Universidad Zaragoza
1542

# Proyecto Fin de Carrera

## Migración y optimización de la aplicación VisionSurfer sobre plataforma Intel® 64 bits


Autor:

José Ángel Gariburo Cortés

Director:

Óscar Puyal Latorre

Ponente:

José Luis Briz Velasco

Escuela de Arquitectura e Ingeniería


2013

# Agradecimientos

Son muchas las personas que han hecho posible que yo haya llegado hasta aquí. Nombrarlas a todas sería muy difícil, así que me resignaré a nombrar a algunas de las más destacables.

Gracias a mis padres, ellos siempre han estado ahí y siempre lo estarán por muy difícil que sea la empresa en la que me aventure. Ellos me han hecho ser mejor persona día a día.

Gracias al resto de mi familia porque siempre me han dado todo su apoyo y cariño.

Gracias a mis amigos por aguantarme todos estos años, con lo bueno y con lo malo, sin ellos esto tampoco habría sido posible.

Gracias a todos los profesores que tanto me han enseñado durante todos estos años de estudio, enseñándome a mirar con otros ojos al mundo.

Gracias a toda la gente de SCATI LABS. que me ha ayudado en todo lo posible durante este período.

Gracias a los amigos que hice durante mi estancia en Dinamarca por darme otro punto de vista sobre la vida.

# Migración y optimización de la aplicación VisionSurfer sobre plataforma Intel® 64 bits

## RESUMEN


En este Proyecto de Fin de Carrera se ha realizado un estudio sobre la viabilidad y conveniencia de la migración de la aplicación comercial de manejo de vídeo *VisionSurfer* de su actual plataforma de 32 bits (IA32) a una de 64 (Intel® 64).

Esta aplicación, gracias a su sistema *"Cluster"* de almacenamiento, permite al usuario grabar la información en todos los discos duros repartidos entre los servidores y así garantizar la disponibilidad del video grabado ante fallos de discos duros e incluso de un servidor completo.

Por otro lado dispone de un sistema inteligente gracias a la virtualización, denominado *Failover* que, detecta la caída de aplicaciones de alguno de los equipos y se pone en marcha automáticamente en otro servidor para garantizar la continuidad de servicio.

En primer lugar se ha estudiado la arquitectura conocida como Intel® 64 y los posibles problemas surgidos al migrar desde IA32. Se han comprobado la disponibilidad y correcto funcionamiento en 64 bits de las librerías usadas en 32 bits, y se han buscado y comparado herramientas de análisis estático de código que nos puedan ayudar en esta tarea.

Con los resultados obtenidos se ha hecho una primera estimación del coste temporal aproximado de la migración total de la aplicación. Y se ha desarrollado una aplicación de ejemplo que sirva para entender y ejemplificar la aplicación objetivo. Debido a la extensión y complejidad de la aplicación *VisionSurfer*, finalmente se ha procedido a la migración de una aplicación crucial dentro de esta llamada *SCATIRTPVideoClient.*

A continuación se han evaluado las posibilidades de mejora del rendimiento en la aplicación. Para ello se han buscado cuellos de botella, causados por el uso intensivo de memoria, detección de posibles fugas de memoria y posibilidad de vectorización.

Por último se han realizado pruebas de aceptación y evaluación del rendimiento.

# Índice

# Índice de figuras

# Capítulo 1

## 1. INTRODUCCIÓN

En un cajero automático se produce un incidente. Una persona se lleva el dinero mientras se realiza el proceso de carga del dispensador, aprovechando un fallo en el protocolo de seguridad de la entidad bancaria. Cuando el delincuente llega a su casa la Policía ya le estaba esperando. Es un caso verídico que muestra la utilidad y eficiencia de los sistemas de video vigilancia.

Cada día vivimos en un mundo más globalizado, donde las personas pueden moverse de una punta a otra del planeta en un mismo día. Esto ha hecho que la video vigilancia cobre un mayor protagonismo, permitiendo compartir casi instantáneamente la información de un sospechoso a lo largo de todo el mundo.

Como consecuencia de la cantidad creciente de cámaras, el número de dispositivos físicos necesarios para gestionarlas ha aumentado.

Este proyecto nace del deseo de reducir este número aprovechando del mayor rendimiento de las arquitecturas de 64 bits sobre las actuales de 32 bits. Reutilizando el software ya desarrollado y mejorando en lo posible su rendimiento aprovechando las características de paralelización y vectorización que ofrecen los nuevos procesadores. De esta manera se espera conseguir gestionar un número mayor de cámaras desde un mismo dispositivo.

El propósito de este proyecto es, por tanto, el estudio de la viabilidad y conveniencia de esta migración, incluyendo la estimación del coste temporal de la misma y del aumento de rendimiento esperado.

### 1.1. MOTIVACIÓN

A lo largo de la carrera he estudiado asignaturas en las que se nos ha mostrado y enseñado como poder mejorar el rendimiento de las aplicaciones que creábamos con cambios relacionados con las herramientas usadas, y la teoría que habíamos aprendido sobre la arquitectura del procesador para el que estábamos programando en ese momento. Son cambios que en muchas ocasiones no se aplican sobre software comercial debido o al desconocimiento de las herramientas o a las fechas de entrega, normalmente superadas, y que hacen que el software no tenga toda la calidad que podría esperarse.

Algo tan simple como un cambio en las opciones del compilador, puede hacer que consigamos una mejora evidente del rendimiento de nuestra aplicación. ¿Qué

podríamos conseguir si además de conocer el compilador, fuésemos capaces de aprovechar las características que nos ofrecen los procesadores de hoy en día?

## 1.2. OBJETIVO Y ALCANCE

Uno de los objetivos es estimar el coste temporal de la migración a 64 bits de la aplicación *VisionSurfer*. Para ello se requiere de un estudio de las arquitecturas de 64 bits y de los problemas más comunes incurridos al migrar una aplicación de 32 a 64 bits, además de la necesidad de búsqueda y verificación del correcto funcionamiento de las librerías anteriormente usadas en 32 bits. Otro objetivo es la búsqueda de herramientas de análisis estático de código que nos ayuden en este proceso.

Con toda esta información se realizará una estimación a partir de una aplicación representativa de la aplicación final. Una vez realizada la estimación, se procederá al análisis de posibles mejoras a realizar sobre la aplicación, para obtener un mayor rendimiento. Estas mejoras consistirán en la vectorización y paralelización, si es posible, de los algoritmos que ocupen la mayor parte del tiempo de ejecución de la aplicación que, una vez optimizados, nos ofrezcan una mejor evidente.

Por último se realizarán las pruebas de aceptación y rendimiento necesarias.

## 1.3. MÉTODOS Y TÉCNICAS

El código desarrollado en este proyecto será enteramente C++. El entorno de desarrollo ha sido Visual Studio 2008, también se han usado las librerías de tratamiento de vídeos de Intel IPP (Integrated Performance Primitives) que consisten en un conjunto de funciones optimizadas para los procesadores de Intel. Además se han considerado librerías de uso gratuito como pueden ser ACE (Adaptive Communication Environment), Cryotpp, etc.

## 1.4. CALENDARIO DEL PROYECTO

La figura 1 muestra el diagrama de Gantt del proyecto.

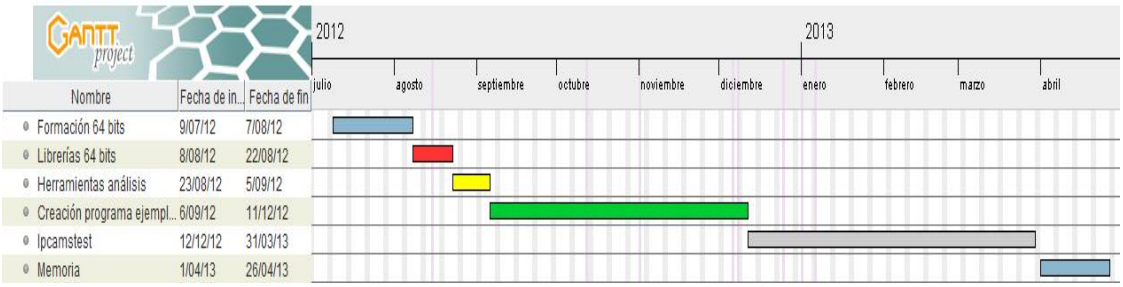



**Figura 1. Planificación del Proyecto.**

Este proyecto se ha dividido en 6 fases:

1. ***Estudio de la arquitectura Intel 64 bits:*** Búsqueda y estudio de la documentación. Será importante detectar los problemas más comunes presentes en las migraciones de 32 a 64 bits que aparecen en la literatura. Conocer tanto las mejoras que podamos obtener con este cambio, (velocidad de procesamiento, manejo de memoria, etc) como las posibles desventajas (incremento del tamaño de código). Realización de pruebas que demuestre estos cambios. Duración estimada: un mes, 110 horas.

2. ***Librerías 64 bits:*** Identificación de las librerías usadas en la versión de 32 bits y de las correspondientes en 64 bits. Comparación de las mismas mediante su análisis estático, ya que puede no ser necesario el cambio de versión, si los cambios presentes en la nueva versión de la librería no nos aportan nada nuevo. Dedicación estimada: tres semanas, 75 horas.

3. ***Herramientas de análisis:*** Selección de herramientas de análisis estático de código, que nos ayuden en el proceso de identificación de posibles errores en nuestro código a la hora de realizar la migración de 64 bits. Duración estimada: dos semanas, 50 horas.

4. ***Creación del programa ejemplo para las pruebas:*** Realización de los cambios pertinentes en las IPP para su adaptación a la librería IPPMedia. Programación, verificación y pruebas de rendimiento del codificador-decodificador. Duración estimada: tres meses, 330 horas.

5. ***Ipcamtest***. Tras haber creado el programa de ejemplo para 64 bits, procederemos a realizar la migración de la aplicación *`Ipcamtest`*. Esta aplicación consiste en un cliente que se conecta a un servidor de vídeos. Esta aplicación los puede comprimir, mostrar por pantalla, guardar, etc. Además puede hacer esto para todo el vídeo o solo para los *`keyframes`*. También se

realizarán las pruebas de rendimiento pertinentes para la comparación entre versiones. Duración estimada: tres meses y dos semanas, 380 horas.

6. ***Memoria***. Realización de la memoria del proyecto. Duración estimada: un mes, 110 horas.

# Capítulo 2

## 2. Formación

Como paso previo a la migración de una aplicación a una arquitectura de 64 bits, este proyecto abarca la formación necesaria para el conocimiento de las características que diferencian una arquitectura de 32 bits, a la que estamos más habituados, frente a una arquitectura de 64 bits, que pese a ser la predominante hoy en día en la mayoría de ordenadores, convive con aplicaciones de 32 bits. Por razones de compatibilidad se siguen usando, provocando un gran desaprovechamiento de estas arquitecturas más nuevas y a priori mejores, ya que como veremos aportan una serie de herramientas adicionales que bien aprovechadas mejoran el rendimiento.

En nuestro caso, la arquitectura en la que nos vamos a basar es la definida por *Intel*, comúnmente conocida como x86-64 o x64. Es curiosos que aunque fuese *AMD* la primera en crear la especificación, no triunfara tanto como *Intel* que pese a llegar algo más tarde, 1 año aproximadamente, tiene una cuota de mercado mayor. Además Intel cuenta con su otra arquitectura de 64 bits, conocida como *Itanium* IA64 y que a diferencia de x86-64, no mantiene compatibilidad con arquitecturas previas de 16 y 32 bits, pero nosotros no vamos a tratar este tema ya que esta arquitectura está enfocada a servidores y alto rendimiento.

Las características que diferencian una arquitectura de 64 bits de una de 32 son:

- Los registros del procesador como *rax, rbx*, etc son ahora de 64 bits.
- Operaciones aritméticas y lógicas de 64 bits. Todos los registros de propósito general (*GPRs General Purpose Registers*) se han extendido a 64 bits. Ejemplos de operaciones que usan registros de 64 btis ahora son: memoria-registro, registro-memoria, *push* y *pop* en la pila, todos los espacios reservados ahora en esta son de 8 bytes, 64 bits y los punteros también han pasado de 4 bytes, 32 bits a 8 bytes, 64 bits.
- Direccionamiento de 64 bits, es decir podemos tener una memoria virtual teórica de 2^64, frente a los 2^32, 4GB que tenemos con las arquitecturas de 32 bits y que actualmente, debido a las memorias RAM se queda "corto" puesto que podemos tener un ordenador relativamente barato con más de 4GB, por lo que estaríamos desaprovechando todo ese espacio de memoria. pese a esto, en las implementaciones actuales sólo se usan 48 bits de los 64 disponibles para direcciones de memoria virtual, lo que supone hasta 256TB, 2^48, una cantidad de memoria que está muy lejos de alcanzarse por el momento, además siempre tendríamos más bits disponibles ya que este no es el límite máximo.

- Se han añadido ocho registros nuevos además de la correspondiente actualización a 64 bits del tamaño de todos los registros anteriormente existentes en 32 bits. Los registros añadidos han sido: *r8, r9, r10, r11, r12, r13, r14, r15*. Esto hace que podamos tener más variables en registros en lugar de en la pila, con el aumento de velocidad que esto supone al ser los registros más rápidos en su acceso que la pila. También en pequeñas subrutinas se pueden pasar los parámetros en registros en lugar de en la pila.
- También se han incrementado los registros XMM (SSE), que son de 128-bits y que se usan para instrucciones SIMD, de ocho a dieciséis.
- Ahora las instrucciones pueden hacer referencia a datos relativos al registro *RIP (relative instruction pointer)*. Esto hace el código independiente de su posición, usado en librerías compartidas y en código cargado en tiempo de ejecución, más eficiente.
- La arquitectura de 64 bits original, AMD64, adoptó los repertorios SSE y SSE2 como el conjunto de instrucciones básicas. SSE3 fue añadido en el año 2005. SSE2 es una alternativa al conjunto de instrucciones de x87, *IEEE 80-bit precision* con la elección de IEEE 32-bit o 64-bit para las operaciones matemáticas en punto flotante. Esto hace que sea compatible con muchas de las CPUs modernas. SSE y SSE2 están disponibles sólo en los procesadores modernos de 32-bits. Esto en 64-bits no pasa, puesto que están presentes en todos ellos.
- El bit de no ejecución *NX bit (**No-Execute bit***, bit número 63 de la página de la tabla de entrada) permite al sistema operativo especificar que páginas del espacio de direcciones virtuales pueden contener código ejecutable y cuáles no. De esta forma, intentar ejecutar código de una página marcada como no ejecutable producirá una violación de acceso de memoria, parecido a si intentásemos escribir en una página de sólo lectura. Esto debería hacer más difícil al código malicioso controlar el sistema mediante ataques del tipo buffer sin comprobar o *buffer overrun*. Una característica parecida a esta ha estado disponible en los procesadores x86 desde el 80286 como un atributo de los descriptores de segmento, aunque sólo funciona en un segmento cada vez. Debido a que el direccionamiento segmentado se ha considerado desde hace mucho tiempo una forma obsoleta de operar, y todos los PCs actuales se saltan este modo.
- Por último también se han eliminado ciertas características de la arquitectura x86. Esto incluye el direccionamiento segmentado, como acabamos de comentar (aunque los segmentos FS y GS se mantienen para su uso como punteros base extra para las estructuras del sistema operativo), el mecanismo para el cambio de estado de una tarea (es una estructura especial que contiene información de una tarea. La usa el sistema operativo para el manejo de tareas) y el modo virtual de 8086. Estas características permanecen sólo en modo legado, lo que permite que en este modo se puedan ejecutar sistemas operativos de 32 y 16 bits sin hacer modificaciones.

Para finalizar esta introducción a las arquitecturas de 64 bits, cabe destacar los problemas más comunes que nos podemos encontrar a la hora de realizar una migración de 32 a 64 bits. Además de estos problemas propios del cambio de arquitectura, existen otros problemas como pueden ser los cambios de versión en las librerías, trabajar con código legado, etc. Estos problemas no los vamos a comentar por ahora, pese a que han surgido a lo largo de este proyecto y quizás hayan sido igual o más importantes que la migración propiamente dicha.

Algunos de los problemas más comunes a la hora de migrar un programa de una arquitectura de 32 bits a una de 64 bits son:

- Almacenamiento de punteros en variables de tipo entero.
- Aritmética de punteros.
- Operaciones de desplazamiento.
- Alineamiento de las estructuras de datos.
- Y otros muchos.

Más adelante veremos la explicación de casi todos ellos y ejemplos que ilustren el problema para un mejor entendimiento de este.

## *2.1. Primer contacto*

Al igual que hacemos cuando empezamos a programa en un lenguaje nuevo, para tener una primera aproximación a lo que puede significar el cambio de 32 a 64 bits en nuestra aplicación, hemos compilado el programa más conocido: *`Hello World`* en C++ tanto para 32 bits como para 64 bits.

Las opciones de compilación usadas para generar los ficheros ensamblador de la aplicación *`Hello World`* han sido:

- *`-Od`* en Debug.
- *`-O2 -Oi`* en la versión Release.

Mayores parámetros de optimización como *`-Ox`* no han producido mejoras respecto a la optimización por defecto *`-O2`*

El enlazado en Windows de las librerías por defecto de C++ es dinámico, y no puede realizarse de manera estática. Aunque si podemos enlazar estáticamente librerías no estándar de C++ o Windows.

El código ensamblador ha sido generado con: *`Listing generated by Microsoft (R) Optimizing Compiler Version 15.00.21022.08`*.

El examen del código ensamblador para 32 y 64 bits que se encuentra en el ANEXO 1, permite apreciar que ambos tienen 10 instrucciones para la versión

*Release*, pero repartidas de forma diferente. Observamos que empleamos más instrucciones en la creación y destrucción del bloque de activación en 64 bits y menos en las instrucciones relacionadas con la ejecución de la lógica del programa. En la versión *Debug*, sí que obtendríamos una reducción del tamaño de código general para la versión de 64 bits.

| **HELLO WORLD** | | |
|---|---|---|
| **Nº Instrucciones** | **32 bits** | **64 bits** |
| *Creación bloque de activación* | 0 | 1 |
| *Cuerpo* | 8 | 6 |
| *Destrucción bloque de activación* | 2 | 3 |
| **TOTAL** | 10 | 10 |

**Figura 2. Número de líneas ensamblador en 32 y 64 bits.**

La *Application Binary Interface (ABI)* para la arquitectura x64 en Windows, pasa los primeros cuatro argumentos en la llamada a una función usando registros, y reservando espacio para estos cuatro valores en la pila también. Cualquier argumento que no tenga un tamaño de 1, 2, 4 u 8 bytes se pasa a la función por referencia.

Estos cuatro primeros argumentos se pasan usando los registros *RCX, RDX, R8 y R9*, estos argumentos se alinean a la derecha en los registros, permitiendo ignorar los bits más significativos si es necesario. Si los argumentos son *float/double* se pasan usando *XMM0L, XMM1L, XMM2L y XMM3L*.

El proceso que llama a la función es el responsable de reservar espacio para los parámetros de la función que es llamada, y debe siempre reservar espacio en la pila para los cuatro registros usados para pasar los parámetros, incluso si la función no tiene tantos parámetros.

La situación de la pila al realizar una llamada a función: se muestra en la Figura 3.

