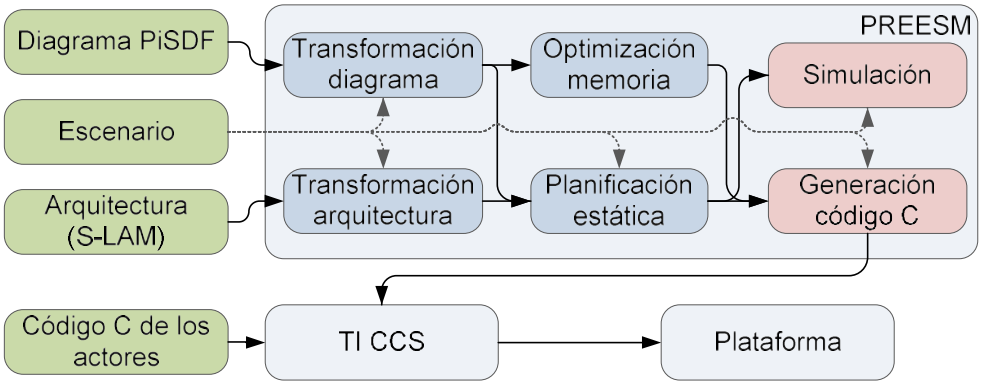


Figura 21. Flujo de trabajo con PREESM.

[19] S-LAM es un modelo de una arquitectura de alto nivel que permite estudiar los posibles cuellos de botella durante el procesado de algoritmos.
[20] xml de sus siglas en inglés *Extensible Markup Language* es un formato de texto flexible derivado de la norma ISO 8879.

4.5. Metodología basada en flujos de datos DPN

Esta es la metodología utilizada como guía para la realización de esta tesis doctoral. Las principales razones que motivan su elección en este trabajo son (1) que el estándar RVC tiene su foco de utilización sobre descodificadores de vídeo digital, y (2) que como se mencionaba en el apartado 2.3.5, varios grupos de investigación que colaboran con el GDEM vienen trabajando con este entorno desde hace tiempo, al igual que el propio doctorando, pudiendo aprovechar de esta manera la amplia experiencia ya adquirida.

Cabe destacar que debido a la relación de colaboración existente entre los investigadores del IETR que han desarrollado PREESM e investigadores del GDEM se valoró, durante la fase de desarrollo de esta tesis doctoral, la utilización de PREESM para el desarrollo de descodificadores de vídeo basados en flujos de datos y especificaciones RVC sobre plataformas Multi-DSP. Sin embargo, debido a que los flujos de datos en base PiSDF limitan la expresividad de los actores que integran la aplicación objetivo, numerosos actores que implementan la funcionalidad de los descodificadores en base RVC quedaban excluidos, principalmente aquellos encargados de ejecutar los algoritmos de predicción inter, haciendo imposible la compatibilidad directa entre este tipo de aplicaciones y PREESM.

Este hecho no anula la posibilidad de utilizar PREESM para explotar, de forma independiente, el paralelismo de ciertos actores, como los que integran la transformada inversa y cuyo comportamiento es determinístico o cuasi-estático.

4.5.1. Modelo de computación

Un Modelo de Computación (del inglés *Model of Computation* – MoC), es un conjunto de elementos que pueden utilizarse para describir el comportamiento de una aplicación. Éste debe servir como interfaz entre el campo matemático o algorítmico y la aplicación informática. No se debe confundir un MoC con un lenguaje de programación, ya que lo que hace es dar una especificación independientemente de la sintaxis del lenguaje. En realidad un lenguaje de programación puede implementar uno o varios MoCs.

Los MoCs permiten al desarrollador abstraerse de las capas más bajas de la implementación y centrarse en el desarrollo de aplicaciones de alto nivel, pudiendo así construir modelos no dependientes de la aplicación y para distintas plataformas.

Existen distintos tipos de MoCs, ya que pueden basarse en modelos tipo FSM [SGT10], mediante sucesión de eventos discretos o redes Petri [MMS94], entre otros. Sin embargo, a la hora de abordar la paralelización de aplicaciones sobre distintas arquitecturas, los MoCs más versátiles son aquellos que se basan en flujos de datos. En este sentido el primer MoC basado en un flujo de datos fue introducido en los años setenta [Kah74]. Desde entonces han surgido un gran número de modelos, que pueden clasificarse en función de dos factores principales: expresividad y grado de predicción. La expresividad de un modelo es inversamente proporcional a su grado de predicción,

es decir, a mayor diversidad de los algoritmos implementados e integrados en el flujo de datos menor será la posibilidad de poder predecir el comportamiento del modelo antes de su ejecución. La Tabla 7 resume las propiedades de los distintos modelos existentes.

	SDF	ADF	IBSDF	DSSF	PSDF	PiSDF	SADF	SPDF	DPN	KPN
Grado de expresividad	Bajo				Medio		Alto (*Máquina de Turing*)			
Jerárquico			✓	✓	✓	✓				
Compositivo			✓	✓		✓				
Reconfigurable					✓	✓	✓	✓	✓	✓
Planificación estática	✓	✓	✓	✓						
Determinable	✓	✓	✓	✓	*	*	✓	*		
Tasa variable		✓			✓	✓	✓	✓	✓	✓
No determinístico							✓	✓	✓	

SDF: Dataflow síncrono
ADF: Dataflow afín
IBSDF: Dataflow basado en interfaz
DSSF: SDF determinístico con FIFOs compartidas
PSDF: SDF parametrizado
PiSDF: SDF parametrizado y con interfaz
SADF: Dataflow con escenario de aviso
SPDF: Dataflow paramétrico planificable
DPN: Dataflow de red de procesos
KPN: Red de procesos Kahn

Tabla 7. Propiedades de los distintos tipos de MoCs basados en flujos de datos.

En función del tipo de flujo de datos elegido la implementación de los actores o entidades funcionales que integran una aplicación tendrá distintas características. Para que un diagrama tenga un alto grado de predicción el sistema debe estar compuesto por actores cuyo funcionamiento sea determinístico, pudiendo conocer el comportamiento de todo el sistema antes de su ejecución, un ejemplo de este caso es el modelo SDF [LM87]; en este caso los actores producirán un número de *tokens* predecibles para un número de *tokens* consumidos. Por el contrario en los modelos como DPN [LP95] pueden darse situaciones no determinísticas, resultando un comportamiento no predecible. En estos casos el comportamiento global del sistema sólo puede observarse en tiempo de ejecución, incluso el comportamiento en sucesivas ejecuciones puede variar aun estimulando el sistema con las mismas variables.

Existen modelos intermedios, como PSDF o PiSDF que se sirven de modificaciones introducidas sobre las unidades funcionales que integran el modelo y que aumentan la expresividad del mismo. El modelo PiSDF es la base del conjunto de herramientas que conforman PREESM (ver apartado 4.4), y cuyas limitaciones ya fueron expuestas.

En este punto podría concluirse que los modelos estáticos, como SDF o IBSDF, ofrecen una ventaja considerable al poder predecir el comportamiento detallado de las entidades que lo integran, sin embargo, para que un diagrama cumpla los requisitos de este tipo de modelos todos los actores deben tener un comportamiento estático, reduciendo de esta manera el grado de funcionalidad de los mismos. Así, bajo estos modelos, la implementación de ciertas unidades funcionales, como por ejemplo los bloques de predicción en un descodificador de vídeo digital no resulta factible, ya que los algoritmos que deben implementarse pueden tomar diferentes decisiones para un mismo parámetro de entrada.

Así pues la elección del MoC dependerá en gran medida de la funcionalidad que se desee expresar mediante el mismo. También deberán tenerse en cuenta las herramientas que den soporte a cada modelo, la facilidad de acceso a las mismas y su grado de desarrollo. El uso de una herramienta específica para cada MoC es clave para poder extraer el mayor grado de optimización posible de cada uno de ellos, si bien herramientas como Orcc, diseñada para modelos DPN, podrían dar soporte a modelos con un menor grado de expresividad.