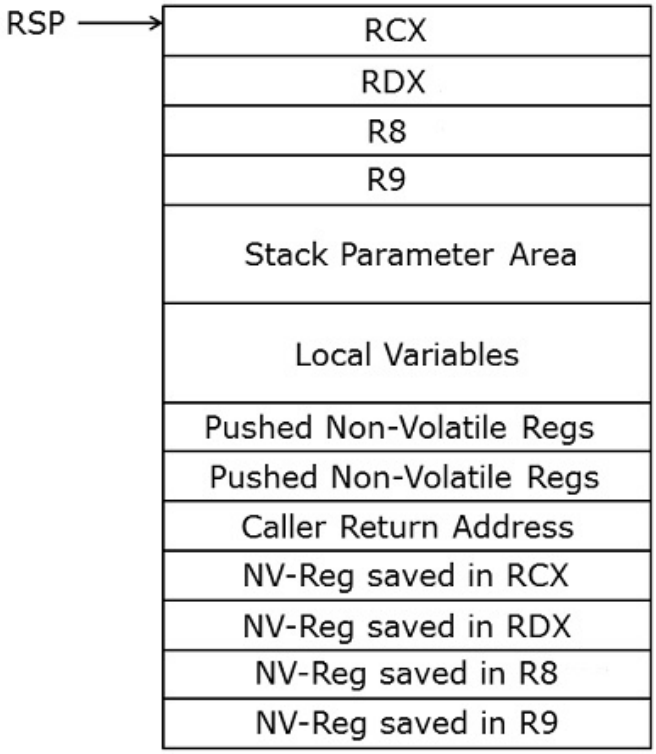



**Figura 3. Dibujo de la pila en 64 bits.**

Observamos cómo, pese a pasar los cuatro primeros argumentos por registro, también se les reserva espacio en la pila. Los registros no volátiles son guardados a lo largo de las llamadas a función. Además no podemos usar RBP como *`frame pointer`*, lo que hace que pase a ser un registro de propósito general, y no pueda ser usado por el *`Debugger`* para recorrer la pila de llamadas. Las funciones además deben restringir el uso de las instrucciones *`push`* y *`pop`* a la creación y destrucción del bloque de activación, ya que el valor de *`stack pointer`* no cambia entre la creación y destrucción de éste.

## *2.2. Mejoras al compilar para 64 bits*

Otro punto a destacar es la mejora que se puede obtener sólo al compilar el programa para 64 bits, sin hacer nada más.

A la hora de compilar una aplicación en 64-bits sobre Windows, debemos saber qué es la capa de abstracción conocida como WoW64 (Windows 32-bit on Windows 64-bit). Se trata de un subsistema de Windows capaz de ejecutar aplicaciones de 32-bits, incluido en todas las versiones de 64 bits de Windows. Hace transparente al programador las diferencias entre los sistemas Windows de 32-bitsy 64-bits, redireccionando el acceso a las librerías de 32 que en los sistemas de 64 bit se encuentran en diferente localización, e interaccionando con el registro de Windows. A alto nivel, es un conjunto de DLLs que recogen las llamadas a y desde procesos de 32 bits, y las traduce (Figura 4).

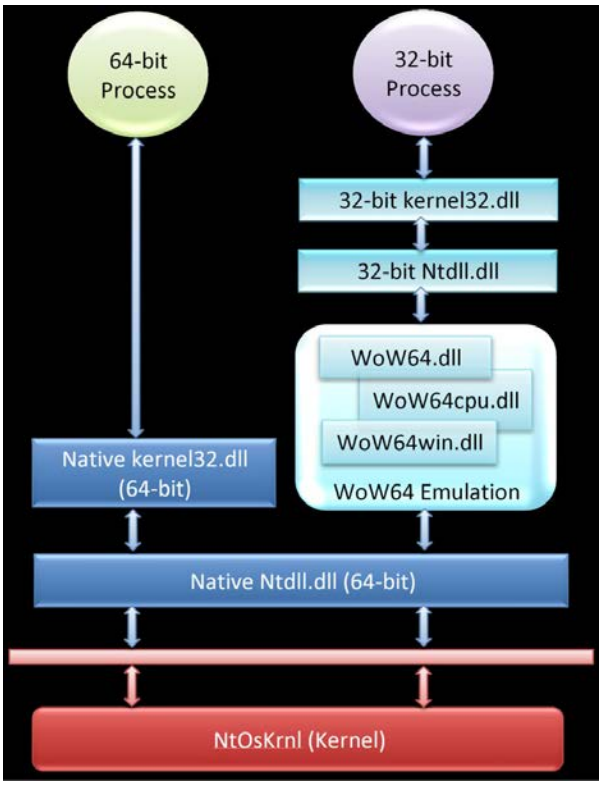



**Figura 4. Modelo de diseño de WoW**

Solamente después de compilar un programa en 64-bits se puede esperar una ganancia de rendimiento del 5 al 15 por ciento, de un 5 a un 10 por ciento solo por el mayor número de registros en una arquitectura de 64-bits y del 1 al 5 por ciento extra al quitar la capa intermedia WoW64. El hecho de contar con más registros en las arquitecturas de 64 bits, hace que el compilador pueda asignar más variables a registros en lugar de en la pila. Esto hace que el acceso a estas variables sea mucho más rápido. Estos porcentajes de mejora y su discusión provienen del autor Andrey Karpov [1].

Otro aspecto que puede ser de gran interés a la hora de realizar una migración de 32 a 64 bits, es la posibilidad de uso de librerías de 32-bits en programas de 64-bits. Hay casos en los que no se puede conseguir una versión de 32 bits de la librería que usamos, ya no tiene soporte, o puede ser que la versión de 64-bits de la librería no funcione correctamente, o no tenga alguna característica necesaria en nuestra aplicación, por ello podemos usar la versión de 32-bits que hemos usado hasta ahora y no perder funcionalidad ni añadir errores.

Se pueden cargar librerías en tiempo de ejecución dentro del espacio de memoria del programa desde el que se cargan, además las librerías cargadas de esta manera pueden necesitar cargar otras más (aumentando en consecuencia el espacio usado). El mayor problema que es la dirección de memoria donde puede residir una aplicación de 64-bits (por encima de los 4GB) o una de 32-bits (hasta 4GB) queda resuelto. El otro problema que puede permanecer es la comunicación de datos entre la

librería de 32-bits y el programas de 64-bits, aunque parece ser que se puede acceder tanto a objetos como a métodos de la librería cargada de esta manera.

Para poder realizar lo mencionado anteriormente, un concepto que hay que conocer es la carga dinámica (*Dynamic Loading*).

Es un mecanismo por el que un programa puede, en tiempo de ejecución, cargar una librería en su memoria, recuperar las direcciones de las variables y funciones que contiene la librería ejecutar esas funciones o acceder a la variables y finalmente quitarla de la memoria donde la había cargado previamente. Esto permite al programa arrancar en ausencia de las librerías para posteriormente cargarlas según las necesite.

La realización de un enlazado dinámico, y no estático se debe a la necesidad de poder enlazar las correspondientes librerías de cámaras distintas. Un enlazado estático crearía un ejecutable de un tamaño muy grande.

| **Dynamic Loading** | | |
|---|---|---|
| **Cometido** | **Windows** | **Linux** |
| Cargar la librería | *LoadLibrary()* | *dlopen()* |
| Obtener método/objeto | *GetProcAddress()* | *dlsym()* |
| Liberar memoria | *FreeLibrary()* | *dlclose()* |

**Figura 5. Funciones para cargar librerías en un programa.**

## *2.2.1. Windows*

*Tal y como podemos ver en la tabla anterior, hay que especificar el `Flag LOAD_LIBRARY_AS_DATAFILE` cuando se llama a `LoadLibraryEx()`.*

Cuando la aplicación ejecuta *`LoadLibrary()`* o *`LoadLibraryEx()`*, el sistema intenta localizar la DLL. Si la encuentra, el sistema mapea el módulo de la DLL en el espacio de direcciones virtual de la aplicación e incrementa el número de referencias en uno. Si la llamada a *`LoadLibrary()`* o *`LoadLibraryEx()`* especifica una DLL cuyo código ha sido mapeado previamente la función simplemente devuelve un manejador para la DLL e incrementa el número de referencias en uno. Dos DLL con el mismo nombre y extensión localizadas en lugares distintos no se consideran las misma DLL.

El sistema llama al punto de entrada del a función dentro del contexto del hilo que llamó a *`LoadLibrary()`* o *`LoadLibraryEx()`.* El punto de entrada de la

función no se llama si la DLL ya había sido cargada previamente y no se había invocado la función *FreeLibrary().*

Si el sistema no puede encontrar la DLL, o si el punto de entrada devuelve *false*, *LoadLibrary()* o *LoadLibraryEx()* devolverán *NULL*.

Si *LoadLibrary()* o *LoadLibraryEx()* tienen éxito, devolverá un manejador del módulo de la DLL. El proceso puede usar este manejador para identificar la DLL y llamar a los siguientes procesos: *GetProcAddress(), FreeLibrary()* o *FreeLibraryAndExitThread().*

El enlazado dinámico en tiempo de ejecución permite al proceso continuar aunque la DLL no esté disponible. El proceso puede ofrecer alternativas al usuario para encontrarla. Puede intentar usar otra, notificar un error o pedirle al usuario la dirección donde se encuentra. También puede causar problemas si la DLL usa la función *DLLMain()* para inicializar cada uno de los hilos del proceso, ya que el punto de entrada no es llamado por lo hilos existentes antes de ser llamado por *LoadLibrary()* o `LoadLibraryEx().`

El funcionamiento es muy parecido al de Linux, se carga la librería con *LoadLibrary()* o *LoadLibraryEx()*, a continuación con *GetProcAddress()* se obtiene el método u objeto que vamos a usar, y por último para liberar la memoria cuando no necesitamos más la librería usamos *FreeLibrary().* Necesitamos incluir la cabecera *<windows.h>* y la librería *Kernel32.dll*.

### *2.2.2. Linux*

El funcionamiento en Linux es muy parecido al de Windows y es el siguiente: primero se carga la librería que queremos con *dlopen()*, ésta nos devuelve un puntero que junto al nombre de la función o el objeto de la librería lo podemos cargar mediante *dlsym()* para usarlo posteriormente. Por último cuando terminamos de usar la librería la borramos de memoria con *dlclose()*. Necesitamos incluir la cabecera *<dlfcn.h>* y las librerías *libdl.so o libdl.dylib* dependiendo del sistema en el que nos encontremos.

## *2.3. Otras formas de mejorar el rendimiento*

Otras formas de mejorar el rendimiento de nuestra aplicación son:

- Uso de *ptrdiff_t, size_t* y tipos derivados de ellos que permiten optimizar el código hasta un 30% según varios *benchmarks*, pese a que los compiladores de hoy en día generan código optimizado, alojando variables de tamaño inferior a 64 bits en registros de 64-bits.

- Declarar funciones como "static" cuando no se usan fuera del fichero en el que están definidas, puesto que permite al compilador hacer *inlining*. Mediante la técnica de *inlining*, el compilador sustituye la llamada a la función por su

código, incrustándolo en el lugar donde se realiza la llamada a la función. De esta manera se evita la sobrecarga que acarrea la realización de una llamada a función, creación y destrucción del espacio reservado en pila, etc.

- Cambiar el orden de las operaciones lógicas, poniendo primero las más propensas a cumplirse o no, para de esta manera ponerlas al comienzo y evitarnos unas cuantas comprobaciones.

- Si en un *switch* no son valores consecutivos, es mejor cambiarlos por *if-elsif*, ya que de esta manera el código generado se ejecuta más rápido, puesto que los *switch* pueden ser tablas de comparación muy costosas de evaluar.

- */favor:INTEL64* está disponible sólo en el compilador de x64 y optimiza el código generado para los procesadores que soportan Intel64, usando características propias de estos procesadores, los cuales suelen tener mejor rendimiento. El código resultante puede ejecutarse en cualquier plataforma x64. (por defecto viene */favor:blend*, que es optimización tanto para Intel como para AMD).

## *2.4. Estimación en coste de la migración*

Para hacer una estimación a priori del coste de la migración del código de 32 a 64 bits, podemos seguir los siguientes pasos:

1. Usar un analizador estático como puede ser PVS-Studio para el proyecto, obteniendo todos los avisos generados por este.

2. Un programador instruido en el tema de 64 bits, analiza todos los avisos (*n_tot*) generados por el analizador durante una jornada y decide si el error es relevante o no (falsa alarma). Si lo es, lo corrige (*n_err*).

3. El total de mensajes producidos por el analizador se divide por el número de mensajes que el programador ha solucionado en un día (*n_tot/n_err*).

4. Por último el resultado es el número de personas/día que se necesitan para llevar a cabo la migración de la aplicación a 64 bits. (*tot = n_tot/n_err = personas/día*)

## *2.5. Guía de buenas prácticas*

En los siguientes puntos se exponen ejemplos y prácticas para la escritura de código portable, sea cual sea la arquitectura sobre la que trabajemos:

- Ya que los enteros y punteros tienen el mismo tamaño en ILP32, se usan indistintamente. Los punteros se convierten a tipos enteros o sin signo para aritmética de direcciones. Además podemos convertir un entero a *long* o *unsigned long* ya que también tienen el mismo tamaño. Podríamos hacer esto entre *long* y punteros en LP64 ya que mantienen el mismo tamaño.

La mejor opción consiste en usar *uintptr_t*, ya que hace que el código no se tenga que cambiar si por ejemplo dejásemos de usar LP64 y además queda más claro nuestra intención. Para usarlos necesitamos incluir *<inttypes.h>.*

```
char *p;
p = (char *) ((int)p & PAGEOFFSET);
% cc ..
warning: conversion of pointer loses bits
```

Así sería correcto:

```
char *p;
p = (char *) ((uintptr_t)p & PAGEOFFSET);
```

- Usar tipos de datos portables, es decir, si cambiamos de plataforma, no tengamos que cambiar de nuevo los tipos de datos y éstos se mantengan. Algunos de estos tipos son: *size_t, ssize_t, ptrdiff_t, uintptr_t*, etc
- Tener cuidado con las estructuras de datos compuestas por varios de éstos. Debido al alineamiento de los datos a 8 bytes en máquinas de 64 bits, podemos calcular mal el tamaño de una estructura.

```
struct MyPointersArray
{
    DWORD m_n;
    PVOID m_arr[1];
}
object;
...
malloc( FIELD_OFFSET(struct MyPointersArray, m_arr) +
    5 * sizeof(PVOID) );
```

Al ocupar DWORD 4 bytes, tenemos libres los 4 restantes hasta completar los 8 que ocupa PVOID en 64 bits, para ello necesitamos usar FIELD_OFFSET o también podemos usar *offsetof().*

- En general, usar tipos capaces de almacenar el tamaño de un puntero en 64 bits (o sea 8 bytes). Estos tipos son: *ptrdiff_t, size_t, intptr_t, uintptr_t, ssize_t, int_ptr*, *DWORD_PTR*, etc. Con esto además nos aseguramos que sea la arquitectura que sea podemos direccionar toda la memoria permitida y no vamos a obtener ningún error de acceso a memoria. Además en las condiciones de los bucles nos evitamos el error que puede surgir de comparar un dato de 32 bits con uno de 64 bits haciendo de esta manera que el bucle sea infinito.

LA Figura 6 muestra la tabla con el tamaño de los tipos de datos y su tamaño en las diferentes implementaciones:

| Type name | Type size (32-bit system) | Type size (64-bit system) | Description |
|---|---|---|---|
| ptrdiff_t | 32 | 64 | Signed integer type which appears after subtraction of two pointers. This type is used to keep memory sizes. Sometimes it is used as the result of function returning size or -1 if an error occurs. |
| size_t | 32 | 64 | Unsigned integer type. Data of this bype is returned by the sizeof() operator. This type is used to keep size or number of objects. |
| intptr_t, uintptr_t, SIZE_T, SSIZE_T, INT_PTR, DWORD_PTR, etc | 32 | 64 | Integer types capable to keep pointer value. |
| time_t | 32 | 64 | Amount of time in seconds. |

| | ILP32 | LP64 | LLP64 | ILP64 |
|---|---|---|---|---|
| char | 8 | 8 | 8 | 8 |
| short | 16 | 16 | 16 | 16 |
| int | 32 | 32 | 32 | 64 |
| long | 32 | 64 | 32 | 64 |
| long long | 64 | 64 | 64 | 64 |
| size_t | 32 | 64 | 64 | 64 |
| pointer | 32 | 64 | 64 | 64 |

**Figura 6. Tamaño de tipos básicos en distintas Sistemas Operativos de 64 bits.**

### *2.5.1. Ejemplos*

A continuación se muestran ejemplos de código propio, en el que se hubiera evitado realizar cambios al migrar a 64 bits si se hubieran adoptado las buenas prácticas anteriores.

- En la clase `SharedBuffer().`

```
/**
        Función que nos devuelve el tamaño en BYTES del buffer
        @return int: el tamaño en bytes de buffer
*/
int size_bytes() const
{
        return (buff_data_->size_ * sizeof(_T));
}
```

Habrá que cambiar el tipo que devuelve la función a *`size_t`*, ya que si no estamos perdiendo los 32 bits más significativos.

- Clase *`ImageData().`*

```
bmfh.bfSize = (width*height*bits_per_pixel/8) +
sizeof(BITMAPINFOHEADER) + sizeof(BITMAPFILEHEADER) + 1024;
```

Estamos extendiendo los valores de *`width`*, *`height`* y *`bits_per_pixel`* que son *`int`* los dos primeros y *`unsigned char`* el último.

```
if ( (pitch == (width*(bits_per_pixel>>3))) &&
(origin==IMAGE_ORIGIN_BL) )
        fwrite(buffer.buffer(), 1, buffer.size(), f);
```

Aquí, como tercer argumento de la función *`fwrite()`* pasamos el resultado del método *`size()`* de la clase buffer, que es del tipo entero. En la definición de *`fwrite()`* el tercer argumento es del tipo *`size_t`*, así si tuviéramos un tamaño mayor que MAX_INT perderíamos información. Por esto se recomienda usar *`size_t`* para datos que representen tamaños.

- Clase *Avifile().*

```
if (wf)
{
    for (unsigned i = 0; i < sz && i < sizeof(WAVEFORMATEX); i++)
        ((char*)wf)[i] = b[i];
}
```

Aquí el error está en usar i como índice del vector *wf*. i es del tipo *unsigned*, por lo que no podríamos direccionar el máximo de memoria posible con él y por tanto no podríamos acceder a todos los elementos. Si bien es verdad que si no va a ocupar más del límite no es un error, pero hay que estar totalmente seguro de ello. Aquí lo deseable hubiese sido que i fuera del tipo *uintptr_t*. Además en algunos bucles el definir el índice como un entero o cualquier tipo que no ocupe 64 bits puede hacer que aparezcan bucles infinitos.