El estudio e implementación de aplicaciones con los distintos tipos de flujos de datos es un campo de análisis amplio y excede los objetivos de esta memoria, para un mejor entendimiento de la temática se recomienda acudir a la literatura científica donde puede ampliarse esta información. En [CHL97] se detalla la utilización de distintos MoCs en la herramienta Ptolemy, y en [Kah74] se introducen los modelos KPN para MoCs.

4.5.2. Estrategias de planificación

Una vez se elige el modelo de computación (DPN en este caso) y se implementan las distintas unidades funcionales que integran la aplicación se debe gestionar la ejecución de las mismas. En el caso de los descodificadores de vídeo desarrollados con RVC se especifican tres niveles de planificación (o *scheduling*[21]) que permiten gestionar todo el funcionamiento de un descodificador.

4.5.2.1. Planificación a nivel de acciones

En primer lugar se encuentra la planificación interna de los actores. Este concepto se introdujo en el apartado 2.3.3, donde se presentaba el funcionamiento interno de un actor, gobernado por una FSM que planifica la ejecución de las distintas acciones que lo componen en función de una serie de condiciones de guarda, prioridades y variables de estado interno. A este nivel de planificación se le denomina planificación de acciones o *action scheduler.* La Figura 22 ilustra este mecanismo, donde en función de los datos de entrada (lectura) y del estado interno del actor, el planificador de acciones ejecutará aquellas acciones que consuman los datos de la FIFO de entrada, produciendo *tokens* en la FIFO de salida.

[21] A lo largo de este apartado se facilita la nomenclatura en lengua inglesa de algunos conceptos claves para la compresión de las implementaciones realizadas y que se utilizan en capítulos posteriores.

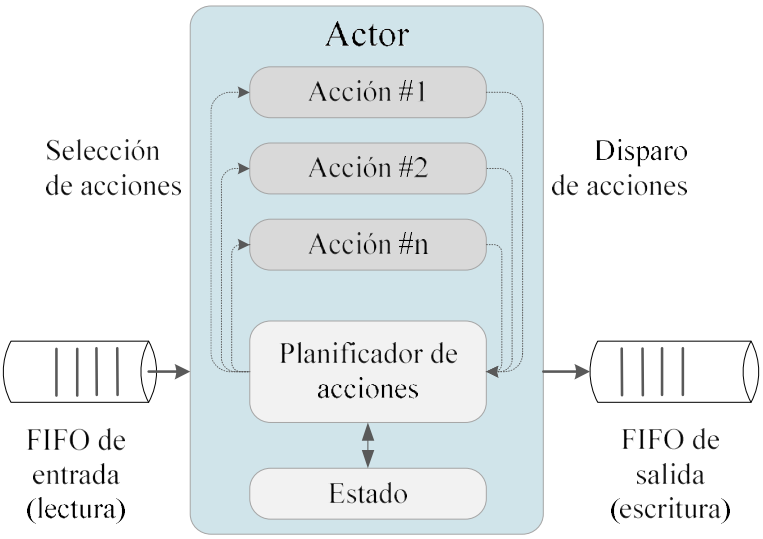


Figura 22. Planificación a nivel de acciones.

4.5.2.2. Planificación a nivel de actores

El siguiente nivel de planificación es el que se refiere a la gestión de la ejecución de los actores que integran una aplicación. Normalmente serán varias decenas los actores que conformen una aplicación, por lo que es preciso prestar atención al orden en que los actores pasan a ejecución. Un actor ejecutará alguna de sus acciones si en sus FIFOs de entrada hay *tokens* disponibles y en sus FIFOs de salida hay espacio libre. De esta forma la ejecución de los actores dependerá en gran medida de la ejecución y producción de *tokens* de sus actores precedentes. Sin embargo la ventaja de este modelo de gestión reside en que en todo momento habrá un actor en ejecución, eliminando así innecesarios estados de espera o tiempos muertos durante el procesado. Las FIFOs deben conectarse a los actores respetando que sólo un actor escriba en cada FIFO, mientras que varios pueden leer de la misma FIFO. La Figura 23 muestra un ejemplo de un diagrama DPN con varios actores interconectados mediante FIFOs.

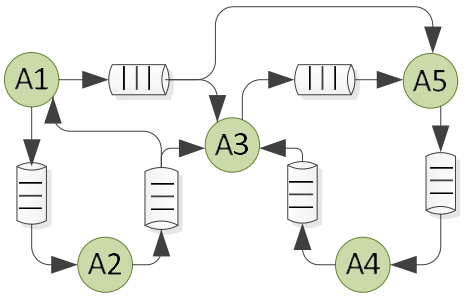


Figura 23. Ejemplo de diagrama DPN con cinco actores interconectados mediante FIFOs.

Existen dos estrategias de planificación de la ejecución de los actores: *Round Robin* (RR) y *Data-driven/Data-demand* (DD/DD). La estrategia RR consiste en la ejecución secuencial de los actores, en este caso el disparo de las acciones dependerá únicamente del cumplimiento de las condiciones del *planificador de acciones* y de los

datos en las FIFOs. En el caso DD/DD la planificación de los actores a ejecutar se realiza basándose en la demanda de datos en las entradas y salidas de los mismos. De esta forma, la planificación se lleva a cabo ejecutando los actores predecesores productores de *tokens* y requeridos por los sucesores. De la misma forma si la salida de un actor está llena se llamará al actor sucesor para que consuma los *tokens* que están disponibles. La Figura 24 ilustra ambas estrategias de planificación.

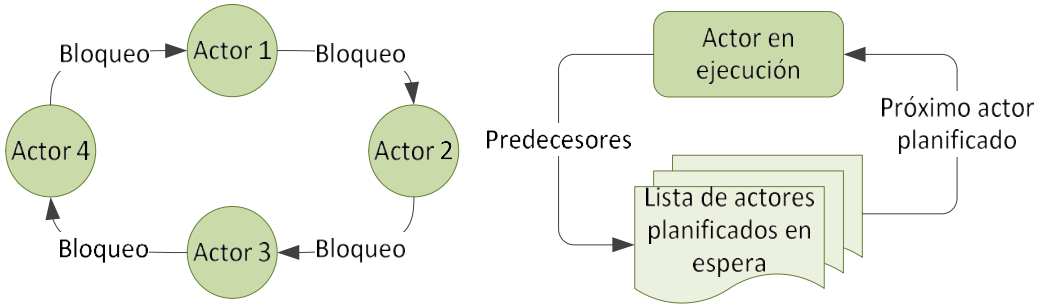


Figura 24. Estrategias *Round-Robin* (izquierda) y *Data-driven/Data-demand* (derecha).

En ambas estrategias de planificación no será necesario almacenar información relativa a la ejecución de los actores, ya que toda la planificación depende de los datos transmitidos mediante las FIFOs. Esto proporciona una ventaja fundamental en arquitecturas multinúcleo y permite reducir el coste computacional de las implementaciones. Para el desarrollo de esta tesis se ha utilizado en todo momento la estrategia *Round Robin*. La utilización de la estrategia RR viene motivada por los siguientes factores: ésta es la estrategia utilizada por defecto en las implementaciones que utilizan el estándar RVC, lo que facilita una comparativa posterior de los resultados con otras implementaciones. Por otro lado, la estrategia DD/DD conlleva un aumento en la complejidad del planificador de actores, así como la necesidad de asegurar la coherencia de la información compartida entre los planificadores.

4.5.2.3. Planificación a nivel de procesadores

Con el fin de evitar posibles situaciones de bloqueo de este tipo de topologías sobre entornos multinúcleo hay un tercer nivel de planificación. En este caso la información a transmitir será un listado de los propios actores que deben pasar a ejecución por parte de algún procesador, bien porque sea preciso que consuma *tokens* disponibles en sus entradas o bien porque sus actores sucesores están esperando información en sus respectivas entradas. De darse una situación como ésta en un entorno multinúcleo los planificadores encargados de controlar la ejecución de los actores en cada núcleo deberán poder compartir información relativa a la ejecución de los mismos. Así, se diseña un nuevo tipo de canal FIFO llamado *scheduling-fifo* (FIFOs de planificación), el cual se utilizará únicamente para transmitir este tipo de información. La Figura 25 ilustra este tercer nivel de planificación mediante un diagrama con dos núcleos y sus respectivos planificadores.

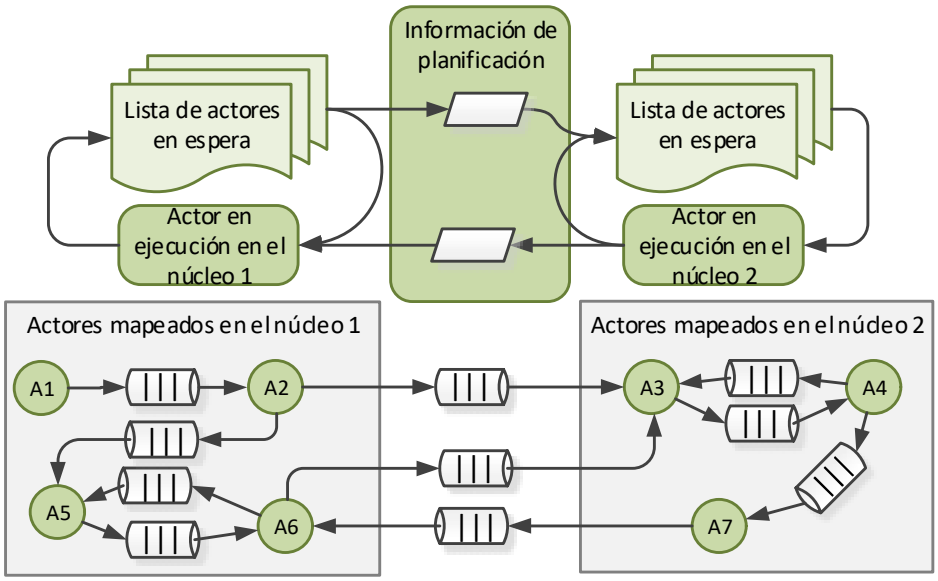


Figura 25. Ilustración de la planificación a nivel de procesadores y actores.

4.5.3. Aplicación de estrategias de planificación sobre plataformas multinúcleo

Como ya se ha explicado antes (ver apartados 2.3 y 4.5.1), una de las principales ventajas de los modelos basados en flujos de datos es la facilidad de implementar éstos sobre plataformas multinúcleo. En este sentido resulta clave el hecho de que las variables propias de cada actor no requieren su utilización por parte de los restantes actores, y que la única información compartida se transmite y aloja en FIFOs. De esta manera se puede ejecutar una aplicación integrada por un número determinado de actores organizados y planificados en listas. En entornos multiprocesador se asignará una lista de actores a cada uno de los núcleos disponibles en la plataforma, quedando así cada una de las listas asociadas a un hilo de ejecución (*thread)*. Al proceso de repartir la carga de los actores entre los procesadores de una plataforma se le llama mapeo (*mapping).*

Para llevar a cabo la ejecución en paralelo de actores sobre plataformas multinúcleo se deben cumplir dos condiciones fundamentales: que las FIFOs estén alojadas en memoria compartida accesible por parte de aquellos actores que leen o escriben en ellas, y que cada actor quede mapeado en un núcleo, según la expresión de la Fórmula 2.

$$(Actores\ en\ núcleo\ 0) \cup (Actores\ en\ núcleo\ 1) \cup (Actores\ en\ núcleo\ N) = \{\emptyset\}$$

Fórmula 2. Fórmula a cumplir al realizar un mapeo de actores.

Numerosos factores deben tenerse en cuenta a la hora de realizar un mapeo de actores entre los núcleos de un sistema multinúcleo; cuanto mejor sea el mapeo de las unidades funcionales mayor ganancia se obtendrá con respecto a un sistema mono-núcleo.

- *Peso del actor* o carga computacional derivada de la ejecución del mismo. Idealmente un mapeo debe resultar en que la suma de las cargas computacionales de los actores de la lista para cada procesador quede balanceada entre todos los núcleos. Por ejemplo en una distribución para cuatro núcleos el sumatorio del peso de los actores mapeados en cada núcleo debería suponer el 25% de la carga computacional total de la aplicación. Sin embargo, en un MoC dinámico, como es el caso de DPN, esta información sólo se puede obtener en tiempo de ejecución.

- *Paralelismo de los actores,* la funcionalidad que se integra dentro de cada actor tendrá un mayor o menor grado de paralelismo con las acciones del resto de actores. De esta manera se debe buscar la distribución entre los procesadores de aquellos actores cuya funcionalidad se puede realizar de forma paralela a otras tareas, evitando al máximo ejecuciones secuenciales de actores.

- *Reducción del número de comunicaciones entre núcleos*. El peso de las comunicaciones entre actores mapeados en distintos núcleos debe reducirse. Dado que habitualmente la información compartida entre actores ubicados en diferentes núcleos debe alojarse en memorias compartidas, de menor velocidad que las memorias cachés de los procesadores, deberán reducirse, en la medida de lo posible, este tipo de comunicaciones.

- *Reducción del volumen de las comunicaciones*. En relación con la anterior consideración también resulta importante la carga relativa de cada una de las comunicaciones. Se debe tener en cuenta que no todas las FIFOs tienen la misma actividad, aquellas que se utilicen para transmitir un mayor número de datos deben considerarse las primeras candidatas a ubicarse en las memorias más rápidas.

Debido a estas consideraciones la realización de mapeos óptimos en sistemas multinúcleo no resulta una tarea trivial, aumentando su complejidad al aumentar el número de núcleos para los que se desea paralelizar una aplicación. En este sentido la experiencia y los conocimientos del programador son claves para la realización de mapeos que permitan explotar al máximo el paralelismo de un sistema. A este tipo de mapeos se les denomina mapeos estáticos.

4.5.4. Mapeo dinámico de actores en plataformas multinúcleo

Recientemente se han desarrollado sistemas de mapeo dinámico, los cuales se sirven de distintos algoritmos que realizan distribuciones de actores a partir de medidas que se van almacenando en tiempo de ejecución. La base del funcionamiento es la toma de medidas de los parámetros que se identifican con los cuatro factores descritos en el apartado anterior. Así, cada cierto número de imágenes descodificadas un algoritmo propone un mapeo a partir de los datos registrados. Dado que el coste computacional de la ejecución del algoritmo no debe suponer una sobrecarga excesiva sobre el proceso de descodificación del vídeo, en la práctica, los algoritmos están diseñados para optimizar solamente alguno de los cuatro factores descritos.

El diseñador de la aplicación es el encargado de seleccionar los parámetros de configuración necesarios así como de elegir entre los algoritmos de mapeo dinámico disponibles en función del más ventajoso para la arquitectura objetivo. En el caso de los descodificadores basados en RVC y cuyo código fuente vaya a generarse con Orcc (ver capítulo 8) están disponibles distintos algoritmos de mapeo dinámico con distintos parámetros configurables [YCR+14]. La Tabla 8 muestra los distintos parámetros que se deben configurar antes de iniciar el proceso de descodificación para estas soluciones, y la Tabla 9 presenta los distintos algoritmos disponibles y el objetivo de su optimización.

Parámetro	Utilidad	Parámetro	Utilidad
-c	Número de procesadores disponibles	-s	Algoritmo a aplicar en mapeos dinámicos
-r	Número de imágenes a descodificar antes de realizar un mapeo	-a	Realizar mapeos dinámicos de forma indefinida cada r imágenes
-q	Guardar en un fichero el mapeo dinámico generado por el algoritmo	-p	Guardar en un fichero los datos de *profile* de una ejecución

Tabla 8. Relación de parámetros de configuración con mapeos dinámicos.

Id.	Significado	Objetivo del algoritmo	Coste computacional
RR	*Round Robin*	Mapeo simple RR	Bajo
WLB	*Weighted Load Balancing*	Optimizar la distribución de la carga de los actores	Medio
QM	*Quick Mapping*	Reducción de las comunicaciones manteniendo un coste computacional reducido	Muy bajo
MR	*METIS* [MET] *Recursive graph partition*	Realiza varias subdivisiones del grafo repartiendo la carga en cada una	Medio
MKCV	*METIS K-Way graph partition (Communication Volume)*	Optimizar el número de comunicaciones	Alto
MKEC	*METIS K-Way graph partition (Edge-Cut)*	Optimizar el peso de las comunicaciones	Alto

Tabla 9. Algoritmos de mapeo dinámico.