- Clase *dec_enc().*

El siguiente ejemplo puede o no ser un error, puesto que depende de la definición del tipo de dato al que le aplicamos la función *sizeof()*, ya que puede haber *padding* y por tanto darnos un tamaño diferente al real.

```
typedef struct tagBITMAPFILEHEADER {
        WORD    bfType;
        DWORD   bfSize;
        WORD    bfReserved1;
        WORD    bfReserved2;
        DWORD   bfOffBits;
} BITMAPFILEHEADER, FAR *LPBITMAPFILEHEADER, *PBITMAPFILEHEADER;

typedef struct tagBITMAPINFOHEADER{
        DWORD      biSize;
        LONG       biWidth;
        LONG       biHeight;
        WORD       biPlanes;
        WORD       biBitCount;
        DWORD      biCompression;
        DWORD      biSizeImage;
        LONG       biXPelsPerMeter;
        LONG       biYPelsPerMeter;
        DWORD      biClrUsed;
        DWORD      biClrImportant;
} BITMAPINFOHEADER, FAR *LPBITMAPINFOHEADER, *PBITMAPINFOHEADER;

unsigned long header_size = sizeof(BITMAPFILEHEADER) +
sizeof(BITMAPINFOHEADER);
```

- IPP (JPEG)

Este es un claro ejemplo de error en aritmética de punteros, en la cual hay que usar siempre tipos del tamaño de la arquitectura: *size_t, ptrdiff_t, uintptr_t*, etc Aunque no debería dar errores puesto que ese usa para obtener los argumentos de la línea de comandos, y ésta no va a ser mayor que 4GB.

```
key->m_length = curr - pos;
```

Tanto *curr* como 'pos' son punteros (8 bytes) y la longitud *m_length* es un entero. Para que estuviera totalmente bien escrito, *m_length* debería ser del tipo *ptrdiff_t*, que es un entero con signo pero que tiene el tamaño de la arquitectura en la que estamos en cada momento.

## 2.6. Diferentes analizadores estáticos

Análisis estático de código es el proceso de detección de errores y defectos en el código fuente del software. Puede realizarse de forma manual o automática.

La revisión manual de código es uno de los métodos más antiguos y seguros en la detección de errores. Se trata de dar recomendaciones en cómo mejorar el código fuente, revelando errores en el presente o posibles errores en el futuro. Además la función del código tiene que ser clara simplemente leyendo el texto y los comentarios, sino hay que mejorarlo. Este método suele funcionar bien porque es más fácil encontrar errores en el código de los demás que en el de uno mismo.

La única desventaja es el alto precio de este método: se necesita juntar a varios programadores cada cierto tiempo para revisar el código más nuevo que se va creando o el código con los cambios sugeridos aplicados, además no pueden hacer esto muy a menudo puesto que su atención disminuiría.

El análisis estático de código automático permite revisar grandes cantidades de código con menos recursos humanos. No realiza modificaciones automáticas, pero proporciona recomendaciones oportunas para que el programador lo haga, y el ratio uso/precio hace de esta solución una de las más usadas en muchas empresas.

Las tareas que un analizador estático de código resuelve (o al menos intenta resolver) se pueden dividir en tres categorías:

- Detección de errores en programas.
- Recomendaciones en el formato del texto, se puede adaptar al estándar que tenga cada empresas: indexación, uso de espacios o tabuladores, etc.
- Medidas relacionadas con el código, nos permiten obtener un valor numérico representativo sobre nuestra propia escala de valores para la propiedad que queramos medir.

Hay también otras maneras de usar los analizadores estáticos de código. Por ejemplo puede usarse para enseñar a los nuevos desarrolladores las normas de escritura de código de la empresa.

Como cualquier otra metodología de análisis estático tiene sus puntos fuerte y sus puntos débiles. Sabemos que no hay un método ideal de testeo de software. Métodos diferentes dan resultados diferentes sobre el mismo código, sólo la combinación de varios hace posible que podamos alcanzar la mayor calidad en nuestro software.

La principal ventaja del analizador de código estático es que permite reducir el precio de eliminar defectos en el software de manera notable. Cuanto antes detectemos los errores menor será el precio que nos costará corregirlos. De acuerdo con el libro *`Code Complete de McConnell`* un error en la etapa de testeo es diez veces más caro de corregir que si lo detectamos en la fase de escritura.

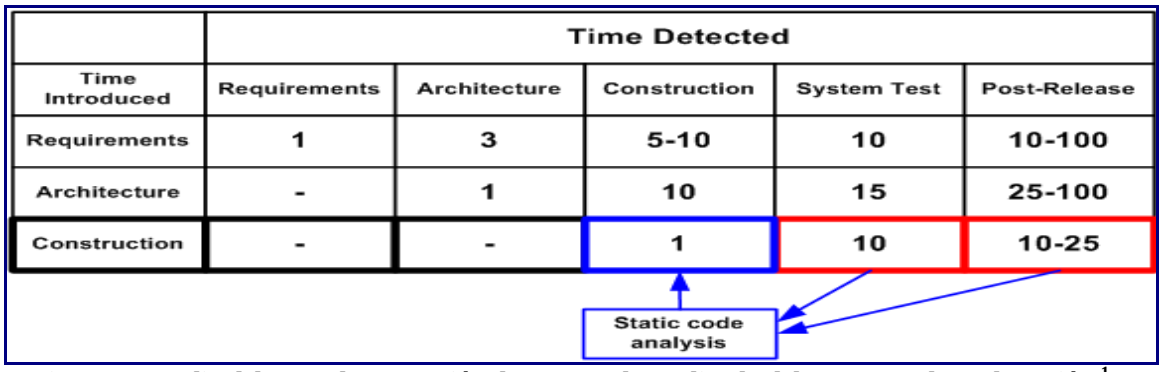



**Figura 7. Media del coste de corrección de errores dependiendo del momento de su detección.**[1]

Las herramientas de análisis estático de código permiten detectar rápidamente un montón de errores en la fase de escritura de código, lo que reduce significativamente el coste de todo el proyecto.

Otras ventajas de los analizadores estáticos de código son:

- Podemos analizar todo el código que queramos, incluso aquellos trozos de código que no se ejecutan casi nunca y que puede costarnos ver los errores allí presentes más que en las zonas más comunes, excepciones y *logs* por ejemplo.
- No dependen del compilador que se use ni de donde se ejecutará el programa. Permite encontrar errores que de otra manera serían difíciles de descubrir, evitando que de repente aparezca un error y resulte estar en todas las versiones desde hace varios años, con el coste que eso conlleva de arreglar.
- Posibilidad de detectar fácil y rápidamente los típicos errores de copiar y pegar muy comunes en largos trozos de código muy parecido. Además aunque los errores son muy fáciles de corregir, se puede perder mucho tiempo para descubrir errores triviales del estilo de *`strcmp(A,A)`*.

[1] McCONNEL, S., *Code Complete. A practical handbook of software construction*, Microsoft Press, 2004.

- aunque no lo parezca en estos errores se pierde mucho tiempo hasta que se detectan.

Las desventajas de los analizadores estáticos de código son:

- No suelen diagnosticar en profundidad fugas de memoria y errores de concurrencia. Para analizar esto se recurre a herramientas de análisis dinámico (Valgrind, Intel Studio XE, etc).
- Advierte sobre errores en fragmentos de código que pueden resultar raros y que sólo el programador puede entender si es un error de verdad o una falsa alarma. Además de este tipo de falsas alarmas pueden darse otras, por lo que a veces se añade un tiempo extra en la revisión de estos avisos.

Los errores detectados con este tipo de analizadores de código son muy diversos. Algunos se centran más en un tipo o área concretos y otros soportan estándares como por ejemplo: MISRA-C:1998, MISRA-C:2004, Sutter-Alexandrescu Rules, Meyers-Klaus Rules, etc.

A menudo aparecen nuevas reglas de diagnóstico y estándares, y otras se quedan obsoletas, por eso no tiene sentido comparar los analizadores estáticos de código en ese sentido. La mejor manera de compararlos es aplicarlos a los mismos proyectos y ver cuántos errores reales detecta cada uno, para así saber cuál es el más conveniente.

Los usuarios quieren comparar los diferentes analizadores estáticos de código, es algo totalmente comprensible. No es fácil comparar diferentes analizadores estáticos de código. El problema reside en establecer criterios de comparación adecuados.

Así por ejemplo, comparar el número de errores que deberían ser detectados y el número de mensajes generados puede parecer una medida muy razonable. Sin embargo no resulta adecuada en nuestro caso, como vamos a explicar a continuación.

Primero empecemos con los parámetros que no tiene sentido que comparemos entre analizadores. Por ejemplo, el número de avisos que es capaz de dar al analizar un archivo o varios. Podemos pensar que cuantos más avisos mejor será el analizador, pero normalmente esto no es así, puesto que el usuario, sólo hace uso de una parte del sistema por lo que no va a tener que tratar con todos los tipos de errores. Por ejemplo los avisos relacionados con librerías y compiladores, no le van a aportar nada relevante, o incluso llegar a ser un estorbo.

Consideremos por analogía una persona que entra en una tienda para comprar un ordenador, en esta tienda hay una amplia variedad de productos, pero esta persona no los necesita, está bien que pueda comprar en la misma tienda una televisión o una videoconsola, pero esto no hace que el ordenador que vaya a comprar sea mejor.

El número de avisos no está relacionado con el número de errores que el analizador puede detectar en un proyecto concreto. Un analizador que tenga en cuenta 200 tipos de avisos enfocados todos ellos a aplicaciones desarrolladas para Windows, puede encontrar muchos más errores en un proyecto con Visual Studio que un analizador multi-plataforma que tenga en cuenta 1500 tipos distintos de avisos.

En definitiva, el número de avisos no debe ser relevante a la hora de comparar varios analizadores estáticos de código.

Tampoco sería correcto comparar el número de avisos relevantes para un sistema en concreto, debido a que:

- Puede ser que un tipo de aviso esté implementado en una regla en un analizador y en otra regla en otro analizador. Entonces uno nos presenta más avisos por pantalla, y decimos que es mejor que el otro, cuando el otro nos está avisando de los mismos posibles errores pero de una manera más compacta.
- El mismo diagnóstico puede tener más "calidad" en un analizador que en otro. Para definir calidad vamos a poner un ejemplo. La mayoría de analizadores tienen en cuenta los llamados números mágicos (valores constantes codificados en el código fuente). Son potencialmente peligrosos en la migración a 64 bits, ya que los tamaños de datos y rangos de valores cambian. Puede que un analizador puede tenga en cuenta solamente los que son peligrosos desde el punto de vista de la migración a 64 bits (4, 8, 32 etc) y que otro tenga en cuenta todos los números mágicos (1, 2, 3 etc).

Otra característica que podemos estar interesados en medir es la velocidad o número de líneas procesadas por segundo. Sin embargo no tiene sentido, puesto que en última instancia el que va a hacer los cambios y va a tener que leerse todos los avisos es el encargado de revisar la salida del analizador, y no hay relación entre la velocidad del analizador y la velocidad a la que puede realizar los cambios una persona. Normalmente hay un parámetro que olvidamos al comparar analizadores, que es la usabilidad del propio analizador por la persona o personas que van a tener que trabajar con él.

Aquí lo importante es que la usabilidad de una herramienta como es un analizador de código influye mucho en la práctica real del uso por parte del programador.

He analizado el proyecto de las IPPMedia, una interfaz propia para el uso de las librerías de Intel IPP (Integrated Performance Primitives), con Visual Studio, Cppcheck y PVS-Studio. Se han detectado algunos aspectos relacionados con el manejo de Visual Studio que viene integrado en el IDE y Cppcheck, y no está relacionados con la calidad o al velocidad del análisis en sí, sino con aspectos de usabilidad que se analizan a cotinuación.

- **Preservación de los mensajes generados.-** Estas herramientas no permiten guardar una lista con los mensajes generados para examinarlos más tarde. Es fácil en un proyecto de este tamaño obtener miles de avisos cuyo análisis requiere varios días. Al no poderse guardar es preciso cada vez volver a analizar todo el proyecto, con la pérdida de tiempo que ello conlleva, además de tener que recordar el último aviso corregido. PVS-Studio sí que permite guardar los resultados y cargarlos cada vez que se quiere continuar leyendo los avisos generados para ir corrigiéndolos.
- **Procesado de mensajes duplicados**.- Aparecen normalmente en los archivos de cabecera (.h). Por ejemplo el analizador detecta un posible problema en una cabecera que la incluyen 15 archivos (.cpp) En lugar de dar el aviso cada vez que

se encuentra con la inclusión del archivo de cabecera, PVS-Studio da el error una vez en la cabecera ya que cuando se corrige ahí ya no lo va a dar en los ficheros que lo incluyen. El siguiente mensaje se dio más de cinco veces mientras se analizaba la IPPMedia:

*d:\ippmedia_x64_7.1\examples\consola.h(92):*
*warning C6054: HANDLE 'std_output' might not be initialized*

Debido a que cada vez que se incluye el archivo de cabecera se escribe este error, la salida del analizador pude parecer muy grande y desordenada haciendo que haya que revisar más mensajes de los necesarios.

- **Selección de los ficheros a analizar.-** Visual Studio y Cppchek analizan ficheros de *plug-ins* como pueden ser los que se encuentran en rutas como: *C:\Archivos de programa\Microsoft Visual Studio 9.0\VC\include*. Esto no tiene sentido, puesto que nadie va a editar los ficheros del sistema. PVS-Studio no "pierde" el tiempo analizando este tipo de ficheros. Un ejemplo es:

*c:\archivos de programa\microsoft*
*sdks\windows\v6.0a\include\ws2tcpip.h(729):*
*warning C6386: Buffer overrun: accessing 'argument 1',*
*the writable size is '1*4' bytes,*
*but '4294967272' bytes might be written:*
*Lines: 703, 704, 705, 707, 713, 714, 715, 720,*
*721, 722, 724, 727, 728, 729*

Visual Studio y Cppcheck no permiten excluir ficheros del análisis en función de un patrón como puede ser: **_test.cpp* o *c:\librerias*, mientras que con PVS-Studio podemos hacer esto.

- **Gestión de la lista de avisos.-** Podemos deshabilitar ciertos avisos en el analizador de código, pero a diferentes niveles según nuestros requerimientos. En algunos analizadores debemos repetir la ejecución en cada caso. Con PVS-Studio a diferencia de Cppcheck podemos manejar estas opciones una vez concluido el análisis diciéndole que tipo de mensajes queremos que nos muestre y cuáles no. De esta manera podemos centrarnos en los que más nos interesan o son más críticos para nuestra aplicación, sin necesidad de volver a realizar todo el análisis con el tiempo que eso conlleva.
- **Filtrado de los avisos que se generan en la salida por texto**.- Por ejemplo podríamos querer ocultar los relacionados con funciones comunes como *printf()* o *scanf()* que comúnmente dan generan avisos por no usar su versión segura o parecido.
- **Falsas alarmas.**- Podemos marcar en Visual Studio con *#pragma warning* para deshabilitar las falsas alarmas, pero con PVS-Studio podemos marcar un mensaje como falsa alarma y de esta manera no volver a ser mostrado sin volver a lanzar el analizador. Esta función es importante en PVS-Studio porque para mi y al menos

en los proyectos en los que lo he usado, se generan muchos avisos y una gran parte son falsas alarmas.

Podemos concluir que la comparación entre analizadores estáticos es difícil, y que no hay una respuesta clara a que herramienta es mejor en general. Podríamos decir cuál es mejor para proyectos en concreto pero no en general.

Un ejemplo que ilustra lo expuesto, tomado de los casos surgidos en los análisis que he efectuado, es el siguiente: el analizador indicaba que estaba usando un tipo de dato que no tenía el tamaño de la memoria del sistema (8 bytes) para indexar un vector. En este caso lo que hay que hacer al migrar a 64 bits es mirar si la estructura se espera que contenga más de *INT_MAX* elementos o no, si es que no se deja como está porque al migrar posiblemente cueste mucho tiempo cambiar el tipo de dato para nada. Si la respuesta es que si va a contener más de INT_MAX elementos hay que cambiar el tipo de dato a *size_t, intptr_t*, que tienen el tamaño máximo de la arquitectura donde se estén ejecutando (si es en 32 bits ocupan 32 y si es en 64 ocupan 64 bits, esto es algo que está definido en el lenguaje y que el compilador se encarga de cumplirlo). Tuve la idea de probar si de verdad llegaba hasta INT_MAX o no. Al ejecutar el código para hacer esta prueba el vector, se llenaba más allá de INT_MAX elementos. El código era el siguiente:

```
int indice = 0;
size_t tamanyo = ...;
for (size_t i = 0; i != tamanyo; i++)
  array[indice++] = BYTE(i);
```

Aunque por la condición del bucle puede perfectamente llegar a más de INT_MAX elementos, al acceder con *indice* debería de darnos un error puesto que es del tipo entero que ocupa 32 bits. Depurando y mirando el código ensamblador se observa que el compilador había usado registros de 64 bits para almacenar la variable de tipo entero, puesto que en las máquina de 64 bits hay más registros y es la forma más rápida de acceder a los datos.

```
mov         byte ptr [rcx+rax],cl
add         rcx,1
cmp         rcx,rbx
jne         wmain+40h (120001080h)
```

Debido a esta optimización del compilador, este fragmento de código común en muchas de aplicaciones es incorrecto.

Decidí pues cambiar el código fuente para ver si el compilador no era capaz de optimizarlo usando registros de 64 bits y efectivamente, ahora no podía contener más de *INT_MAX* elementos como cabía esperar.

```
int indice = 0;
for (size_t i = 0; i != tamanyo; i++)
{
```

```
  array[indice] = BYTE(indice);
  ++indice;
}
movsxd      rcx,r8d
mov         byte ptr [rcx+rbx],r8b
add         r8d,1
sub         rax,1
jne         wmain+40h (140001040h)
```

Ahora como era de esperar el registro usado no es de 64 bits (*rcx*) sino que es de 32 bits (*r8d*).

# Capítulo 3

## Aplicación *Examples*

En este capítulo se va a explicar la creación de la aplicación Examples. Esta aplicación es una versión actualizada y completamente creada desde cero de una aplicación ya existente que realiza esta tarea para arquitecturas de 64 bits.

*Examples* consiste en un códec de vídeo para los formatos H.264 y MPEG-4 que utiliza la librería de las IPP para realizar esta función. Permite decodificar/codificar varios archivos de vídeo a la vez, creando un hilo por cada uno para su ejecución. Descomprime el archivo de entrada para devolver el mismo vídeo comprimido.

Los parámetros de la aplicación son:

- -i: nombre(s) de los fichero(s) de entrada
- -m: nombre(s) de los fichero(s) descomprimidos, salida del *decoder* y entrada del *encoder*.
- -o: nombre(s) de los fichero(s) de salida comrpimidos.
- --REF: nombre del fichero de referencia para calcular *PSNR (Peak Signal-to-Noise Ratio).*
- -t: número de hilos, debe coincidir con el número de ficheros de entrada.
- -r: resolución de los archivos de entrada (ancho x alto).
- -f: formato de color del archivo de salida *(gray yv12 nv12 yuy2 uyvy yuv420 yuv422 yuv444 rgb24 rgb32 bgr24 bgr32 bgr565 bgr555 bgr444).*
- -b: *bitdepth* del archivo de salida.
- -n: límite de frames a procesar.
- -d: hacer o no deinterlineado.
- -k: no mantener el ratio de aspecto si cambiamos el tamaño.
- --RGB: guardar cada frame en un archivo *.bmp*
- --AVI: crear un archivo *.avi* con las imágenes descomprimidas.
- -p: archivo de parámetros
- -c: tipo de codificador de vídeo *(mpeg4 y h264)*.
- -R: resolución del archivo de salida (ancho x alto).
- -B: bitrate (bits/second). Por defecto 2000000.
- -F: framerate (frames/Second). Por defecto 30.
- -h: ayuda de la aplicación.
- --ipp_cpu: optimización del código de las IPP usado *(SSE, SSE2, SSE3, SSSE3, SSE41, SSE42, AES, AVX, AVX2).*
- --ipp_threads: número de hilos internos para la ejecución de las IPP, por si usamos la versión con hilos de ésta.