La Figura 26 resume el proceso de descodificación de vídeo en un entorno multinúcleo haciendo uso de las técnicas de mapeo dinámico descritas. En este caso el proceso de descodificación comienza leyendo los parámetros de configuración (*c*, *r*, *s* y *a*) y arrancando la descodificación con un único núcleo. Durante todo el proceso de descodificación se van almacenando medidas relativas a la carga computacional de los actores o la utilización de las FIFOs. Transcurridas un número *r* de imágenes descodificadas se calcula el mapeo en base al número de núcleos *c* y al algoritmo *s*, indicados al inicio y se reanuda la descodificación. Si se ha especificado el parámetro *a* al inicio se aplicarán sucesivos remapeos cada *r* imágenes descodificadas, en caso contrario la descodificación continuará de forma indefinida con el primer mapeo.

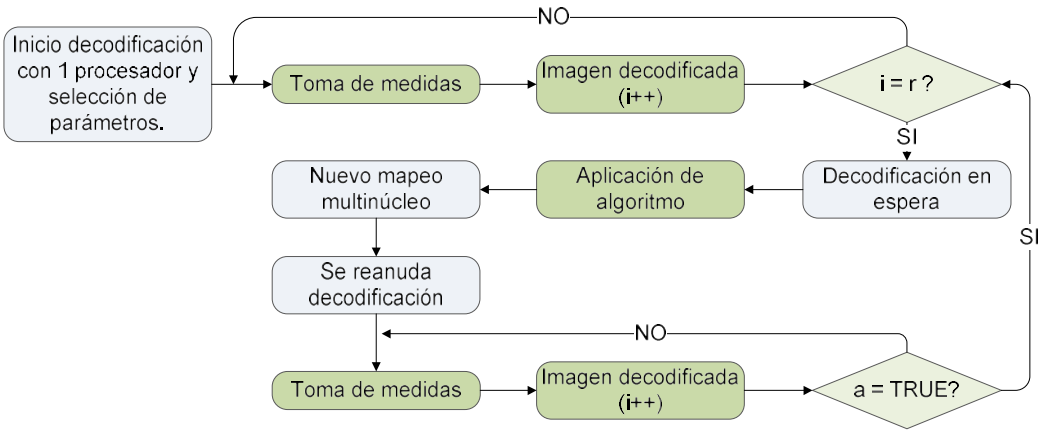


Figura 26. Diagrama del proceso de descodificación al utilizar mapeo dinámico.

Una desventaja adicional a la sobrecarga introducida al utilizar sistemas de mapeo dinámico es la necesidad de utilizar un hilo de ejecución extra encargado de realizar el procesamiento del algoritmo seleccionado; la tarea encargada de realizar dicho procesamiento es *agent_routine*. En sistemas de altas prestaciones esto no supone un problema ya que mientras la descodificación del vídeo está detenida[22] el SO permite la ejecución del hilo asociado al algoritmo en un mismo procesador. Sin embargo sobre plataformas Multi-DSP esto no es posible ya que el RTOS (ver capítulo 7) no es capaz de implementar esta funcionalidad. En el apartado 9.5 se detalla la integración de este tipo de sistemas en el trabajo de investigación realizado.

Cabe destacar que el hecho de indicar un número determinado de núcleos disponibles para el mapeo dinámico no implica que todos ellos vayan a ser utilizados en todo momento. Los algoritmos de distribución automática de la carga computacional pueden dejar temporalmente libre uno o varios de los núcleos indicados.

4.5.5. Limitaciones de la metodología

La principal limitación de los modelos DPN reside el comportamiento no determinístico de los actores que implementan la funcionalidad de las aplicaciones. Esto impide conocer por adelantado el comportamiento de la aplicación ya que éste depende de los datos que tenga que procesar. En sistemas estáticos, como SDF, es posible realizar distribuciones optimas de las unidades funcionales antes incluso de compilar el código, puesto que su comportamiento es completamente determinista (un ejemplo de modelo SDF es el que permite desarrollar la herramienta PREESM). En modelos DPN es necesario tomar decisiones en tiempo real, como los presentadas para la implementación de mapeos dinámicos (ver apartado 4.5.4).

4.6. Resumen

A lo largo de este capítulo se han presentado, en primer lugar, algunas de las metodologías desarrolladas recientemente en el campo de la programación de

[22] Para parar y sincronizar correctamente los procesos de descodificación y cálculo de mapeo se utilizan semáforos.

plataformas multinúcleo. En estas metodologías siempre se busca la abstracción de los detalles de bajo nivel de las arquitecturas objetivo, con el fin de proporcionar las herramientas que permitan al desarrollador explotar el paralelismo de las aplicaciones en entornos multinúcleo.

La metodología que se propone en esta tesis doctoral se apoya en el compilador Orcc (ver capítulo 8), una herramienta abierta de programación paralela basada en flujo de datos con un modelo de computación DPN. En el capítulo se discuten las ventajas de esta opción frente a otras en el contexto de la tesis. También se hace una descripción más detallada de las metodologías basadas en flujo de datos y particularmente del modelo de computación DPN.

TERCERA PARTE: Tecnologías de referencia

En esta tercera parte de la memoria se resumen las distintas tecnologías utilizadas en la fase de desarrollo de la tesis doctoral. Se introducen los elementos que las integran y los conceptos necesarios para la comprensión de los siguientes capítulos de la memoria.

El capítulo 5 presenta el *software* de descodificación de vídeo digital para el estándar HEVC denominado OpenHEVC. El capítulo 6 introduce los conceptos principales de la API de OpenMP que da soporte al desarrollo de aplicaciones en entornos multinúcleo. En el capítulo 7 se detallan las plataformas multinúcleo utilizadas durante el desarrollo de la tesis. Por último, el capítulo 8 presenta la infraestructura del compilador Orcc que materializa la metodología expuesta en la sección 4.5 y da soporte a las implementaciones basadas en el estándar RVC introducido en el apartado 2.3.

5. OpenHEVC

En este capítulo se introduce el *software* de descodificación OpenHEVC [OHE]. OpenHEVC es un *software* de código abierto conforme a la norma HEVC que está siendo desarrollado por distintos grupos de investigación y se encuentra disponible para la comunidad científica.

En el apartado 5.1 se introduce OpenHEVC, explicando brevemente sus orígenes y el entorno necesario para ponerlo en marcha. Seguidamente, en el apartado 5.2 se explica brevemente el funcionamiento del descodificador OpenHEVC. En el apartado 8.4 del capítulo 8 se explica cómo puede utilizarse el código de OpenHEVC en combinación con descripciones basadas en RVC-CAL para mejorar rendimiento final de los descodificadores cuyo código ha sido generado de forma automática.

5.1. El descodificador OpenHEVC

OpenHEVC es una implementación de un descodificador HEVC escrita en lenguaje C y su desarrollo forma parte del proyecto libre y con fines de investigación Libav [LAV]. El descodificador es compatible con 16 de los 21 perfiles de HEVC (hasta un muestreo 4:4:4 con 12 bits) e incluye en sus últimas versiones las extensiones SHEVC [ISO12] y MV-HEVC [TWCH+14]. El código que integra el descodificador está optimizado para GPPs de Intel [INT] o ARM [ARM]. El código de OpenHEVC se encuentra disponible en la plataforma de desarrollo colaborativo *GitHub* [OHEVC]. Además, OpenHEVC se utiliza en proyectos como 4EVER [4EV], H2B2VS [H2B] o VLC [VLC].

Como se ha mostrado en los apartados 3.2 y 3.3 existen diversas implementaciones basadas en este descodificador, las cuales han ofrecido buenos resultados en cuanto al rendimiento proporcionado sobre GPPs tipo Intel y ARM. El elevado rendimiento conseguido se debe principalmente al alto grado de optimización del código fuente del descodificador y a la utilización de instrucciones SIMD del repertorio SSE para la familia de procesadores x86 de Intel. Todos los ARMs con tecnología Neon (a partir de ARM Cortex 7) disponen de instrucciones SIMD, pero no el mismo repertorio SSE que x86, por lo que estas optimizaciones deben adaptarse al repertorio correspondiente del procesador.