El diagrama de flujo del descompresor se muestra en la Figura 9, y la del descompresor en la Figura 10.

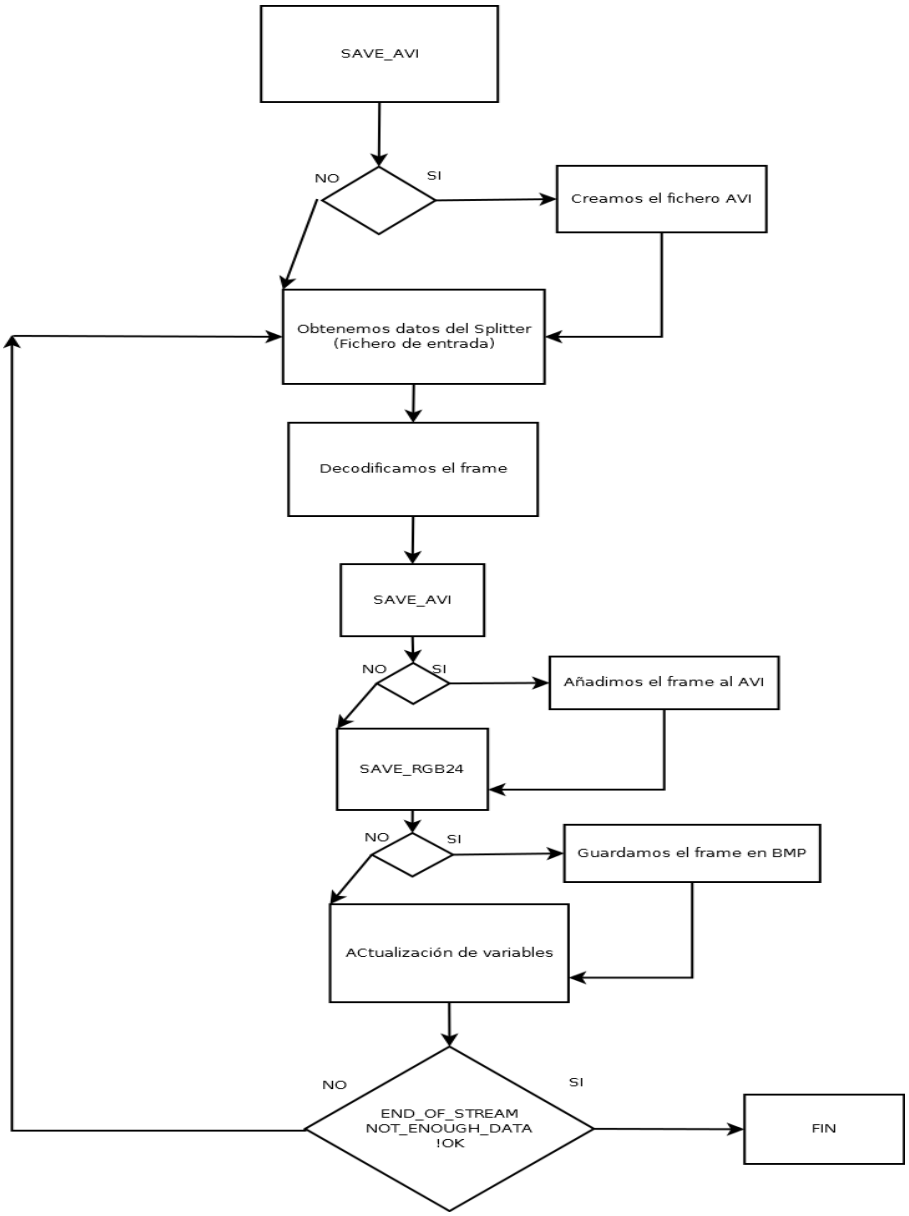



**Figura 8. Diagrama de flujo del descompresor.**

La lógica del compresor la podemos entender con este diagrama de flujo:

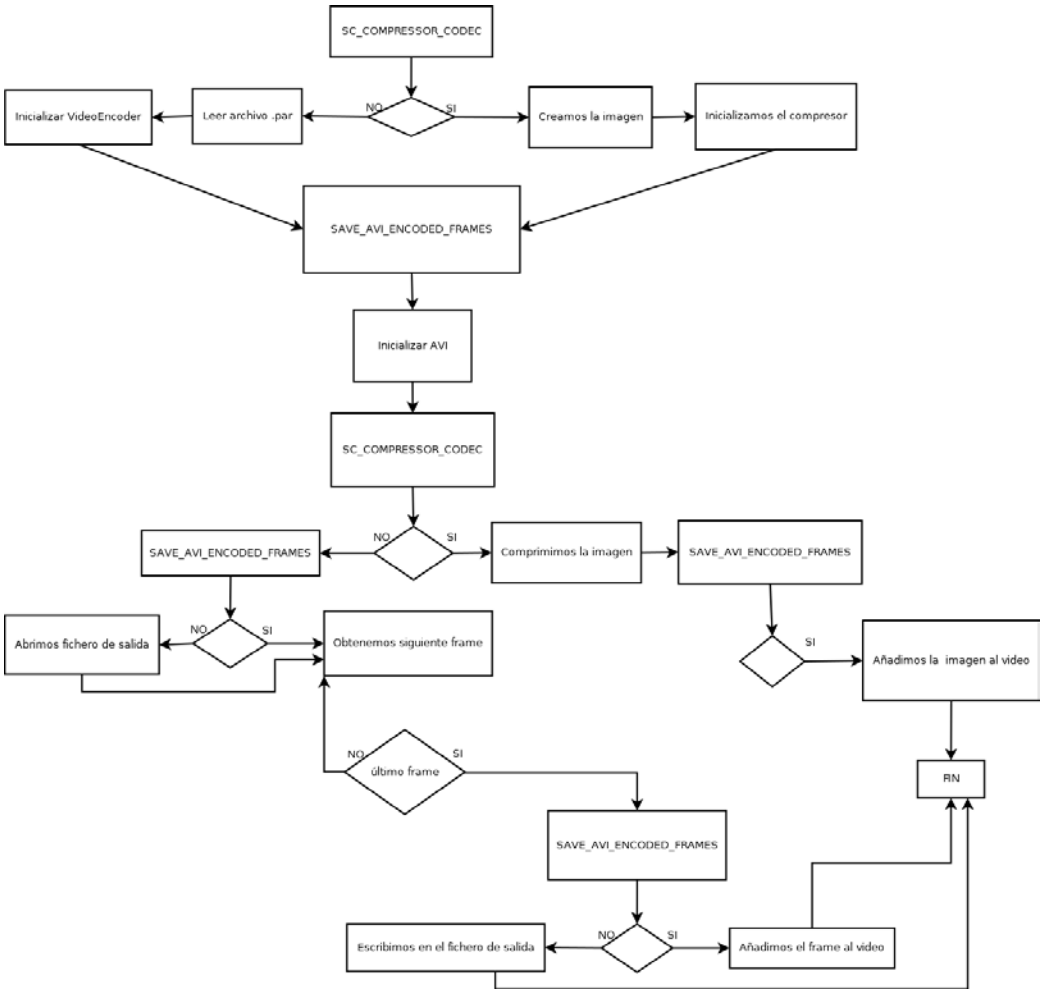



**Figura 9. Diagrama de flujo del compresor.**

# Capítulo 4

## 4. Migración de ScatiRTPVideoClient a 64 bits

Tras haber completado las fases anteriores, ahora nos vamos a centrar en la migración completa de una aplicación comercial de 32 a 64 bits, objetivo principal del proyecto. La aplicación elegida ha sido ScatiRTPVideoClient. Esta aplicación se encarga de recibir vídeos de un servidor, para mostrarlos por pantalla, a la vez que se pueden grabar en disco, comprimir y todo ello teniendo en cuenta sólo los *keyframes* o la secuencia completa de vídeo. Un *keyframe* es una imagen a partir de la cual se puede estimar el movimiento realizado hasta la siguiente, sin necesidad de tener toda la información. En la aplicación se puede definir el intervalo de tiempo entre keyframes para obtener una mayor calidad (disminuyéndolo) o para obtener mayor rendimiento (aumentándolo). Estas características son opuestas, por lo que si queremos una de ellas tenemos que perder en la otra.

### 4.1. FUNCIONAMIENTO CLIENTE - SERVIDOR

El servidor, que no hemos migrado a 64 bits, funciona de la siguiente manera:

1. Iniciamos la aplicación:

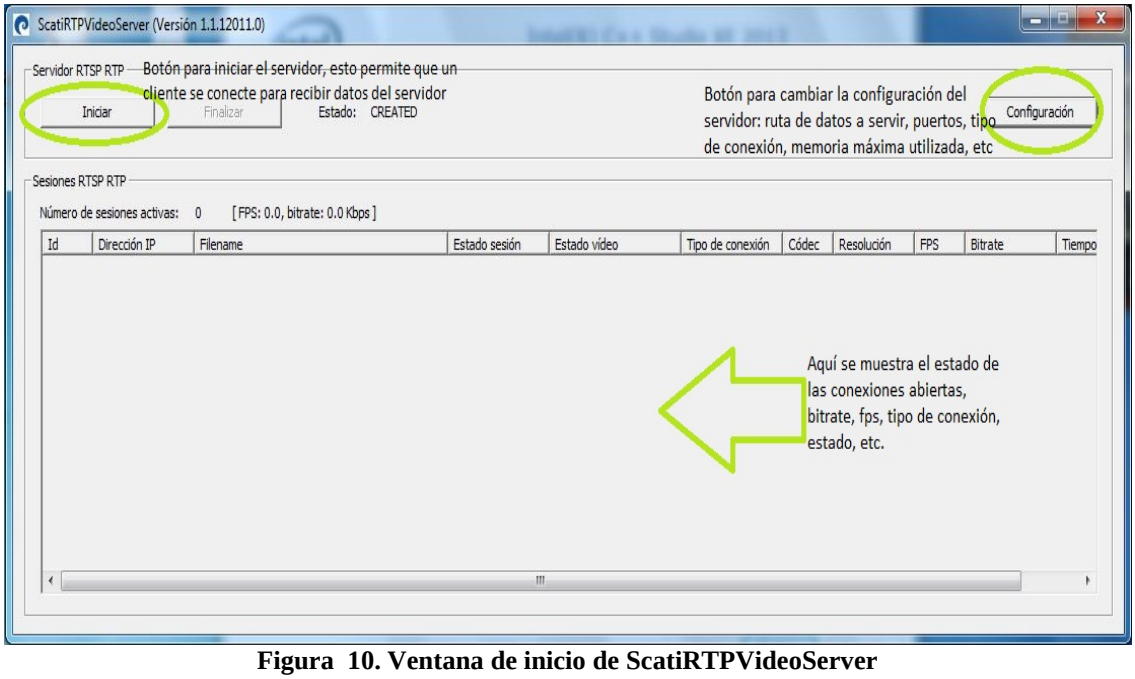



**Figura 10. Ventana de inicio de ScatiRTPVideoServer**

Esta es la apariencia del servidor sin iniciar, como se pude ver en la imagen podemos iniciar el servicio presionando en el botón "Iniciar" arriba a la izquierda, o podemos configurar las opciones con el botón "Configuración" que se encuentra arriba a la derecha.

2. Iniciamos el servicio:

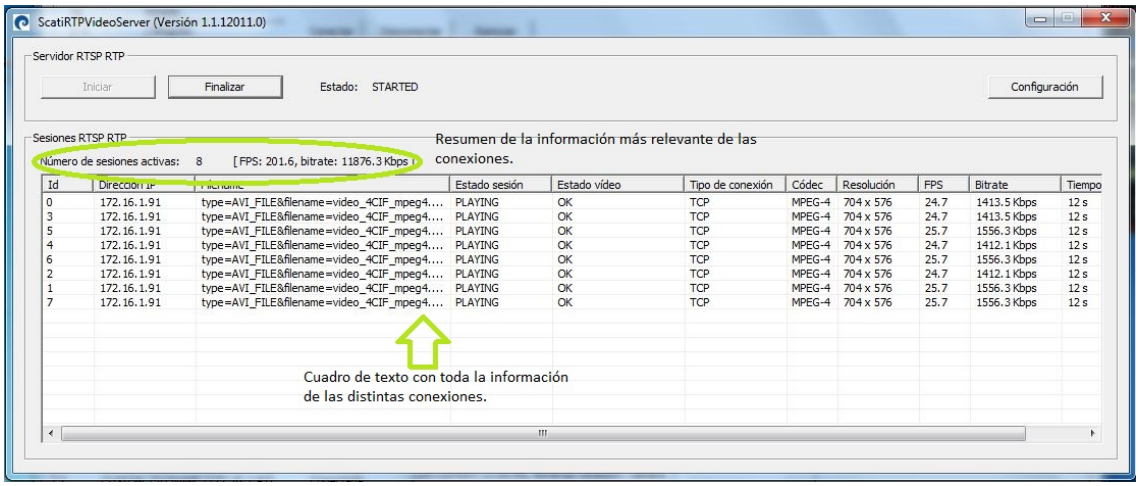



**Figura 11. Ventana de estado de conexiones de ScatiRTPVideoServer**

Aquí podemos observar cómo se ha deshabilitado la opción de iniciar y se ha habilitado la opción de finalizar. Además se muestran todas las conexiones activas, con la información correspondiente asociada: ID, dirección IP, tipo y nombre del archivo que está siendo servido, estado de la sesión, estado del vídeo, tipo de la conexión, códec de vídeo, resolución, FPS a los que se está sirviendo, *bitrate* y el tiempo de la conexión.

Opcionalmente, podemos cambiar las opciones:

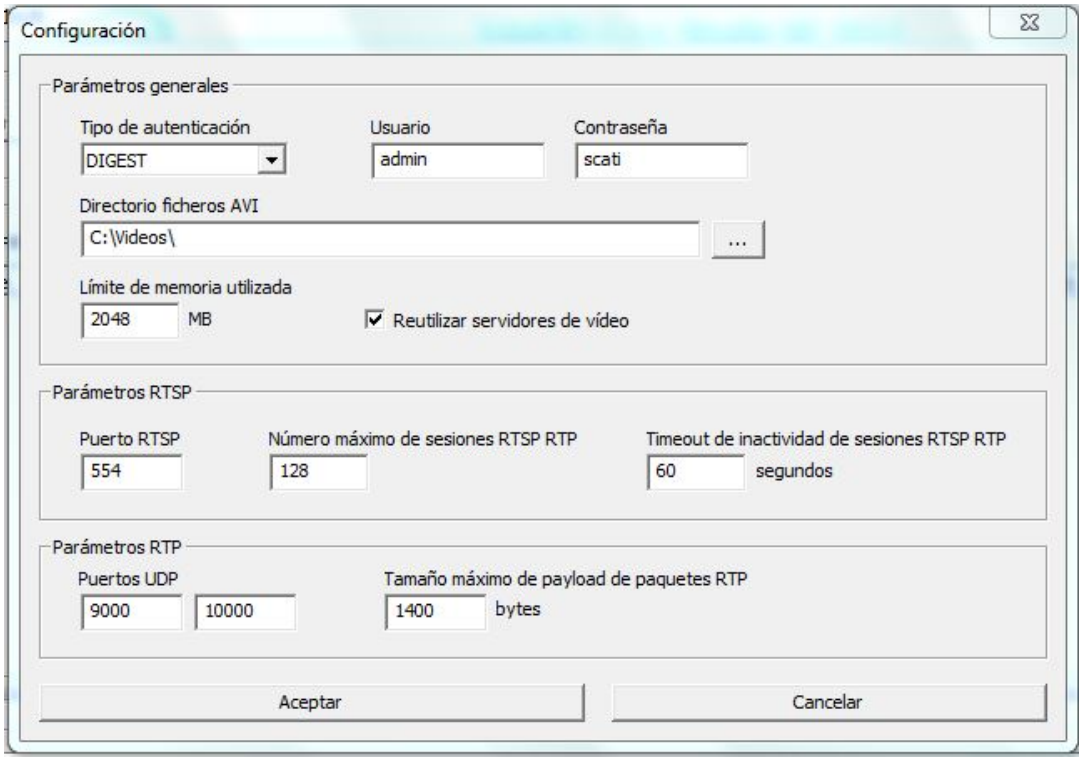



**Figura 12. Ventana de configuración de ScatiRTPVideoServer**

Las opciones más destacadas son: usuario y contraseña, ruta donde se encuentran los archivos, máximo de memoria utilizable por el programa, reutilización de servidores de vídeo[2] , puertos para las conexiones y *timeout* de conexión.

Vamos a describir ahora el funcionamiento de la aplicación sobre la que hemos realizado la migración, SCATIRTPVideoClient.

[2] Esto hace que cuando hay muchas conexiones y falla alguna no se creen nuevas, sino que se utilicen las previamente creadas

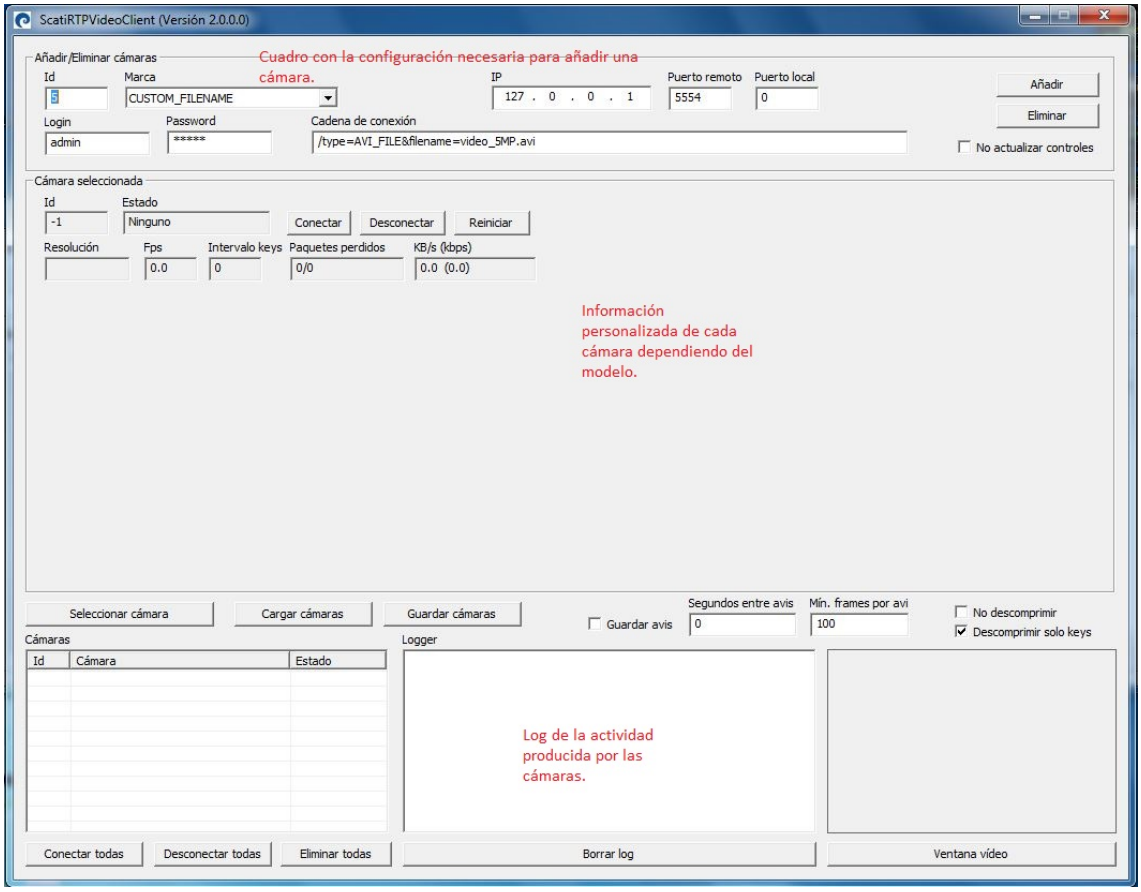



**Figura 13. Ventana de inicio de ScatiRTPVideoClient**

La Figura 13 muestra la ventana de nuestra aplicación. Podemos observar las diferentes partes en las que está dividida. En la parte superior se encuentran los campos necesarios para añadir una cámara, para conectarnos a ella posteriormente, en el centro se muestran las características de la cámara seleccionada. Por último la parte inferior esta divida en tres zonas. A la izquierda tenemos el cuadro donde se muestran todas las cámaras conectadas, junto a las opciones que nos permiten añadir, conectar/desconectar y eliminar las cámaras. En el centro hay una ventana de log, donde se registran los sucesos ocurridos asociados a cada una de las cámaras que estemos manejando en cada momento, y por último, a la derecha se encuentra la ventana de vídeo donde se muestra el vídeo recibido de la cámara. Además hay otras opciones para el tratamiento de las imágenes recibidas de las cámaras, como grabación a disco, descomprimir o no, sólo *keyframes*, etc.