El entorno de trabajo de OpenHEVC está preparado para que, tras la descarga del código del descodificador, el diseñador pueda hacer uso del mismo en un corto espacio de tiempo. Además de un compilador nativo a la plataforma donde se vaya a hacer funcionar el descodificador, serán necesarias las bibliotecas SDL[23] o SDL2 y la herramienta *yasm*[24]. El conjunto de archivos que componen el *software* de OpenHEVC incluye un fichero de configuración del entorno, el cual debe ser revisado, y en su caso modificado, antes de compilar el código. Mediante este fichero se puede realizar una configuración de prácticamente todos los aspectos que afectan tanto al descodificador como a la plataforma donde se vaya a probar el *software*.

Una vez las herramientas y extensiones están disponibles en la plataforma, y el código ha sido compilado se puede utilizar el descodificador HEVC. En la fecha de redacción de esta memoria el descodificador OpenHEVC soportaba las características descritas en la Tabla 10.

[23] SDL (*Simple DirectMedia Layer)*: Biblioteca que permite el acceso del *software* a recursos de bajo nivel como el audio y el *hardware* de gráficos para la reproducción del vídeo.
[24] La herramienta Yasm es un ensamblador para la arquitectura x86 de Intel.

Característica	Soporte en OpenHEVC
Perfiles	Desde *Main* hasta Main 4:4:4 12, y *Scalable Main*.
Configuración de la codificación (secuencias)	*All-Intra, Low-Delay* y *RandomAccess*
Procesamiento paralelo	*Tiles*, *Slices* y WPP.
Tamaño de Cus	64, 32, 16 y 8
Tamaño de Tus y Pus	32, 16, 8 y 4
Tamaño de Pus Inter	64, 32, 16 y 8

Tabla 10. Relación de características soportadas por OpenHEVC.

5.2. Funcionamiento de OpenHEVC

A continuación se realiza un breve resumen del funcionamiento del descodificador OpenHEVC. Dado el elevado número de archivos y funciones que conforman el descodificador aquí se presentará únicamente su estructura de alto nivel.

La descodificación de una secuencia de vídeo codificada conforme al estándar HEVC con el descodificador OpenHEVC comienza con la extracción de las cabeceras NAL de la trama. Una vez éstas han sido identificadas se inicia la descodificación de cada una de las imágenes codificadas en la secuencia. Cada vez que una imagen es descodificada se reinicia el proceso. La Figura 27 muestra el diagrama de flujo seguido durante la descodificación de las *slices* que compoenen las imágenes codificadas con el descodificador OpenHEVC, esta información ha sido extraída de [RNH+16], donde puede consultarse con mayor detalle.

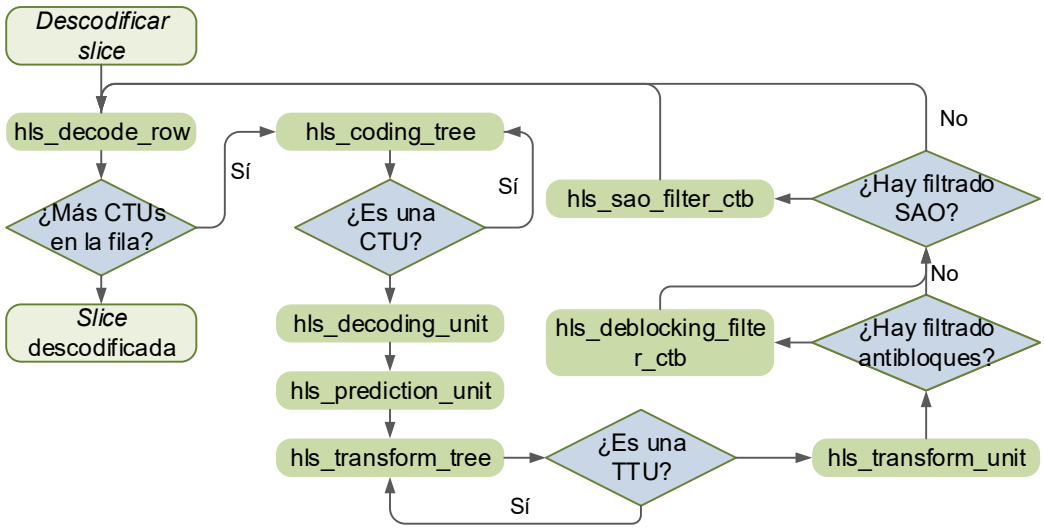


Figura 27. Diagrama del funcionamiento del descodificador OpenHEVC.

Para la descodificación de cada *slice* se llama de forma recursiva a la función *hls_decode_row,* que incia la descodificación a nivel de CTU. Seguidamente se extraen las CUs de cada CTU (*hls_coding_tree*), y se lleva a cabo la transformada inversa (*hls_transform*) y se realizan las predicciones correspondientes (intra/inter). Cuando todas las CUs de la CTU han sido descodificadas se aplican los filtrados antibloques

(*hls_deblocking_filter_ctb*) y SAO (*hls_sao_filter_ctb*), en caso de haberse utilizado en la codificación.

5.3. Resumen

En este capítulo se han resumido las principales características del *software* OpenHEVC, una implementación de código abierto en lenguaje C de un descodificador HEVC. El interés por este *software* reside en la posibilidad de incorporar secciones de código del mismo a las descripciones RVC-CAL de los descodificadores para generar código más eficiente.

6. OpenMP

En este capítulo se presenta la Interfaz de Programación de Aplicaciones (*Application Programming Interface* – API) de *Open Multi-Processing* (en adelante OpenMP). Esta ha sido una de las herramientas centrales en el desarrollo de la tesis doctoral, ya que mediante su uso se han conseguido implementar las distintas versiones de un descodificador HEVC sobre distintos tipos de plataformas, incluyendo una plataforma Multi-DSP. OpenMP permite la paralelización de aplicaciones escritas en C, C++ o Fortran en sistemas que disponen en su arquitectura de memoria compartida. Los conceptos de OpenMP que se van a describir son los más relevantes para esta tesis doctoral, para una consulta detallada sobre la API de OpenMP se recomienda acudir a la referencia [Ope03].

En el apartado 6.1 se introduce brevemente el origen histórico de OpenMP. En los apartados del 6.2 al 6.5 se presentan los fundamentos y elementos principales necesarios para trabajar con OpenMP. En el apartado 6.6 se realiza una breve discusión acerca de las ventajas e inconvenientes de utilizar esta herramienta. Finalmente, en el apartado 6.7, se discuten dos alternativas a la utilización de OpenMP sobre plataformas Multi-DSP.

6.1. Origen de OpenMP

La API de OpenMP [OMP] se crea en el año 1997 por parte de la OpenMP ARB (*Architecture Review Board*), la cual está formada por miembros de diversas empresas del sector de las telecomunicaciones y la electrónica como ARM, Fujitsu, IBM, Intel o Texas Instruments entre otros. La OpenMP ARB es una corporación sin ánimo de lucro que se encarga del mantenimiento y publicación de nuevas versiones de OpenMP, además de organizar las conferencias y eventos relacionados con el consorcio.

A finales de los años 80 surgen los sistemas SMP, (*Symmetric Multiprocessor System* – Sistemas Multiprocesador Simétricos) cuya arquitectura se basa en una memoria compartida por varios núcleos mediante un bus de datos común, con ello surgen los primeros problemas a la hora de gestionar un sistema multiprocesador. El programador de una aplicación que se ejecute en paralelo debe asegurarse de que el acceso a las variables compartidas por varios procesos en ejecución simultánea se realiza correctamente. Además, se deben tener en cuenta posibles diferencias en la sincronización del reloj del sistema de cada procesador, lo que provocaría diferencias en

la velocidad de ejecución por parte de cada núcleo (en la mayoría de sistemas multiprocesador actuales el reloj del sistema es común a todos los procesadores al encontrarse encapsulados en el mismo chip, evitando así este problema).