A continuación cargamos 8 cámaras para hacer una prueba, cada una de estas cámaras nos manda un vídeo (Figura 14).

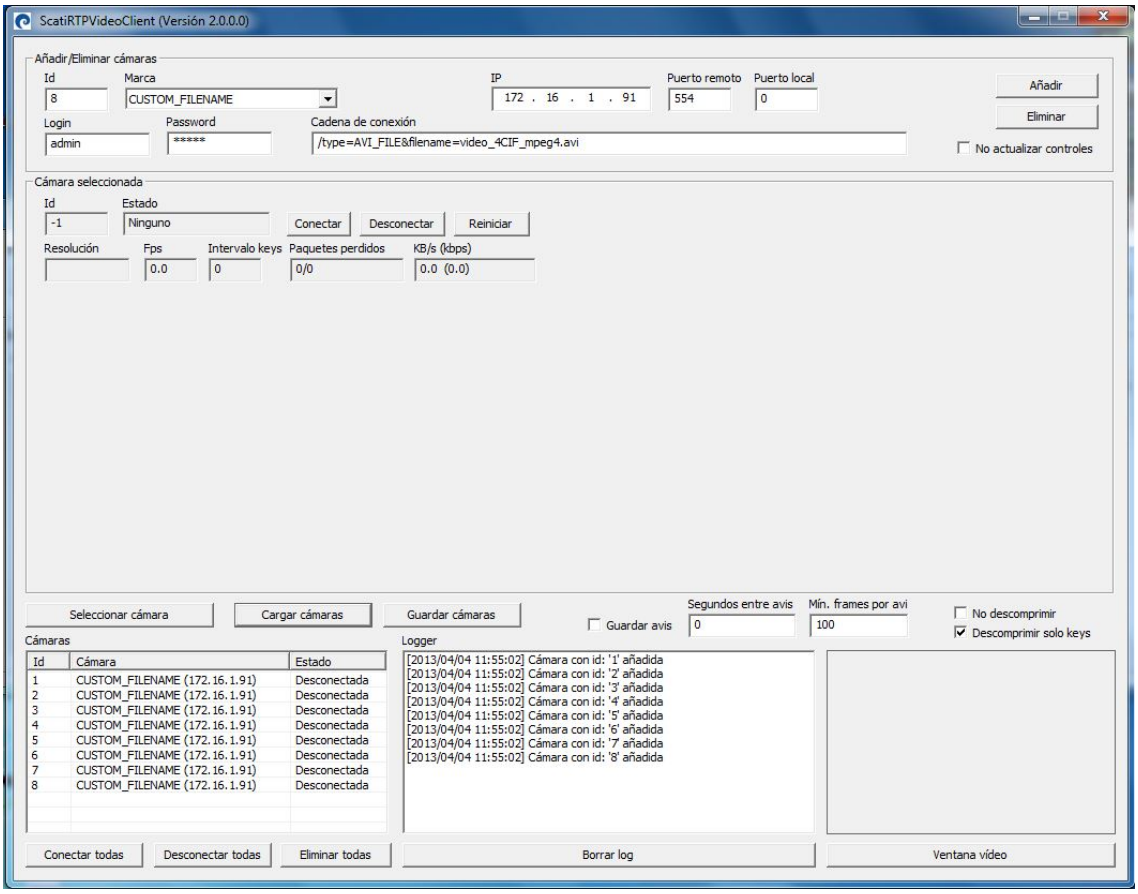



**Figura 14. Ventana de vista de cámaras ScatiRTPVideoClient**

Vemos como aparecen cada una de las cámaras que conectamos a nuestra aplicación, además en la ventana de log se nos informa de que han sido añadidas estas ocho cámaras.

El siguiente paso es conectarlas, ya que sólo las hemos añadido en el paso anterior (Figura 15).

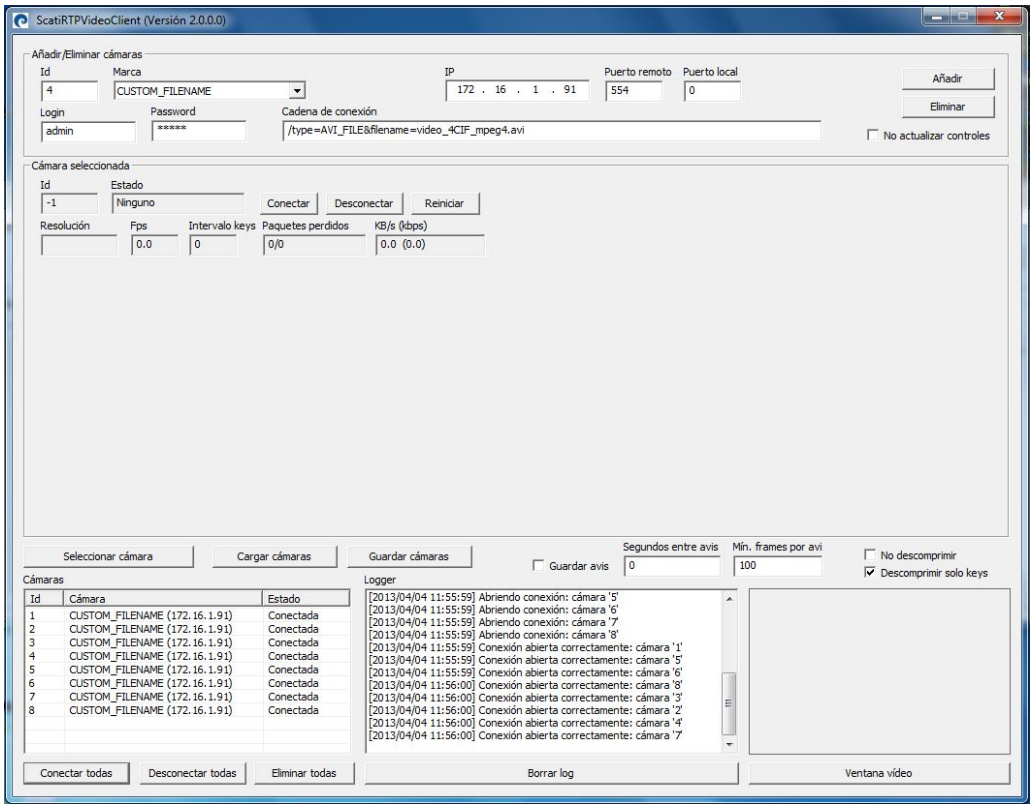



**Figura 15. Ventana de conexión de cámaras ScatiRTPVideoClient**

Una vez conectadas, podemos visualizar cualquiera de las ocho cámaras. La Figura 16 muestra, un ejemplo.

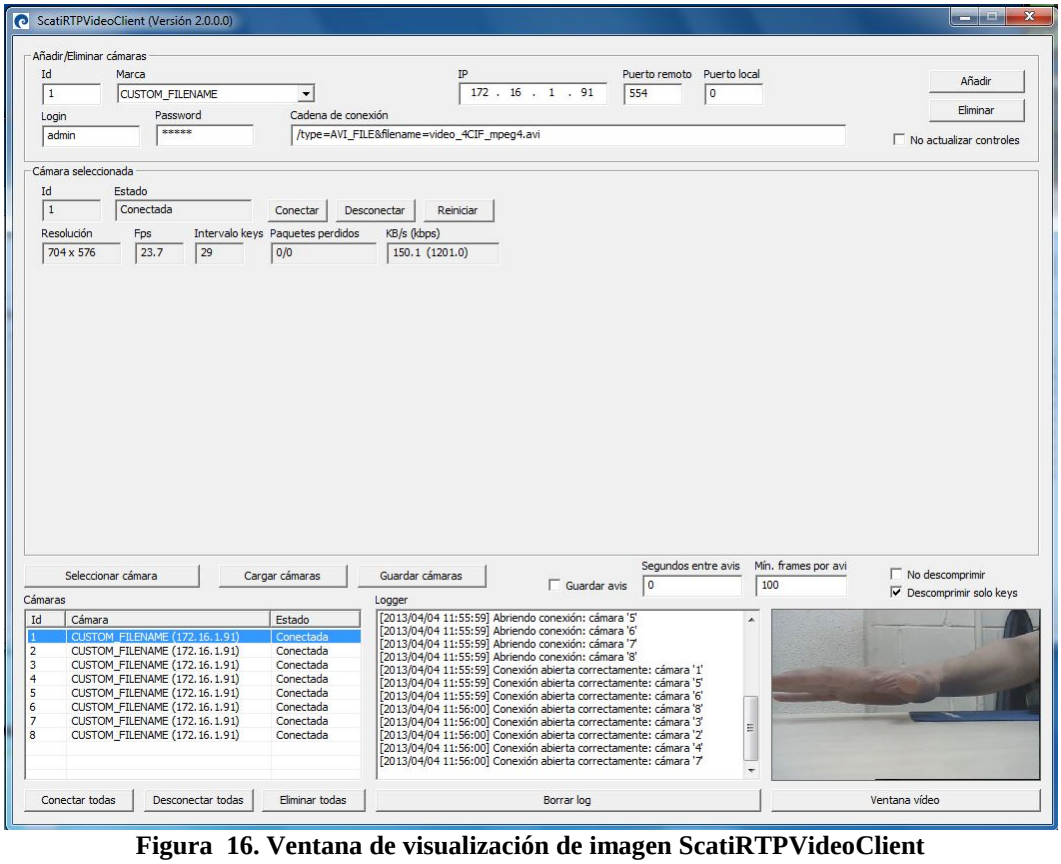



**Figura 16. Ventana de visualización de imagen ScatiRTPVideoClient**

Observamos cómo, en este caso, la información asociada a una cámara situada en el centro de la aplicación, nos muestra información relacionada con el vídeo recibido. En este caso no hay más opciones, puesto que para probar, los vídeos los sirve el servidor que hemos introducido anteriormente y no una cámara real.

Ahora sólo faltaría desconectar todas las cámaras y eliminar las conexiones para poder cerrar la aplicación (Figura 17 y 18).

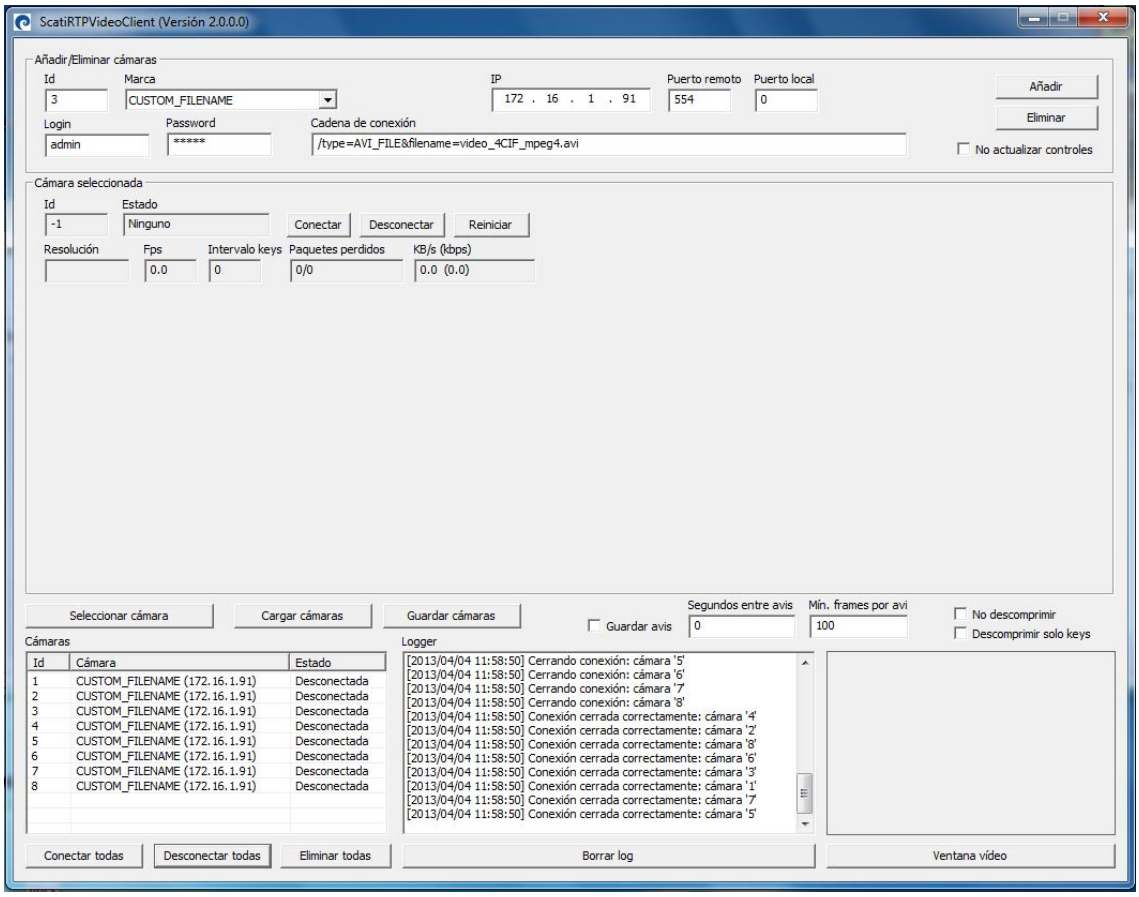



**Figura 17. Ventana de desconexión de cámaras ScatiRTPVideoClient**

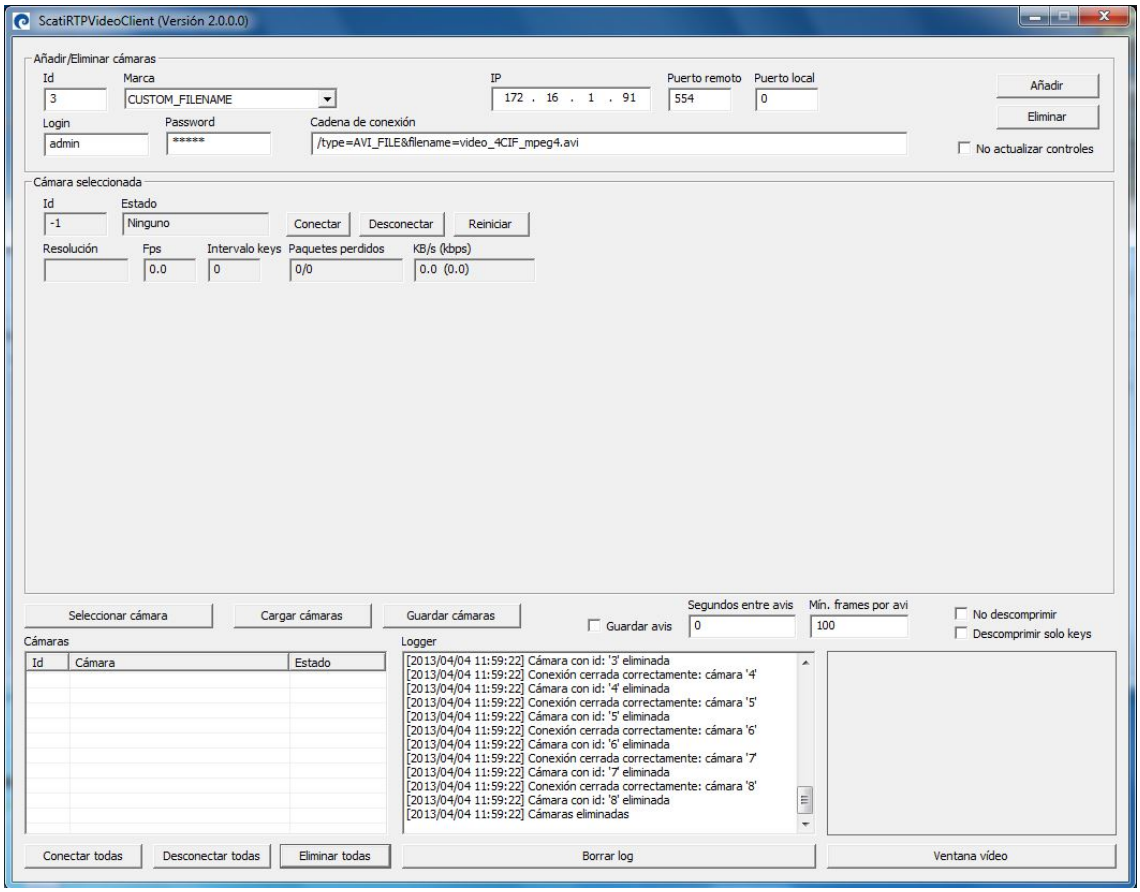



**Figura 18. Ventana de eliminación de cámaras ScatiRTPVideoClient**

Podemos comprobar que la aplicación está completamente migrada a 64 bits desde el administrador de tareas de Windows, donde a diferencia de otras aplicaciones para 32 bits, ésta carece de los caracteres *32 al final del nombre (Figura 19).

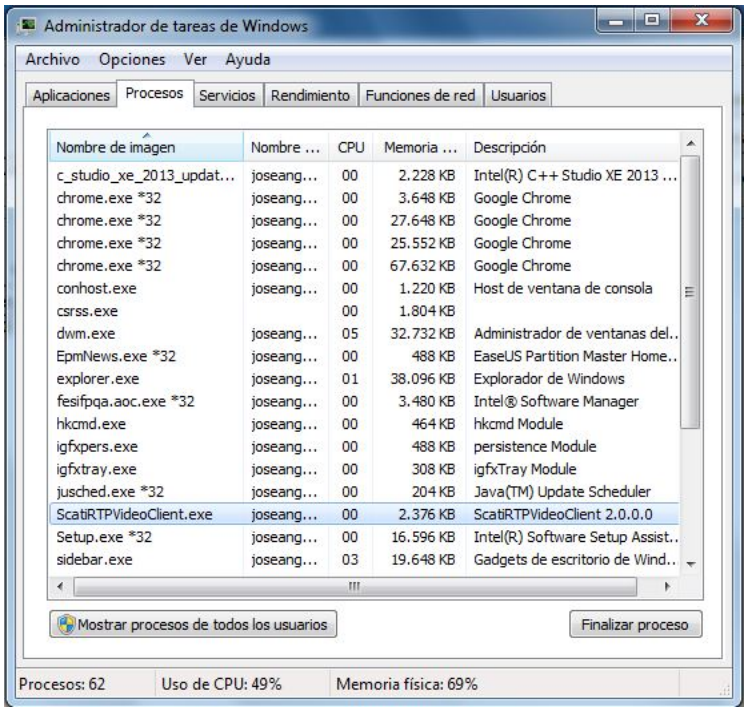



**Figura 19. Ventana de recursos del sistema Windows 7.**

## 4.2. Estimación de la migración

A partir del coste temporal que supone la migración de esta aplicación, analizado a continuación, podemos estimar el coste de la migración a 64 bits de la aplicación *VisionSurfer*.

### 4.2.1. ScatiRTPVideoClient

- Migración librerías (mínimo 1 día para cada librería: última versión y compilar) (total: 136h):
  - Codec de vídeo IPP: 2 semanas (14 días - 112h).
  - ACE: 1 día (falta compilar con opciones de compilador) (8h)
  - Cryptopp: 1 día (8h).
  - AviFile: 1 día (8h).

- Formación 64 bits (warnings, tipos de datos, instalación herramientas): 3 días (24h)

- Compilación 64 bits (opciones compilador, directorios, etc.): 2 días (16h)

- Migración código 64 bits (warnings) (total: 28h)
- Corregir warnings Visual Studio C++: 1 día (8h)
- Ejecutar PVS-Studio y corregir warnings: 2,5 días (20h)

- Pruebas ejecución 32 y 64bits: 2 días (16h)

- Problemas no planificados (p.e. avifile): 2 días (16h)

- Documentación (cambios librerías, warnings, compilación...): 3 días (24h)

**Total**: **260h** (32,5 días) (6 semanas y media).

### 4.2.2. ScatiVision

Migración librerías (mínimo 1 día para cada librería: última versión y compilar) (total: 280h):

- IJL: (están 'obsoletas', ahora se utilizan ejemplos de IPP como en H.264 y MPEG-4): 2 semanas y media (100h) (riesgo)
- SIP: (funciones eliminadas de IPP, buscar alternativas, etc.): 2 semanas y media (100h) (riesgo).
- El resto de librerías (están en una tabla en el estudio) tienen versión de 64 bits y no es necesario (a priori realizar cambios en el código fuente): 2 semanas (80h)

- Formación 64 bits (warnings, tipos de datos, instalación herramientas): 3 días (24h) (por persona) (suponemos 2 personas: 48h)

- Compilación 64 bits (opciones compilador, directorios, etc.): 5 días (40h)

- Migración código 64 bits (warnings) (multiplicamos por 10 - según las líneas de código) (total: 280h)
- Corregir warnings Visual Studio C++: 10 días (80h)
- Ejecutar PVSStudio y corregir warnings: 25 días (200h)

- Pruebas ejecución 32 y 64bits: 5 días (40h)

- Problemas no planificados (p.e. avifile): 10 días (120h) (riesgo)

- Documentación (cambios librerías, warnings, compilación...): 10 días (80h)

**Total**: **888h** (111 días) (22 semanas y 1 día)

## 4.3. Cambios librerías para 64 bits

A continuación se detallan los cambios realizados en las librerías para su correcto funcionamiento en 64 bits.

**IPPMedia:**

- Se ha cambiado de versión de IPP de **6.1.3.047** a **7.1.1.119**.
- Se han añadido los cambios realizados en SCATI a los ejemplos de las IPP de la nueva versión:

- Forzado de *keyframes* en compresores H.264 y MPEG-4.
- Añadido límite de *bitrate* en modo de calidad constante en compresor H.264.
- Cambios de formato de color revisados.

- Reestructuración de librerías resultantes de IPPMedia:
  - En lugar de tener una sola librería 'IPPMedia.lib', ahora se tienen múltiples librerías de cada uno de los módulos de los ejemplos (p.e. 'h264_enc.lib', 'h264_dec.lib', etc). Esto es debido a que el script que viene con ellas permite hacer esto más fácilmente que juntar todas las librerías en una sola.
  - En cada aplicación que utilice las librerías IPP, hay que utilizar la función 'InitPreferredCpu()' para que se utilicen las optimizaciones de la CPU correspondiente. De esta manera el código de las funciones optimizadas de las IPP cambia dependiendo de la arquitectura.
  - No se han incluido las MFC (Microsoft Foundation Classes), librería que envuelve parte del API de Windows en C++, en las librerías de los ejemplos de las IPP.
  - Utilización de las librerías de IPPMedia en el proyecto de ejemplos tanto en 32bits como en 64bit:
    - Es necesario añadir las librerías de los ejemplos utilizadas (utilizando 'pragma', tal y como se hacía hasta ahora con la librería IPPMedia).

      ```
      #pragma comment (lib, "vm_plus.lib")
      ```

    - Necesario añadir las cabeceras de las IPP necesarias en los ficheros que las utilizan.

      ```
      #include "umc_app_utils.h"
      ```

    - Es necesario añadir las siguientes librerías para el enlazado: shell32.lib, ole32.lib, oleaut32.lib, uuid.lib, advapi32.lib, winmm.lib pdh.lib, SetupAPI.lib.
  - Añadida captura de excepciones en los ejemplos de las IPP (en las propiedades de los proyectos de ejemplos 'Enable C++ Exceptions: Yes With SEH Exceptions (/EHa)'). De esta forma se capturan excepciones producidas en los ejemplos, que de otra forma no se podrían capturar y harían que el programa que utiliza estos ejemplos se cerrara, sin poder recuperar el sitio en el que ha tenido lugar el error.

**ACE:**

- Se ha cambiado de versión de ACE de **5.7.1.1** a **6.1.0**. El cambio se ha realizado para trabajar con la última versión de estas librerías.
- Se han compilado tanto en 32 bits como en 64 bits sin tener que realizar ninguna modificación:

- Inicialmente no se han modificado las directivas de compilación. Se van a realizar las pruebas iniciales; de esta forma de comprueba si funcionan las librerías con los cambios mínimos necesarios.
  - No se utilizan las directivas ACE_HAS_MFC, ACE_LEGACY_MODE ni _USE_32BIT_TIME_T. Esta última no tiene sentido en 64-bits.

**Cryptopp:**

- Se ha cambiado de versión de CryptoPP de **5.6.0** a **5.6.1**. El cambio se ha realizado para trabajar con la última versión de estas librerías.
- Se han compilado tanto en 32 bits como en 64 bits:
  - Se han modificado las propiedades del proyecto de las librerías Cryptopp de *'Multi-threaded'* a *'Multi-threaded DLL'* (tanto en *Debug* como en *Release*).

**Avifile:**

- Se han compilado estas librerías tanto en 32 bits como en 64 bits sin tener que realizar ninguna modificación.
  - Pese a no tener que realizar ninguna modificación, han aparecido varios warnings relacionados con el cambio de tamaño en la arquitectura de 64 bits.

```
warning C4267: 'initializing' : conversion from 'size_t'
to 'int', possible loss of data

warning C4244: '+=' : conversion from 'int64_t' to
'uint_t', possible loss of data

warning C4244: 'return' : conversion from 'time_t' to
'uint_t', possible loss of data

warning C4244: '=' : conversion from 'LRESULT' to
'unsigned long', possible loss of data
```

Una vez realizados los cambios oportunos en las librerías, se han adaptado las librerías propias de 32-bits a los cambios producidos en las nuevas versiones de las librerías de terceros usadas.

- Adaptación a nueva versión de IPP:
  - Cambios en las siguientes clases:
    - **IPP_DecompressorCodec**: modificaciones para adaptarse a los cambios de los ejemplos de las nuevas IPP.
    - **SC_CompresorCodecIPP**: modificaciones para adaptarse a los cambios de los ejemplos de las nuevas IPP. Se ha detectado una fuga de memoria en el compresor H.264 de las IPP utilizando la herramienta de Intel: Studio XE.

        - **ImageProcess**: modificaciones en las funciones de redimensionado; las funciones utilizadas hasta ahora ya no existen (estaban *deprecated* hace varias versiones de las IPP. Se han modificado por las funciones.
- Adaptación a nuevas librerías de ACE:
    - No se utilizan versiones especiales de ACE con MFC; se ha cambiado el fichero *svc.conf* para eliminar *mfc*.
    - No se utiliza la directiva de precompilación *_USE_32BIT_TIME_T*.
- Cambios en las propiedades del proyecto
    - En configuración *Debug* ha sido necesario añadir la siguiente librería como ignorada: *LIBCMTD*.
- Se ha cambiado la directiva de compilación de *__WIN32__* a *__WINDOWS__* para distinguir entre sistemas operativos Windows y otros.
- Se ha detectado un problema en la escritura de ficheros AVI con la librería AviFile. Es un error muy extraño ya que se produce sólo en *Release* en 32 bits; parece que hay algún tipo de incompatibilidad con esta librería y con la utilización de las funciones *_open*, *_close* y *_write*, se han cambiado por las funciones *fopen*, *fclose* y *fwrite*.
- Se ha corregido un problema porque no se llegaba a utilizar la CPU al 100% con varias descompresiones simultáneas, y es que el programa sólo realizaba una descompresión simultáneamente. Se ha detectado utilizando la herramienta de Intel: Studio XE.

## 4.4. Solución warnings migración a 64 bits

Una vez completado este proceso, se ha compilado para 64-bits con éxito sin realizar ningún cambio, salvo la aparición de *warnings* propios de la migración, como la conversión entre tipos de datos de tamaños diferentes.

```
... warning C4267: 'argument' : conversion from 'size_t' to 'int', possible loss of data ...
```

- Se han corregido los warnings detectados en la compilación anterior añadiendo las conversiones necesarias a entero. Con estos cambios se han eliminado todos los warnings. Lo ideal sería utilizar los tipos de datos correspondientes siempre que fuera posible (evitar el uso de *int* y utilizar *size_t* (o *ssizet_t*) cuando corresponda.
- Se modifica el fichero de ACE: OS_NS_unistd.inl para evitar un warning similar a los anteriores en la llamada a la función *swab*.

Por último, se ha usado la herramienta de análisis estático de código PVS-Studio, para obtener un informe detallado de todos los posibles warnings relacionados con la migración.

Muchos de los warnings obtenidos en esta sección están relacionados con la conversión de *memsize* a *int*, por lo que no haría falta cambiarlos.

- Se han ignorado los siguientes warnings tras ver que no son imprescindibles:

  V101, V102, V103, V104, V106, V108, V110, V112, V113, V117, V119, V121, V220, V302

  Hay muchos warnings en el código de ACE, pero al ser una librería que no depende de nosotros no podemos entrar a cambiar nada, puesto que lo más seguro es que estropearíamos algo.

- Hay varios avisos también en el código de los ejemplos de las IPP.
- Otros warnings también en: *ScatiTime, ImageData, SharedBuffer*.

*Se ha observado también algún warning relacionado con la no inclusión de la directiva de precompilación de ACE -> _USE_32BIT_TIME_T*.

En SharedBuffer: ¿Devolvemos size_t o hacemos un cast a int? V110.

```
int size_bytes() const
 {
        return (buff_data_->size_ * sizeof(_T));
 }
```

Aparece el aviso V111 en las macros de ACE (habría que intentar evitar este warning).

- Avisos en las funciones de log de *init_encoder* de *SC_CompresorCodecIPP* V204.

**Warning GA (General Analysis)**

- Ignorados: V524, V547 Comprobaciones que son siempre verdaderas o falsas, V550 No parece dar problemas, V595 No comprobación de NULL al utilizar punteros.

**Otros Warnings**

- Modificado *SC_Base64*, la función iba más allá del límite superior del vector, aunque en ejecución no se había registrado ningún fallo. En el fichero *PtzCommand.cpp* se puede producir *underflow* del buffer, debido a que el tamaño que se usa en la función *memcpy* es un entero y por lo tanto puede ser negativo, habría que usar un dato del tipo unsigned. V512.

- Se ha producido por una indexación incorrecta en un **if** al que sólo le sigue una instrucción y que por tanto no tenía llave, V628.

- Modificado *ComprStats* para utilizar puntero a *char* en lugar de *std::string* V510.
  - Se han corregido varios cambios de tipo incorrectos en clases de *ipcamstest_gui*: V576, V601.

- Eliminadas algunas variables declaradas pero no utilizados: V808 Variables declaradas pero no usadas, V807 Se usaba varias veces el mismo acceso a una clase y se ha creado un objeto que lo contenga para acceder únicamente al principio a la clase y mejorar el rendimiento.

- Algunos V803 al no utilizar el elemento anterior del *iterator* en el bucle es más efectivo usar *++iterator* que *iterator++* , algunos V802 optimización de tamaño de estructuras reduciendo su tamaño en memoria y algunos V809 al usar la función *delete()* no hace falta comprobar el puntero, por ahora no se quitan.

# Capítulo 5

## 5. Pruebas de rendimiento

En este capítulo se presentan las pruebas de rendimiento realizadas con la aplicación SCATIRTPVideoClient, la cual se conecta a un servidor de vídeo mediante el *protocolo RTP (Real-time Transport Protocol)* y *RTSP (Real Time Streaming Protocol)* y muestra las imágenes por pantalla, comprime el vídeo para guardarlo en disco, etc.

El objetivo de estas pruebas es conocer el límite de conexiones que podemos tener abiertas a la vez en la aplicación, comparando las versiones de 32 y 64 bits. Estas pruebas se han realizado en dos ordenadores diferentes, para tener una mejor visión de la diferencia en cada caso, no sólo por mejora del hardware, sino por la optimización de la librería de las IPP. Esta librería es el cuello de botella de la aplicación, pues es la encargada de hacer el procesamiento de datos con mayor carga de trabajo.

### 5.1. Examples

Las primeras pruebas se han realizado sobre la aplicación *examples* encargada de descomprimir y comprimir vídeo.

| | | | | | | | | Decodificación | | | Codificación | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | Total | Decodificación | Conversión | Total | Codificación | Conversión |
| Códec | Resolución | Cámaras | CPU Dec | CPU Enc | Memoria Dec | Memoria Enc | Fps | Fps | Fps | Fps | Fps | Fps |
| H.264 | 4CIF | 1 | 22,00% | 50,00% | 13 | 44 | 90,85 | 1147,41 | 287,79 | 15,76 | 16,98 | 438,68 |
| H.264 | 4CIF | 2 | 95,00% | 75,00% | 26 | 86 | 22,51 | 1073,81 | 244,7 | 13,82 | 15,03 | 351,82 |
| H.264 | 4CIF | 4 | 98,00% | 80,00% | 49 | 172 | 7,935 | 1127,8 | 275,3025 | 3,4 | 14,2475 | 323,295 |
| H.264 | 4CIF | 8 | 99,00% | 65,00% | 99 | 342 | 3,62375 | 577,5575 | 244,00125 | 1,27875 | 13,39 | 319,9 |
| H.264 | 4CIF | 16 | 15,00% | 50,00% | 198 | 681 | 1,57 | 616,485 | 236 | 0,55 | 14,05 | 332,85 |
| H.264 | 1MP | 1 | 24,00% | 50,00% | 25 | 130 | 45,33 | 286,2 | 103,98 | 6,68 | 7,32 | 146,08 |
| H.264 | 1MP | 2 | 82,00% | 45,00% | 109 | 260 | 8,58 | 127,435 | 65,53 | 2,315 | 6,335 | 118,88 |
| H.264 | 1MP | 4 | 20,00% | 42,00% | 230 | 522 | 3,455 | 160,0475 | 82,3675 | 0,855 | 6,37 | 120,7925 |
| H.264 | 1MP | 8 | 15,00% | 40,00% | 447 | 1042 | 1,16375 | 245,18375 | 91,2175 | 0,31125 | 5,5725 | 113,585 |
| H.264 | 5MP | 1 | 15,00% | 22,00% | 140 | 455 | 2,83 | 226,2 | 23,49 | 0,69 | 3,03 | 37,34 |
| H.264 | 5MP | 2 | 15,00% | 27,00% | 279 | 909 | 1,22 | 159,51 | 21,065 | 0,305 | 2,545 | 27,5 |

**Figura 20. Rendimiento aplicación `Examples` 64 bits.**

La Figura 20 muestra los *frames* por segundo en descompresión y compresión para las versiones de 32 y 64 bits de la aplicación *Examples*, así como el porcentaje de uso de CPU y la cantidad de memoria virtual usada. Nótese que el *SpeedUp* conseguido está en torno a 1. Esto nos indica que no hay una mejoría de rendimiento por el cambio a 64-bits. La migración en todo caso permite manejar más memoria, algo muy importante para la aplicación, puesto que con ocho cámaras de 1MPixel, ya ocupamos 1GB de memoria virtual, lo que supone la cuarta parte del máximo accesible con 4 GB.

Otro dato a destacar, pese a que no está relacionado con la migración a 64 bits, es el mejor aprovechamiento de la CPU a la hora de manejar dos vídeos simultáneamente debido al uso de hilos, ya que al contar con 2 CPUs, el trabajo se distribuye mucho mejor. De aquí podemos concluir que sería ideal contar con un núcleo por cada flujo de vídeo a tratar, pero esto es imposible para un gran número de cámaras (por encima de 32), dado que un número mayor de flujos de vídeo que de núcleos de procesamiento provoca una pérdida importante de rendimiento debido a esperas de sincronización entre hilos.

## 5.2. SCATIRTPVideoClient

A continuación se detallan las pruebas realizadas a la aplicación SCATIRTPVideoClient de la que se han obtenido datos separados para descompresión y compresión, así como para vídeos con un códec H.264 y MPEG-4. También se han tomado medidas con un número distinto de cámaras (4, 8, 16, 32, 64 y 100) y un tamaño de imagen diferente (4CIF, 1Mpx, 5Mpx). Para la versión de 32 bits, se han tomado las medidas de los datos límite (100 cámaras, 5Mpx), que nos dan una referencia clara del límite rendimiento para nuestra aplicación.