Las soluciones a estos problemas fueron una serie de instrucciones especiales, o directivas, que se podían añadir al código de los programas y que permitían al compilador generar el código de forma correcta para cada procesador. El problema radicaba en que cada fabricante desarrolló sus propias herramientas para dar solución a sus problemas particulares, no habiendo ningún tipo de estandarización. Con el ánimo de dar solución a esta situación surge el OpenMP ARB a comienzos de los años 90.

OpenMP es una herramienta que permite al programador abstraerse de la gestión de la sincronización y la concurrencia en un sistema multiprocesador mediante el uso de directivas o sencillas sentencias multiplataforma que describen el comportamiento del código paralelizado. Cabe destacar que OpenMP permite la ejecución simultánea de código ya existente mediante un proceso de paralelización incremental, de esta forma puede trabajarse de forma independiente sobre distintas secciones de código sin necesidad de abordar de una sola vez una modificación global.

OpenMP viene utilizándose en grandes proyectos como Altair RADIOSS, MATLAB (NaN y TSA *toolboxes*) o MACROS, todos ellos disponibles en el sitio web de OpenMP [OMP], entre otros. Además, distintos trabajos publicados en la literatura científica complementan el uso de esta API. Así, en [LKR10] investigadores de la Universidad de Pittsburgh presentan un estudio sobre la utilización de forma periódica de la técnica *fork/join*, y en [NMQC14] investigadores de la Universidad de Houston evalúan el impacto de la utilización de la API de OpenMP sobre el consumo energético sobre un procesador Intel.

A lo largo de los siguientes apartados se presentan los distintos elementos que se integran en la API de OpenMP, mediante los cuales se implementa la paralelización de secciones de código de una aplicación.

6.2. Ejecución *fork/join* de OpenMP

OpenMP está basado en el modelo de programación llamado *fork/join* (bifurcación/unión), como se ilustra en la Figura 28. Según este modelo la aplicación comienza con la ejecución de código secuencial por parte de un único hilo (hilo maestro). En cualquier momento en el que un hilo llega a una región de código a ejecutar de forma paralela se produce un *fork*, o bifurcación, inmediatamente después varios hilos (equipo de hilos – *team of threads*) ejecutan simultáneamente la región de código especificada en la región paralela, hasta que ésta termina y se produce el *join*, o unión.

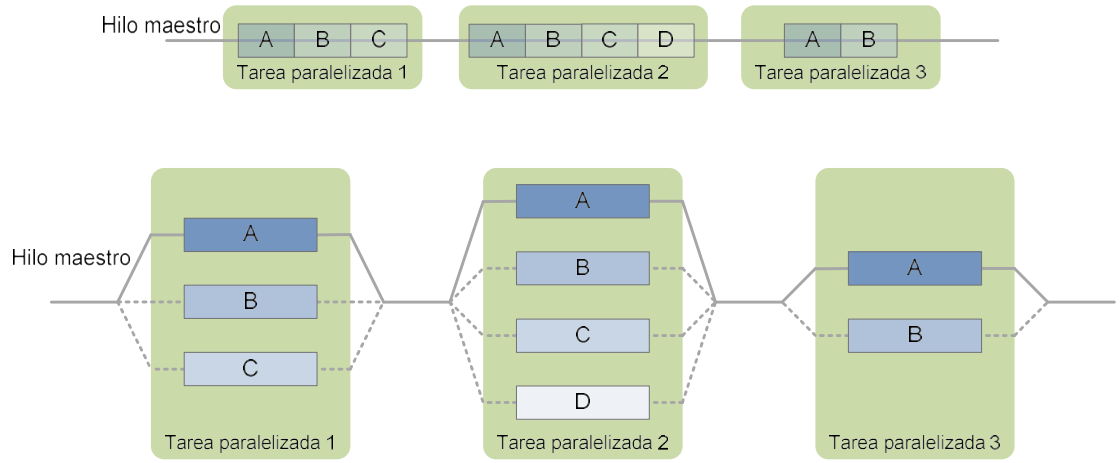


Figura 28. Diagrama de la ejecución *fork/join* de OpenMP.

Las regiones de código paralelizado deben identificarse y parametrizarse mediante el uso de las distintas directivas incluidas en la API. A continuación se describen los distintos componentes que forman parte de OpenMP.

6.3. Componentes de OpenMP

Existen tres componentes integrados en OpenMP que permiten al desarrollador realizar las descripciones de alto nivel deseadas. A continuación se enumeran todos ellos:

- Directivas: las directivas empleadas en OpenMP permiten básicamente definir el paralelismo de una aplicación. Las directivas deben incluirse en el código de una aplicación para que el compilador proceda a interpretarlas y generar el código para tantos hilos como se hayan definido previamente. En el caso de código escrito en C o C++, todas las directivas se expresan mediante *pragmas,* sentencias especiales que permiten controlar el comportamiento del compilador; naturalmente, es necesario que el compilador utilizado soporte OpenMP. Todas las directivas de OpenMP deben llevar el identificador *omp* tras el *pragma*, así como el nombre de la directiva y los parámetros asociados a ésta si los hubiese. Mediante las distintas directivas de OpenMP se pueden indicar las regiones de código a ejecutar en paralelo así como los parámetros necesarios que controlen su ejecución. La Figura 29 muestra el modelo de una directiva OpenMP en lenguaje C y en la siguiente subsección se listan las directivas de OpenMP. Además, el comportamiento de las directivas se puede complementar mediante el uso de cláusulas (ver apartado 6.5).

```
#pragma omp directiva [cláusula, cláusula [] ... ]
```

Figura 29. Modelo de directiva OpenMP.

- Funciones de biblioteca: las funciones de biblioteca son otro elemento a utilizar en una aplicación con código paralelizado mediante OpenMP. Mediante estas funciones se controlan parámetros tales como el número de hilos que OpenMP

puede gestionar o el control de ciertos elementos de sincronización como *mutex* (cerrojos).

- Variables de entorno: mediante las variables de entorno de OpenMP se puede controlar el comportamiento de las regiones paralelas en tiempo de ejecución, el tipo de planificación (*scheduler*), o ajustar dinámicamente el número de hilos al llegar a un *fork*. Cabe destacar que algunos de los parámetros introducidos como variables de entorno pueden ser modificados dinámicamente por ciertas funciones de la biblioteca.

6.4. Directivas y funciones de la biblioteca OpenMP

A continuación se presenta una relación de las principales directivas de OpenMP disponibles para la programación de aplicaciones en lenguaje C/C++.

A la hora de abordar la paralelización del código de una aplicación el primer paso es identificar aquellas secciones de código que van a ejecutarse en paralelo por parte de varios hilos, y qué secciones se ejecutarán en serie por parte de un único hilo. A aquellas secciones de código paralelizable se las denomina regiones paralelas y deben estar definidas por la directiva *parallel* de OpenMP. La Figura 30 muestra la declaración y ejecución de una región paralela mediante las correspondientes directivas de OpenMP en lenguaje C. En este ejemplo, tres núcleos ejecutarán, de forma simultánea, la sentencia *printf*, imprimiendo por pantalla el número de hilo en ejecución. Como se observa en el ejemplo, se deben indicar el número de hilos que van a realizar la ejecución de la región paralela. En el caso de la Figura 30, se hace uso de la función de biblioteca `omp_set_num_threads(n)` mediante la que se puede fijar este parámetro en tiempo de ejecución. Otra opción que ofrece OpenMP es crear una variable de entorno en el sistema denominada OMP_NUM_THREADS que contenga el valor de núcleos que se deseen utilizar en una región paralela.

```
Int main(void){
   omp_set_num_threads(3);      //Se define el número de hilos
   #pragma omp parallel         //Inicio de la región paralela
   {
      int n;                    //Variable privada
      n = omp_get_thread_num();
      printf("Hola mundo soy el hilo %d.\n",n);
      return 0;
   }                            //Fin de la región paralela
```

Salida por la terminal:

```
Hola mundo soy el hilo 0.
Hola mundo soy el hilo 1.
Hola mundo soy el hilo 2.
```

Figura 30. Ejemplo de declaración y ejecución de una región paralela con OpenMP.