## 5.2.1. H264 - MPEG-4. Compresión y Descompresión

En la figura 21, podemos observar las medidas obtenidas de la aplicación ScatiRTPVideoClient para la versión de 64 bits, con distintos números de cámaras, distintos tamaños de imagen (4CIF, 1Mpx y 5Mpx) y tanto para el códec H.263 como MPEG-4. Las medidas más relevantes a tener en cuenta son: FPS, Bitrate, uso de CPU y memoria.

| Códec | Resolución | Cámaras | FPS totales | FPS cámara | Bitrate total | Bitrate cámara | CPU total | CPU cámara | Memoria total | Memoria cámara | CPU compresión | FPS totales compresión (estimados) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H.264 | 4CIF | 4 | 100 | 25 | 728 | 182 | 25 | 6,25 | 111 | 27,75 | 20 | 100 |
| H.264 | 4CIF | 8 | 199 | 24,875 | 1444 | 180,5 | 72 | 9 | 186 | 23,25 | 61 | 235 |
| H.264 | 4CIF | 16 | 283 | 17,6875 | 2040 | 127,5 | 94 | 5,875 | 243 | 15,1875 | 76 | 350 |
| H.264 | 4CIF | 32 | 284 | 8,875 | 2088 | 65,25 | 95 | 2,96875 | 588 | 18,375 | 74 | 365 |
| H.264 | 4CIF | 64 | 276 | 4,3125 | 2038 | 31,84375 | 96 | 1,5 | 1123 | 17,546875 | 76 | 349 |
| H.264 | 4CIF | 100 | 281 | 2,81 | 2107 | 21,07 | 97 | 0,97 | 1734 | 17,34 | 77 | 354 |
| H.264 | 1MP | 4 | 62 | 15,5 | 3067 | 766,75 | 49 | 12,25 | 257 | 64,25 | 42 | 72 |
| H.264 | 1MP | 8 | 82 | 10,25 | 4089 | 511,125 | 94 | 11,75 | 445 | 55,625 | 80 | 96 |
| H.264 | 1MP | 16 | 82 | 5,125 | 4096 | 256 | 95 | 5,9375 | 820 | 51,25 | 80 | 97 |
| H.264 | 1MP | 32 | 84 | 2,625 | 4186 | 130,8125 | 95 | 2,96875 | 1561 | 48,78125 | 79 | 101 |
| H.264 | 1MP | 64 | 85 | 1,328125 | 3995 | 62,421875 | 96 | 1,5 | 3033 | 47,390625 | 80 | 102 |
| H.264 | 1MP | 100 | 84 | 0,84 | 4312 | 43,12 | 98 | 0,98 | 4691 | 46,91 | 82 | 100 |
| H.264 | 5MP | 4 | 19 | 4,75 | 2512 | 628 | 50 | 12,5 | 798 | 199,5 | 41 | 23 |
| H.264 | 5MP | 8 | 25 | 3,125 | 3475 | 434,375 | 94 | 11,75 | 1448 | 181 | 76 | 31 |
| H.264 | 5MP | 16 | 26 | 1,625 | 3275 | 204,6875 | 94 | 5,875 | 2317 | 144,8125 | 75 | 33 |
| H.264 | 5MP | 32 | 25 | 0,78125 | 3626 | 113,3125 | 98 | 3,0625 | 5663 | 176,96875 | 76 | 32 |
| H.264 | 5MP | 64 | 25 | 0,390625 | 3387 | 52,921875 | 97 | 1,515625 | 11272 | 176,125 | 75 | 32 |
| H.264 | 5MP | 100 | 26 | 0,26 | 3708 | 37,08 | 97 | 0,97 | 14230 | 142,3 | 75 | 34 |
| MPEG-4 | 4CIF | 4 | 100 | 25 | 641 | 160,25 | 7 | 1,75 | 68 | 17 | 4 | 100 |
| MPEG-4 | 4CIF | 8 | 199 | 24,875 | 1278 | 159,75 | 16 | 2 | 100 | 12,5 | 11 | 289 |
| MPEG-4 | 4CIF | 16 | 397 | 24,8125 | 2567 | 160,4375 | 43 | 2,6875 | 140 | 8,75 | 34 | 502 |
| MPEG-4 | 4CIF | 32 | 772 | 24,125 | 4993 | 156,03125 | 93 | 2,90625 | 261 | 8,15625 | 60 | 1197 |
| MPEG-4 | 4CIF | 64 | 766 | 11,96875 | 5145 | 80,390625 | 92 | 1,4375 | 462 | 7,21875 | 55 | 1281 |
| MPEG-4 | 4CIF | 100 | 771 | 7,71 | 4960 | 49,6 | 91 | 0,91 | 723 | 7,23 | 53 | 1324 |
| MPEG-4 | 1MP | 4 | 99 | 24,75 | 1601 | 400,25 | 28 | 7 | 135 | 33,75 | 20 | 139 |
| MPEG-4 | 1MP | 8 | 200 | 25 | 3207 | 400,875 | 75 | 9,375 | 219 | 27,375 | 57 | 263 |
| MPEG-4 | 1MP | 16 | 248 | 15,5 | 3974 | 248,375 | 93 | 5,8125 | 354 | 22,125 | 69 | 334 |
| MPEG-4 | 1MP | 32 | 254 | 7,9375 | 4014 | 125,4375 | 95 | 2,96875 | 653 | 20,40625 | 61 | 396 |
| MPEG-4 | 1MP | 64 | 250 | 3,90625 | 4083 | 63,796875 | 92 | 1,4375 | 1272 | 19,875 | 58 | 397 |
| MPEG-4 | 1MP | 100 | 252 | 2,52 | 3883 | 38,83 | 95 | 0,95 | 1927 | 19,27 | 50 | 479 |
| MPEG-4 | 5MP | 4 | 42 | 10,5 | 4608 | 1152 | 50 | 12,5 | 361 | 90,25 | 34 | 62 |
| MPEG-4 | 5MP | 8 | 57 | 7,125 | 6205 | 775,625 | 94 | 11,75 | 648 | 81 | 61 | 88 |
| MPEG-4 | 5MP | 16 | 53 | 3,3125 | 5735 | 358,4375 | 94 | 5,875 | 1240 | 77,5 | 59 | 84 |
| MPEG-4 | 5MP | 32 | 57 | 1,78125 | 6071 | 189,71875 | 96 | 3 | 2400 | 75 | 56 | 98 |
| MPEG-4 | 5MP | 64 | 57 | 0,890625 | 6264 | 97,875 | 98 | 1,53125 | 4589 | 71,703125 | 63 | 89 |
| MPEG-4 | 5MP | 100 | 59 | 0,59 | 5898 | 58,98 | 92 | 0,92 | 7091 | 70,91 | 48 | 113 |

**Figura 21. Rendimiento H.264, MPEG-4 en descompresión y compresión.**

## Frames máximos

En las Figuras 22 y 23, podemos observar los frames por segundo que es capaz de descomprimir la aplicación en su versión de 64 bits para distinto número de cámaras y tamaños de imagen. La gráfica para 32 bits es muy similar y no la mostramos. Podemos observar que a partir de cierto límite, este valor alcanza su máximo y se mantiene constante pese a que incrementemos el número de cámaras. Esto nos indica el máximo que podemos obtener para cada tamaño de imagen. El factor limitante principal es el tamaño de la imagen: con imágenes de 5Mpx el programa satura con el mínimo número de cámaras.

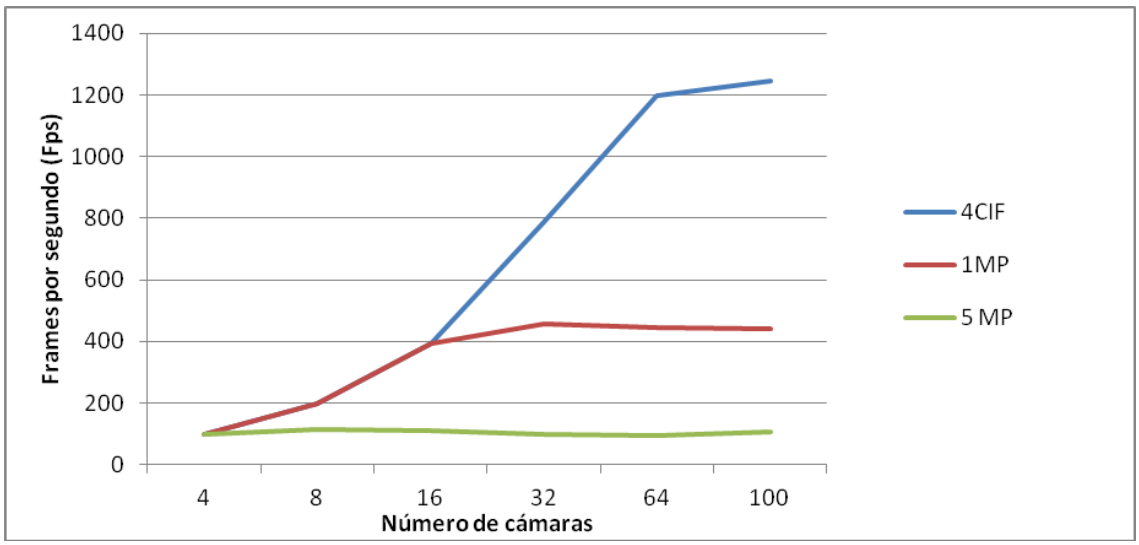



**Figura 22. Fps H.264, en descompresión para la versión de 64 bits.**

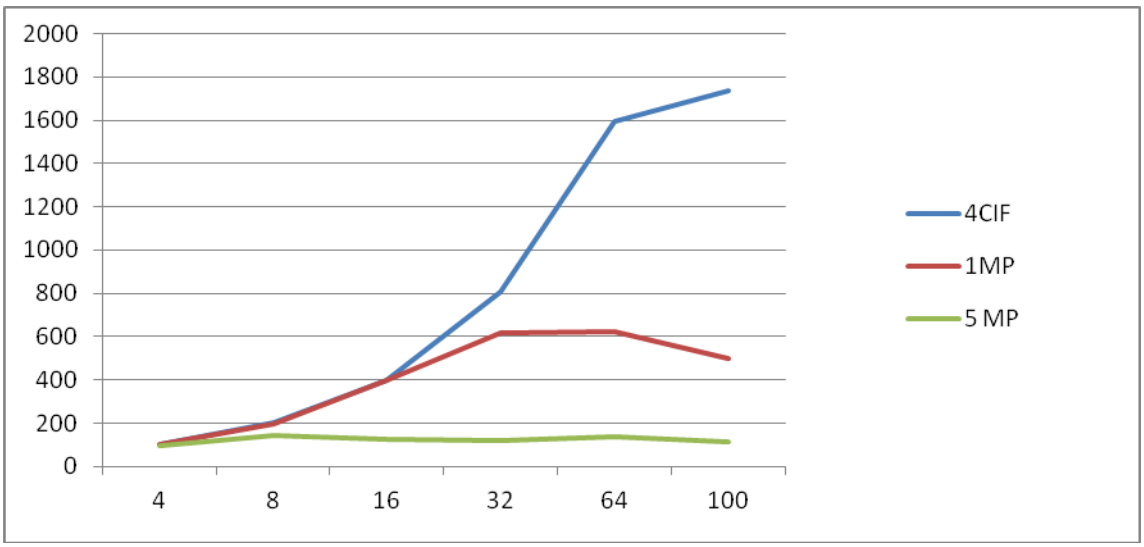



**Figura 23. Fps MPEG-4, en descompresión para la versión de 64 bits.**

## Uso de CPU

En las Figuras 24 y 35 se muestra el uso de CPU para la versión de 64 bits. Observamos que a partir de cierto número de cámaras, para cada tamaño de imagen, el porcentaje de uso de la CPU se mantiene alrededor del 90%. Esto nos indica que cuando llegamos a este valor, el incremento del número de cámaras simultáneas hace que el rendimiento de la aplicación se degrade. Nótese que este valor nunca llegará al 100% debido a que el ordenador siempre estará ejecutando un mínimo de operaciones en el 10% restante.

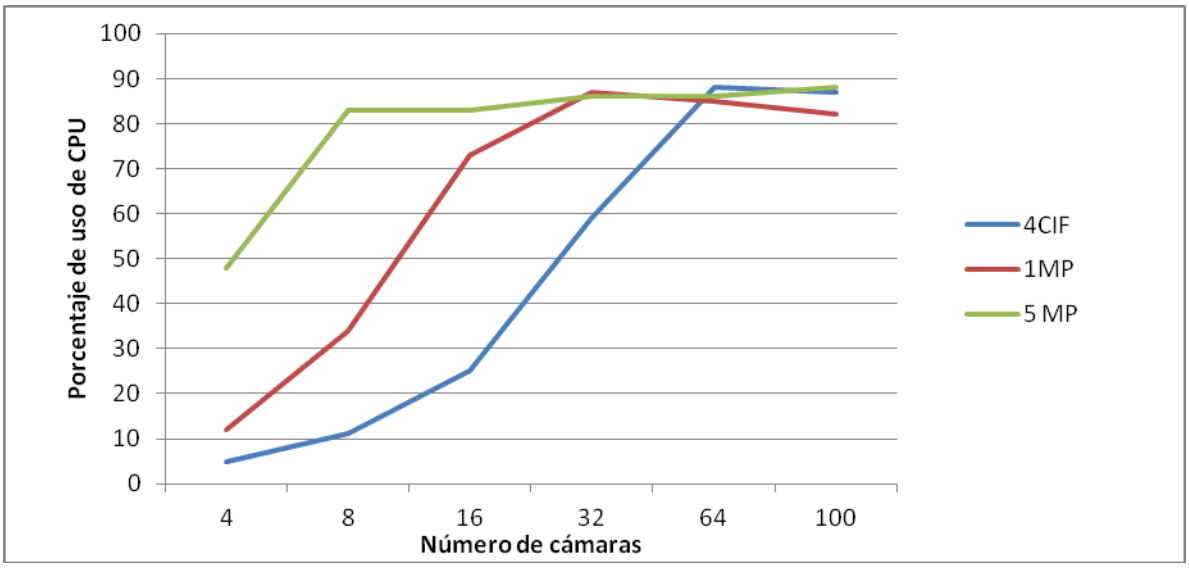



**Figura 24. Uso de CPU en H.264 para la versión de 64 bits.**

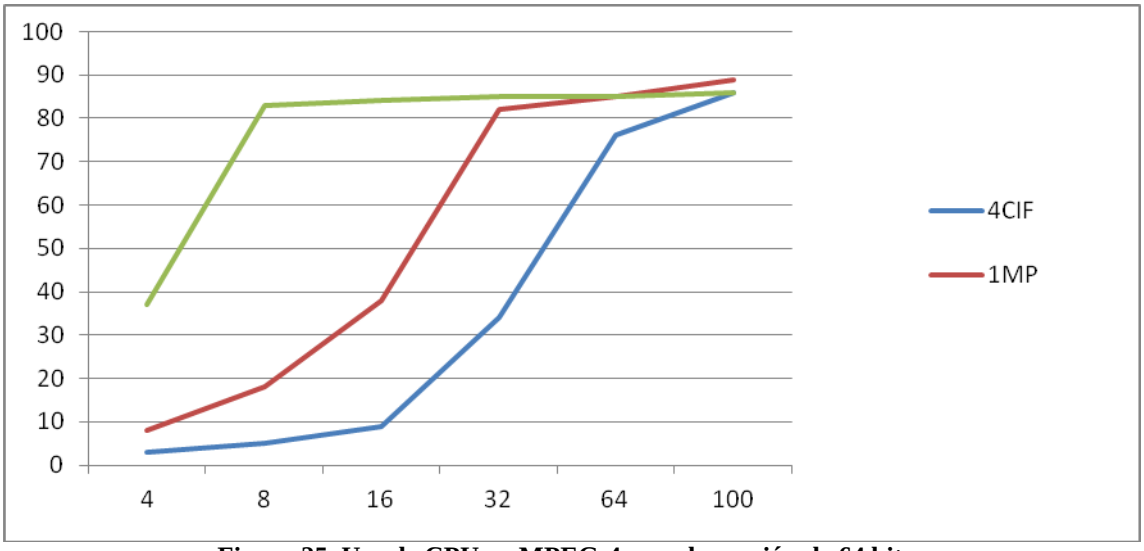



**Figura 25. Uso de CPU en MPEG-4 para la versión de 64 bits.**

# Capítulo 6

## Conclusiones y trabajo futuro

La realización de una migración a 64 bits está fuertemente condicionada por la tecnología usada, es decir, por las características de los componentes de software empleados en el programa a migrar. Puede ser un proceso que se realice simplemente compilando lo ya existente para plataformas de 64 bits, con el posterior enlazado a sus correspondientes librerías de 64 bits, o puede ser un proceso mucho más complejo, en el que haya que comenzar por el cambio de versión de las librerías implicadas, su correcta revisión y análisis estático así como su compilación para entornos de 64 bits. Pueden surgir problemas derivados del código que implique el cambio de código legado, con todos los inconvenientes que esto conlleva. En este Proyecto se ha llevado a cabo un análisis pormenorizado de todos estos aspectos, enfocado a una aplicación en explotación (VisionSurfer 4.6, SCATI LABS.). La estimación resultante del proceso de migración ha sido de 4 meses (Capítulo 4).

Respecto al rendimiento observado en nuestra aplicación de 64 bits, la mejora producida es leve, consiguiendo incluso resultados peores en ciertos casos. Esto es debido al uso de la librería de las IPP optimizadas tanto para 32 como para 64 bits, y que soportan aproximadamente un 90% del tiempo de ejecución total. Además, como era de esperar el programa generado ocupa un mayor tamaño debido al crecimiento de los tipos de datos como punteros, y la duplicación de los primeros cuatro parámetros en una invocación a función.

El principal beneficio de la migración a 64 bits que podemos reseñar es la posibilidad de manejar cantidades de memoria superiores a 4GB (límite de los SO de 32 bits). Esto nos ha permitido poder manejar más flujos de vídeo de entrada en la aplicación, elevando el número de cámaras simultáneas hasta 300 aproximadamente, respecto a las 64 que podíamos manejar en su versión de 32 bits. Pese a que en un principio se esperaba poder mejorar el rendimiento, la necesidad real, en función de las necesidades del mercado, era poder conectar más cámaras de las permitidas hasta este momento.

Respecto a las líneas de desarrollo seguidas a partir del resultado de este proyecto, se ha seguido con la migración de otras aplicaciones implicadas en *VisionSurfer*. Esta aplicaciones por el momento han presentado problemas similares y han mostrado resultados de rendimiento parecidos, una vez más la posibilidad de una mayor manejo de memoria ha sido la mejora más destacable, si bien es verdad que en otras aplicaciones y para caso concretos se han obtenido mejores resultados que en la versión de 32 bits. Esto puede deberse a la supuesta mejora obtenida al tener un mayor número de registros,que disminuye el *spill code*, y por el paso de parámetros por registro en lugar de memoria (pila) como se mostró en el Capítulo 2.

# ANEXO 1:

# CÓDIGO ENSAMBLADOR APLICACIÓN HELLO WORLD 32 Y 64 BITS

A continuación se muestra el código ensamblador para las distintas versiones del programa *Hello World*.