Toda región paralela queda delimitada por llaves que deben abrirse inmediatamente después de la directiva *parallel* y, tras el cierre de las mismas, la ejecución continuará de forma secuencial en un solo hilo. Como se ha observado en el

ejemplo de la Figura 30, el código ejecutado por los tres hilos es el mismo, OpenMP permite identificar cada uno de los hilos que están ejecutándose en una región paralela mediante la función de biblioteca `omp_get_thread_num();`, la cual devuelve el identificador en una variable de tipo entero.

OpenMP ofrece tres directivas principales para gestionar la ejecución de una región paralela, éstas son *for, section* y *single.* La Tabla 11 muestra sus particularidades y un pequeño ejemplo que ilustra cómo se utilizan.

	For	***Section***	***Single***
Descripción	Permite paralelizar la ejecución de un bucle de tipo *for.*	OpenMP permite la ejecución simultánea de distintas secciones de código mediante esta construcción.	Dentro de una región paralela se puede indicar, mediante la directiva *single*, que cierta sección de código se ejecute por un único hilo
Limitaciones o consideraciones	Deben conocerse el número de iteraciones del bucle. No se permiten sentencias del tipo *break*, *continue* o *goto*.	Todos los hilos esperan al resto al terminar la ejecución de su sección. Si el número de secciones es mayor que el de núcleos, se fuerza la ejecución de varias secciones por un hilo.	OpenMP introduce una barrera de sincronización al final de la sección *single*. Existe una versión *master* de la directiva en la cual la sección *single* es ejecutada por el hilo maestro.

Ejemplo	`omp_set_num_threads(n);`
	`[...]`
	
	`#pragma omp parallel for`
	
	`for(i=0;i<Index; i++)`
	`var[i] = var[i] + 1;`
	
	`[...]`
		`#pragma omp parallel`
		`{`
		`#pragma omp sections`
		`{`
		`#pragma omp section`
		`[...]//Código hilo 1`
		`#pragma omp section`
		`[...]//Código hilo 2`
		`#pragma omp section`
		`[...]//Código hilo n`
		
		`}//Fin de las secciones`
		`}//Fin de región paralela`
			`#pragma omp parallel`
			`{`
			`#pragma omp single`
			`{[...]//Código ejecutado de forma serializada`
			`}`
			`[...]//Código ejecutado por varios hilos`
			
			`}//Fin de región paralela`

Tabla 11. Directivas de OpenMP para la gestión de regiones paralelas.

Sin embargo, las posibles dependencias de datos (por ejemplo en la ejecución de un bucle *for*) en el acceso a variables compartidas dentro de una región paralela será responsabilidad del programador.

Finalmente, cabe destacar que por defecto todas las variables declaradas antes del comienzo de una región paralela serán tratadas como variables compartidas por parte de todos los hilos, y serán alojadas en memoria compartida. Así mismo, todas las variables declaradas dentro de una región paralela quedarán definidas por defecto como privadas, en este caso cada hilo tendrá una copia local de la variable no accesible por parte del resto de hilos, generalmente estas variables quedarán alojadas en la memoria local de cada núcleo. Este comportamiento por defecto puede verse modificado mediante el uso de cláusulas en las directivas.

6.5. Cláusulas fundamentales

Como ya se introdujo en el apartado 6.3, las directivas de OpenMP pueden acompañarse de una serie de cláusulas que complementan su funcionalidad. En este sentido se han definido tres grupos de cláusulas: aquellas que afectan a las variables, ya sean compartidas o privadas, las que se ocupan de la sincronización y por último las que complementan la planificación de las regiones paralelas. La Tabla 12 ordena y resume la utilidad de las cláusulas fundamentales disponibles en OpenMP. En [OMPb] se pueden encontrar varios ejemplos de utilización de cláusulas fundamentales, así como una explicación más pormenorizada.

Cláusulas que afectan al uso de variables	
`Shared`	Los datos declarados en la región paralela estarán considerados como compartidos y serán accesibles por parte del resto de hilos.
`Private`	Las variables declaradas en la región paralela se consideran privadas, no son inicializadas y el valor de las mismas se pierde al salir de la región paralela.
`Default`	Permite cambiar la configuración por defecto en la región paralela.
`Firstprivate`	La variable privada toma el valor que tuviera ésta antes de la región paralela.
`Lastprivate`	El valor de una variable privada se conserva al terminar la región paralela.
Cláusulas de sincronización	
`Critical`	Garantiza que varios hilos no accedan de forma simultánea a una variable.
`Atomic`	La modificación de variables compartidas se realiza de forma atómica por parte de varios hilos.
`Ordered`	La ejecución de la sección se hará en orden, resulta útil al imprimir datos por pantalla.
`Barrier`	Los hilos del equipo esperan en la barrera (*barrier*) a que todos los hilos lleguen a ese punto ante de continuar su ejecución.
`Nowait`	Al final de la sección paralela los hilos no se esperan a que el resto finalicen.
Cláusulas de planificación con bucles (*scheduling*)	
`Static`	Las iteraciones se dividen en secciones que se asignan estáticamente a hilos de ejecución.
`Dynamic`	Las iteraciones son ejecutadas dinámicamente por los hilos disponibles.
`Guided`	Similar a *dynamic*, el número de iteraciones pendientes va disminuyendo dinámicamente.

Tabla 12. Cláusulas fundamentales de OpenMP.

6.6. Ventajas e inconvenientes de la utilización de OpenMP

La principal ventaja de OpenMP es la facilidad con la que pueden paralelizarse con ella secciones de código ya escrito; esta paralelización puede llevarse a cabo, además, de forma incremental. Estas características permiten flexibilizar la migración de código serie a entornos multinúcleo. Además, la estructuración en una serie directivas para la gestión de las regiones paralelas permite a los desarrolladores sin experiencia llevar a cabo un rápido aprendizaje que les posibilita centrar los esfuerzos de desarrollo en la mejora del paralelismo de aquellas secciones de código especialmente sensibles. Por último, OpenMP permite al programador abstraerse, en la mayoría de los casos, de los problemas de gestión de la sincronización y acceso a memoria compartida, facilitando en caso de ser necesario las instrucciones necesarias para solventar los inconvenientes que puedan surgir.

Por el contrario no todas las distribuciones de la API de OpenMP son idénticas entre sí, dependiendo en gran medida de las implementaciones que los fabricantes de chips llevan a cabo para sus plataformas. Si bien todas las distribuciones están sujetas a las normas especificadas para cada versión de la API, el rendimiento de las aplicaciones puede verse afectado por posibles sobrecargas al utilizar OpenMP[25]. Otra desventaja de OpenMP es que no se encuentra disponible para todas los tipos de procesadores, como por ejemplo sucede con aquellos sin memoria compartida en su arquitectura. Sin embargo su uso está muy extendido y se encuentra disponible para casi todos los procesadores multinúcleo con memoria compartida.

Resulta de especial importancia prestar atención a las distintas versiones de OpenMP, ya que no todas están disponibles para cada plataforma o fabricante; además las posibilidades ofrecidas en cada versión pueden variar de forma significativa[26]. La Tabla 13 resume las ventajas y desventajas de utilizar la API de OpenMP.

Ventajas	Desventajas
Código paralelizado fácilmente *migrable*	Riesgo de introducir situaciones de difícil depuración
Paralelismo incremental	El compilador debe soportar OpenMP
Permite la paralelización de alto y bajo nivel de granularidad	Mayor dificultad al definir variables según sean privadas o compartidas
Compatible con diversos aceleradores	Se requiere una plataforma con memoria compartida
Código unificado tanto para regiones paralelas como de ejecución serie.	Riesgo al gestionar regiones de código compartido o privado
El código de ejecución serie no requiere modificarse	Compleja asignación de hilos a procesadores en secciones con alto grado de granularidad

Tabla 13. Relación de pros y contras al utilizar OpenMP.

Teniendo en cuenta la discusión anterior, la utilización de OpenMP para la paralización del código de un descodificador de vídeo basado en RVC resulta acertada, siempre y cuando OpenMP esté disponible para la arquitectura objetivo, dado que: (1) se pueden realizar modificaciones incrementales sobre el código, considerablemente extenso y complejo, que integra el descodificador; (2) se puede realizar una paralelización de alto nivel, en el caso de RVC a nivel de actores o unidades funcionales; (3) no es necesario modificar el código interno de una región paralela o que se vaya a ejecutar en serie, (4) el mismo código con pequeñas modificaciones puede utilizarse sobre distintas plataformas multinúcleo, y (5) se puede asignar dinámicamente el número de núcleos a utilizar.

[25] La primera versión de OpenMP para plataformas Multi-DSP de Texas Instruments introducía una sobrecarga cercana al 90%. Sin embargo en posteriores versiones dicha sobrecarga se ha visto reducida hasta aproximadamente un 10%.

[26] La versión v4.0 de OpenMP incluye facilidades para la paralelización de código sobre ciertas plataformas heterogéneas.

6.7. Alternativas a OpenMP sobre plataformas multi-núcleo

Existen diversas herramientas alternativas a OpenMP que facilitan la programación de aplicaciones cuyo código es susceptible de paralelizarse. Sin embargo, la mayoría son específicas de cada fabricante, su funcionalidad está limitada a unas pocas arquitecturas y/o su desarrollo no está soportado por un consorcio inter-industrial abierto, haciendo difícil el acceso al código fuente de las APIs. A continuación se presentan dos alternativas válidas para plataformas Multi-DSP.

6.7.1. Desarrollo *ad-hoc*: gestión de la memoria y tareas

De forma paralela a APIs como OpenMP, el fabricante Texas Instruments permite la creación, gestión y sincronización de tareas y recursos sobre el SYS/BIOS (o kernel RTOS, ver apartado 7.2), y más concretamente sobre las últimas plataformas Multi-DSP homogéneas de la familia C66 [C66]. El fabricante facilita mediante diversos documentos técnicos las explicaciones y ejemplos necesarios para abordar el diseño de aplicaciones sobre plataformas multinúcleo. Así, en [SPR3P] se facilita la guía de usuario de SYS/BIOS, la cual describe todos los detalles sobre la gestión de recursos del RTOS y de la plataforma, en [TIWP] se perfilan técnicas para reducir el tiempo de desarrollo de aplicaciones multinúcleo con dispositivos KeyStone de Texas Instruments (ver apartado 7.1.2). Por último, en [SPR0C] se detallan las características de los distintos niveles de memoria y de los diferentes periféricos que soporta el chip.

En el caso de elegir esta vía de desarrollo el programador debe diseñar de forma manual e individualizada a cada caso, toda la gestión necesaria de todos los recursos de la plataforma. Es por esto que esta alternativa requiere de un elevado conocimiento de las particularidades de las herramientas propias del fabricante; cabe destacar que, aún en este caso, el tiempo de desarrollo puede ser elevado, y de forma general no se recomienda el uso de esta metodología salvo en aquellas aplicaciones de menor tamaño o complejidad.

Esta alternativa no permite por tanto soportar una solución metodológica en sí. Sin embargo, puede resultar de interés para llevar a cabo pruebas iniciales de estudios en desarrollo, de verificación de ciertos principios de aplicaciones de pequeña complejidad, o en aquellas en las que desee tener un control exhaustivo de todos los aspectos que controlan el comportamiento de bajo nivel del RTOS.

6.7.2. OpenCL

Open Computing Language (OpenCL) [OCL] es una herramienta integrada por una API y un lenguaje de programación específico llamado OpenCL C, la cual fue originalmente diseñada para facilitar la paralelización de aplicaciones sobre Unidades de Procesamiento Gráfico (GPUs) y Unidades Centrales de Procesamiento (CPUs). OpenCL fue estandarizado en 2013 por el consorcio internacional Khronos[27] [KHR], si bien fue inicialmente impulsado por Apple y desarrollado de forma conjunta por

[27] El grupo Khronos dispone de distintos grupos de trabajo para la creación de APIs, entre los cuales está OpenCL.

grandes grupos de la industria como IBM, Intel o NVIDIA. Cabe destacar que el modelo de programación de OpenCL está basado en el *framework* CUDA [CUDA] de NVIDIA.

El funcionamiento de OpenCL se basa en un *host* (anfitrión) que despacha tareas a otras unidades de procesamiento, bien porque éstas pueden ejecutar los algoritmos de forma más eficiente que el propio *host* o bien porque se desea paralelizar una serie de instrucciones. A estas unidades de procesamiento se las denomina aceleradores.

Como se puede observar en la Figura 31, los aceleradores utilizados bajo OpenCL están ubicados en una estructura jerárquica controlada por el *host*, el cual, utilizando instrucciones propias de la API de OpenCL, despacha tareas a cada uno de ellos (*Compute Device*, según la nomenclatura OpenCL). Éstos a su vez están compuestos de uno o varios clústers (*Compute Unit* - CU), los cuales ejecutarán las tareas paralelizadas en los elementos de procesamiento (*Processing Element*) de que dispongan. Uno o varios dispositivos aceleradores pueden agruparse en lo que, en OpenCL, se denomina un *contexto*. En función del entorno de desarrollo habrá un determinado número de contextos, pudiendo estar todos ellos controlados por un único *host*. Esto permite utilizar distintas arquitecturas dentro de un mismo entorno, pudiendo aprovechar así las ventajas que ofrecen cada una de ellas.

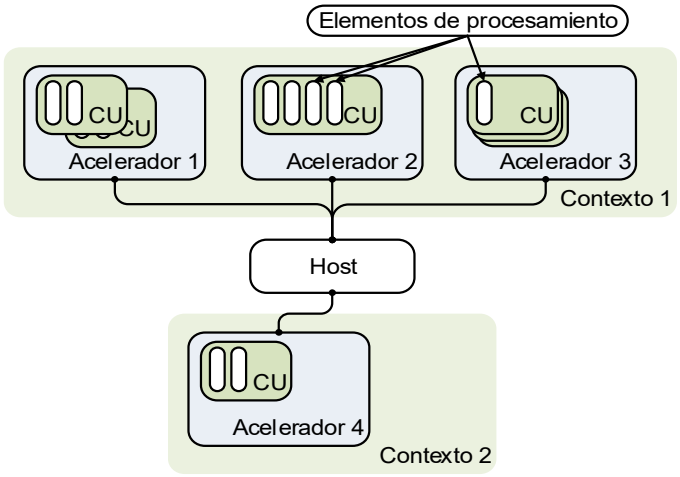


Figura 31. Diagrama de la distribución jerárquica con OpenCL.

De esta forma OpenCL permite gestionar sistemas compuestos de distintos tipos de procesadores, dentro de plataformas heterogéneas (ver apartado 7.4). En este tipo de sistemas las características de cada procesador pueden ser muy diferentes, por lo que OpenCL permite realizar una gestión individualizada de cada dispositivo. Esto se logra utilizando distintas versiones del *framework* OpenCL sobre los diferentes dispositivos, atendiendo así a las funcionalidades específicas de cada uno de ellos.

La herramienta OpenCL es relativamente novedosa, habiéndose iniciado su utilización en entornos Multi-DSP durante el desarrollo de esta tesis doctoral. A día de hoy, su integración con la metodología de diseño propuesta en esta tesis está dificultada por su modelo de reparto de tareas: OpenCL está diseñado para despachar *paquetes de*

trabajo o secciones de código bien definidas a los dispositivos aceleradores para que éstos los procesen, especificando para ello la cantidad de datos transferidos. Por lo tanto la posibilidad de gestionar la paralelización de la ejecución de los actores RVC, cuya ejecución es cíclica en base a su gestión mediante planificadores bien especificados, queda descartada por el momento.

6.8. Resumen

En este capítulo se ha presentado la API de OpenMP, mediante la cual es posible paralelizar código en lenguaje C para procesadores multinúcleo con arquitectura de memoria compartida.

La principal ventaja de OpenMP frente a otras alternativas es su fácil integración en un código ya escrito y aún no paralelizado. Se ha detallado cómo OpenMP permite llevar a cabo una paralelización del código de forma estructurada y modular.

Además, se han presentado las principales características y sentencias de programación que integran la API de OpenMP y que se han utilizado durante la fase de desarrollo de esta tesis doctoral.

Por último se han presentado dos alternativas a uso de OpenMP para la paralelización del código en arquitecturas multinúcleo.