- Plataforma *x86 Debug*.

```
;	COMDAT _wmain
_TEXT	SEGMENT
_argc$ = 8                                      ; size = 4
_argv$ = 12                                     ; size = 4
_wmain PROC                                     ; COMDAT

; 9	: {

	push	ebp
	mov	ebp, esp
	sub	esp, 192                        ; 000000c0H
	push	ebx
	push	esi
	push	edi
	lea	edi, DWORD PTR [ebp-192]
	mov	ecx, 48                                 ; 00000030H
	mov	eax, -858993460                         ; ccccccccH
	rep stosd

; 10	:	std::cout << "Hello World!!!" << std::endl;

	mov	esi, esp
	mov	eax, DWORD PTR
__imp_?endl@std@@YAAAV?$basic_ostream@DU?$char_traits@D@std@@@1@AAV21@@Z
	push	eax
	push	OFFSET ??_C@_0P@MKFFDJMN@Hello?5World?$CB?$CB?$CB?$AA@
	mov	ecx, DWORD PTR
__imp_?cout@std@@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A
	push	ecx
	call
	??$?6U?$char_traits@D@std@@@std@@YAAAV?$basic_ostream@DU?$char_traits@D
@std@@@0@AAV10@PBD@Z ; std::operator<<<std::char_traits<char> >
	add	esp, 8
	mov	ecx, eax
	call	DWORD PTR
__imp_??6?$basic_ostream@DU?$char_traits@D@std@@@std@@QAEAAV01@P6AAAV01@AAV01@
@Z@Z
	cmp	esi, esp
	call	__RTC_CheckEsp

; 11	:	return 0;

	xor	eax, eax

; 12	: }

	pop	edi
	pop	esi
	pop	ebx
	add	esp, 192                        ; 000000c0H
	cmp	ebp, esp
	call	__RTC_CheckEsp
	mov	esp, ebp
	pop	ebp
	ret	0
	_wmain ENDP
```

- Plataforma *x86 Release*.

```
;	COMDAT _wmain
_TEXT	SEGMENT
_argc$ = 8                                      ; size = 4
_argv$ = 12                                     ; size = 4
_wmain PROC                                     ; COMDAT

; 10   :     std::cout << "Hello World!!!" << std::endl;

       mov    eax, DWORD PTR
__imp_?endl@std@@YAAAV?$basic_ostream@DU?$char_traits@D@std@@@1@AAV21@@Z
       mov    ecx, DWORD PTR
__imp_?cout@std@@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A
       push   eax
       push   ecx
       call
       ??$?6U?$char_traits@D@std@@@std@@YAAAV?$basic_ostream@DU?$char_traits@D
@std@@@@AAV10@PBD@Z ; std::operator<<<std::char_traits<char> >
       add    esp, 4
       mov    ecx, eax
       call   DWORD PTR
__imp_??6?$basic_ostream@DU?$char_traits@D@std@@@std@@QAEAAV01@P6AAAV01@AAV01@
@Z@Z

; 11   :     return 0;

       xor    eax, eax

; 12   : }

       ret    0
       _wmain ENDP
```

- Plataforma *x64 Debug*.

```
_TEXT	SEGMENT
argc$ = 48
argv$ = 56
wmain	PROC

; 9    : {

$LN3:
       mov    QWORD PTR [rsp+16], rdx
       mov    DWORD PTR [rsp+8], ecx
       push   rdi
       sub    rsp, 32                                  ; 00000020H
       mov    rdi, rsp
       mov    rcx, 8
       mov    eax, -858993460                          ; ccccccccH
       rep stosd
       mov    ecx, DWORD PTR [rsp+48]

; 10   :     std::cout << "Hello World!!!" << std::endl;

       lea    rdx, OFFSET FLAT:$SG20069
       mov    rcx, QWORD PTR
__imp_?cout@std@@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A
       call
       ??$?6U?$char_traits@D@std@@@std@@YAAEAV?$basic_ostream@DU?$char_traits@
D@std@@@@AEAV10@PEBD@Z ; std::operator<<<std::char_traits<char> >
       mov    rdx, QWORD PTR
__imp_?endl@std@@YAAEAV?$basic_ostream@DU?$char_traits@D@std@@@1@AEAV21@@Z
       mov    rcx, rax
       call   QWORD PTR
__imp_??6?$basic_ostream@DU?$char_traits@D@std@@@std@@QEAAAEAV01@P6AAEAV01@AEA
V01@@Z@Z

; 11   :     return 0;

       xor    eax, eax
```

```
; 12   : }

       add    rsp, 32                                 ; 00000020H
       pop    rdi
       ret    0
wmain  ENDP
```

- Plataforma *x64 Release*.

```
;      COMDAT wmain
_TEXT  SEGMENT
argc$ = 48
argv$ = 56
wmain  PROC                                    ; COMDAT

; 9    : {

$LN3:
       sub    rsp, 40                                 ; 00000028H

; 10   :     std::cout << "Hello World!!!" << std::endl;

       mov    rcx, QWORD PTR
__imp_?cout@std@@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A
       lea    rdx, OFFSET FLAT:??_C@_0P@MKFFDJMN@Hello?5World?$CB?$CB?$CB?$AA@
       call
       ??$?6U?$char_traits@D@std@@@std@@YAAEAV?$basic_ostream@DU?$char_traits@
D@std@@@@AEAV10@PEBD@Z ; std::operator<<<std::char_traits<char> >
       mov    rdx, QWORD PTR
__imp_?endl@std@@YAAEAV?$basic_ostream@DU?$char_traits@D@std@@@1@AEAV21@@Z
       mov    rcx, rax
       call   QWORD PTR
__imp_??6?$basic_ostream@DU?$char_traits@D@std@@@std@@QEAAAEAV01@P6AAEAV01@AEA
V01@@Z@Z

; 11   :     return 0;

       xor    eax, eax

; 12   : }

       add    rsp, 40                                 ; 00000028H
       ret    0
wmain  ENDP
```

# ANEXO 2:

# WINDOWS 32 BITS ON WINDOWS 64 BITS

Como ya hemos dicho antes, WOW64 es un subsistema de Windows que permite ejecutar procesos de 32 bits en un entorno de Windows 64 bits.

Cuando se lanza una aplicación de 32 bits, lo primero que se ejecuta es el lanzador nativo de librerías. Este reconoce que es un proceso de 32 bits y lo trata de una manera especial. Se configura un entorno de emulación WOW64 para los procesos de 32 bits y se transfiere el control al cargador de 32 bits, en Ntdll.dll.

La capa de emulación de WOW64 se ejecuta entre la aplicación de 32 bits y la dll de 64 bits Ntdll.dll y traduce las llamadas de la aplicación a Ntdll.dll de 32 bits. a la Ntdll.dll de 64 bits. Las llamadas de retorno se traducen de forma similar.

Algunas de las limitaciones de WOW64 son:

- El espacio de direcciones está limitado a 2gB por defecto y 4GB si usamos /LARGEADDRESSAWARE.
- Un proceso de 32 bits no puede cargar una DLL de 64 bits (excepto para ciertas DLLs del sistema)
- No se pueden ejecutar procesos de 16 bits.
- El API de la *Virtual DOS Machine (VDM)* está desactivada.
- Otras limitaciones en arquitecturas *Itanium.*

La redirección del registro permite al código de 32 bits acceder a los registros apropiados en las máquinas de 64 bits. La redirección divide el registro en nodos de 32 y 64 bits.

La redirección de archivos del sistema permite al código de 32 bits usar nombres de fichero y rutas que hacen referencia a datos o módulos de 64 bits.

Muchas veces los desarrolladores escriben en el código directamente las rutas en sus aplicaciones. Por esto para mantener la compatibilidad de las aplicaciones el sistema de ficheros de 64 bits se llama todavía *Ssytem32.*

Esta redirección esta activada por defecto en WOW64. Para desactivarla se puede usar la función *Wow64DisableWow64FsRedirection().* Para volver a activarla hay que usar la función *Wow64RevertWow64FsRedirection().* Esto sólo se aplica al hilo que hace la llamada a la función.

Si usamos DLLs que ambos clientes (32 y 64 bits) necesitan, hay que manejar las referencias a las DLLs para asegurar que no se usa la versión correcta.

Podemos usar una tecnología como COM que automáticamente las une.

Dejar las DLLs en una carpeta y añadir esa ruta a la variable de entorno *PATH.* Como norma general hay que nombrar las DLLs de 32 y 64 bits. Si las nombramos igual, debemos ponerlas en un directorio distinto cada una. Para 64 bits es: *C:\Windows\System32\Dll.dll* y *C:\Windows\Syswow64\Dll.dll* para 32 bits.

# ANEXO 3:

# RESULTADOS DE LA COMPILACIÓN IPP 6.1 (X64)

Esta es la salida producida por las IPP versión 6.1 al ser compiladas para 64 bits con Visual Studio 2008.

```
------------------------------------------------------------------------

Compiling...
umc_mp4_parser_w.cpp
.\src\codec\mpeg4_mux\umc_mp4_parser_w.cpp(309) : warning C4267:
'=' : conversion from 'size_t' to 'Ipp32s', possible loss of data

umc_mp4_mux_atoms.cpp
.\src\codec\mpeg4_mux\umc_mp4_mux_atoms.cpp(738) : warning C4267:
'=' : conversion from 'size_t' to 'Ipp32s', possible loss of data
.\src\codec\mpeg4_mux\umc_mp4_mux_atoms.cpp(742) : warning C4244:
'=' : conversion from '__int64' to 'Ipp32s', possible loss of data
.\src\codec\mpeg4_mux\umc_mp4_mux_atoms.cpp(799) : warning C4267:
'=' : conversion from 'size_t' to 'Ipp32u', possible loss of data
.\src\codec\mpeg4_mux\umc_mp4_mux_atoms.cpp(837) : warning C4244:
'=' : conversion from '__int64' to 'Ipp32s', possible loss of data

.........................

umc_video_resizing.cpp
.\src\codec\color_space_converter\umc_video_resizing.cpp(81) :
warning C4267: 'argument' : conversion from 'size_t' to 'int',
possible loss of data
.\src\codec\color_space_converter\umc_video_resizing.cpp(81) :
warning C4267: 'argument' : conversion from 'size_t' to 'int',
possible loss of data
.\src\codec\color_space_converter\umc_video_resizing.cpp(97) :
warning C4267: 'argument' : conversion from 'size_t' to 'int',
possible loss of data


------------------------------------------------------------------------
```

Podemos ver en las sentencias subrayadas en Amarillo como el *warning* es todo el rato el mismo, la conversión de un tipo de 64-bits como pueden ser *size_t o __int64* a tipos de 32-bits como Ipp32s o int, produce una pérdida de datos ya que el tamaño de destino es menor que el de origen. Para ellos habrá que cambiar los tipos de 32 bits para que sean del tamaño de la arquitectura. De esta forma evitaremos estos problemas que pueden hacer que la aplicación falle por completo.

El mayor problema reside en lo tedioso que puede resultar el cambio de los tipos implicados. Esto es debido a que el cambio de un tipo de dato, puede hacer que aparezcan nuevos problemas en ficheros que hasta ahora parecían correctos, por lo que no podemos hacernos una idea exacta del tiempo que puede llevarnos, ya que se

pueden ir generando nuevos errores conforme avanzamos en el proceso de cambio de los tipos para que todo funcione correctamente.

# ANEXO 4:

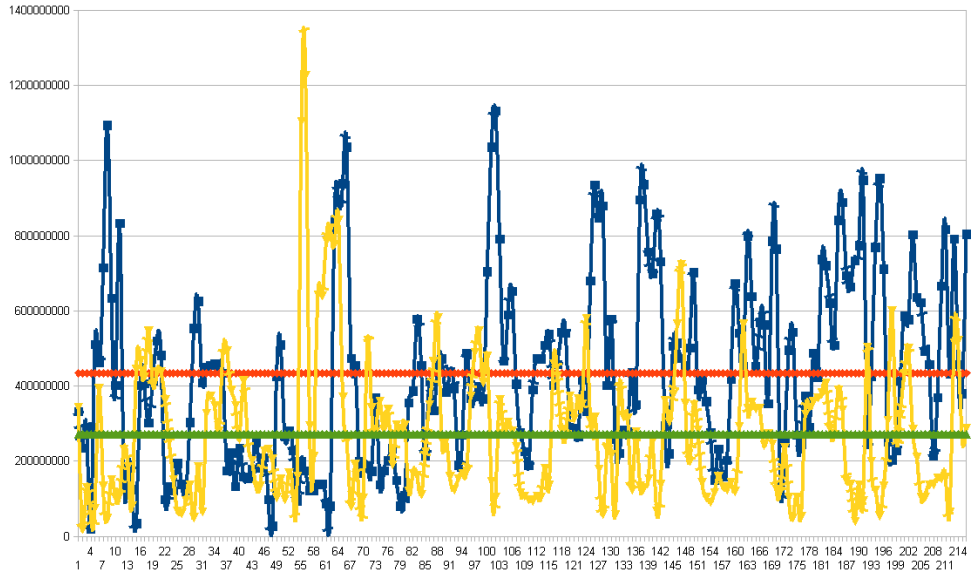
# MEJORAS POR EL USO DE LA LIBRERÍA IPP

Además de las funciones que se describen en el proyecto pertenecientes a las IPP, estas ofrecen también una reescritura de las funciones del sistema más comunes. Debido a la optimización que realiza Intel sobre todas las funciones incluidas en su librería IPP, podemos esperar que estas funciones también estén optimizadas y por tanto, ofrezcan mejor rendimiento que las que estamos acostumbrados a usar normalmente. Las funciones del sistema optimizadas por Intel son:

- Funciones para el manejo de hilos: *create(), wait(), set_priority(), close(), …*
- Funciones para el manejo de sockets: *select(), next(), create(), accept(), write(), close(), …*
- Funciones para el manejo de semáforos y *mutex*, muy parecidas a las funciones usadas en los hilos.
- Funciones para el uso de archivos: *fseek(), ftell(), fopen(), fclose(), fread(), fwrite(), fgets(), fputs(), fscanf(), fprintf(), …*

Tras analizar el comportamiento de nuestra aplicación de prueba, hemos visto claramente que la operación de escribir a disco los vídeos generados y la operación de escritura de cada frame en un archivo de imagen *bmp*, representan la mayoría del tiempo que usa la aplicación en su ejecución. Por tanto si fuésemos capaces de optimizar la operación de escritura en fichero, podríamos obtener una mejora global en el rendimiento de la aplicación que fuese sustancial.

Para ello hemos probado a cambiar la función con la que escribimos que es: *fwrite()*, por su equivalente en la librería de las IPP *vm_file_fwrite()* y el resultado ha sido el siguiente.




Como se puede observar en azul tenemos el número de *clocks* de la CPU para la llamada *fwrite()* para cada imagen y en rojo tenemos el número medio que usaremos para comparar con la versión de las IPP. En la versión que hemos usado la función de las IPP *vm_file_fwrite()* que a priori está optimizada, efectivamente comprobamos que esto es así, podemos apreciar en el color amarillo que refleja lo mismo que el azul pero para esta función y en la media (en color verde) que es sustancialmente menor. El *SpeedUp* conseguido con este cambio tan pequeño es de 1,6. Dado que esta operación se realiza por cada frame del archivo de video que estamos decodificando, esto supone un ahorro de tiempo considerable a tener muy en cuenta, y que se puede traducir en la posibilidad de tratar algún video más simultáneamente.

# ANEXO 5

# WARNINGS PVS-STUDIO

A continuación se detallan los diferentes tipos de avisos producidos por la herramienta de análisis estático de código PVS-Studio.

Para un mayor entendimiento de cada error, se pude acceder a esta URL, cambiando *Code* por el código de cada aviso. http://www.viva64.com/en/*Code*

- **V101:** El analizador ha detectado un error potencial debido a una conversión implícita en una operación de asignación. Esto puede resultar en un error al calcular la expresión de la asignación.

```
size_t a;
unsigned b;
...
a = b; // V101
```

- **V102:** El analizador ha encontrado un posible error en aritmética de punteros. El error puede ser causado por un *overflow* al determinar el valor de la expresión.

```
short a16, b16, c16;
char *pointer;
...
pointer += a16 * b16 * c16;
```

- **V103:** El analizador ha encontrado un posible error relacionado con la conversión implícita de un tipo del tamaño de la memoria a uno de 32-bits. El error consiste en la pérdida de los 32 bits más significativos.

```
size_t Width, Height, FrameCount;
...
unsigned BufferSizeForWrite = Width * Height * FrameCount *
sizeof(RGBStruct);
```

- **V104:** El analizador ha encontrado un posible error dentro de una operación aritmética relacionado con la conversión implícita a un tipo del tamaño de la memoria.

```
size_t n;
unsigned i;
// Infinite loop (n > UINT_MAX).
for (i = 0; i != n; ++i) { ... }
```

- **V106:** El analizador ha encontrado un posible error con una conversión implícita de un argumento de la función a un tipo del tamaño de la memoria.

```
CArray<int, int> myArray;
...
int invalidIndex = 0;
INT_PTR validIndex = 0;
while (validIndex != myArray.GetSize()) {
  myArray.SetAt(invalidIndex, 123);
  ++invalidIndex;
  ++validIndex;
}
```

- **V108:** El analizador ha encontrado un posible error en la indexación de un vector, debido al uso de un tipo de tamaño inferior al de la memoria, provocando que quizás no se pudiese indexar entero.

```
extern char *longString;
extern bool *isAlnum;
...
unsigned i = 0;
while (*longString) {
  isAlnum[i] = isalnum(*longString++);
  ++i;
}
```

- **V110:** El analizador ha encontrado un posible error relacionado con la conversión implícita del valor devuelto. El error provoca que se pierdan los 32 bits más significativos del valor devuelto de 64 bits.

```
extern char *begin, *end;
unsigned GetSize() {
  return end - begin;
}
```

- **V111:** El analizador ha encontrado un posible error relacionado con la transferencia de uno de los argumentos del tipo de tamaño de memoria en la función con un número variable de argumentos. Esto puede provocar que la función tome como parámetro algo que no lo es, es decir, coja el final o principio de un argumento como el siguiente debido al cambio de tamaño.

```
const char *invalidFormat = "%u";
size_t value = SIZE_MAX;
printf(invalidFormat, value);
```

- **V112:** El analizador ha encontrado el uso de un número mágico. El posible error puede ser que se use como el tamaño asumido para arquitecturas de 32 bits, y provoque un error debido al cambio de tamaño en arquitecturas de 64 bits.

```
size_t ArraySize = N * 4;
size_t *Array = (size_t *)malloc(ArraySize);
```

- **V113:** El analizador ha encontrado un posible error relacionado con la conversión implícita de un tipo del tamaño de memoria a un tipo doublé o viceversa. El posible error consiste en no poder guardar la totalidad del valor en un tipo *double*.

```
SIZE_T size = SIZE_MAX;
double tmp = size;
size = tmp; // x86: size == SIZE_MAX
            // x64: size != SIZE_MAX
```

- **V117:** El analizador ha encontrado un posible error relacionado con el uso de un tipo del tamaño de memoria en una unión. Esto provoca que no se pueda almacenar todo el dato, ya que el espacio del tipo sin signo es menor que el de un puntero.

```
union PtrNumUnion {
  char *m_p;
  unsigned m_n;
} u;
...
u.m_p = str;
u.m_n += delta;
```

- **V119:** El analizador ha detectado una expresión aritmética no segura que contiene varias operaciones *sizeof()*. Estas expresiones pueden devolver valores incorrectos debido a que no tienen en cuenta el alineamiento en estructuras para 64 bits.

```
struct MyBigStruct {
  unsigned m_numberOfPointers;
  void *m_Pointers[1];
};
size_t n2 = 1000;
void *p;
p = malloc(sizeof(unsigned) + n2 * sizeof(void *));
```

- **V121:** El analizador ha detectado un error potencial relacionado con la llamada al operador new. Un valor que no es del tipo del tamaño de la memoria se le pasa a este operador como argumento. El operador new coge el valor size_t y pasa un tipo de 32-btis puede provocar un overflow.

```
unsigned a = 5;
unsigned b = 1024;
unsigned c = 1024;
unsigned d = 1024;
char *ptr = new char[a*b*c*d]; //V121
```

- **V204:** Este aviso informa sobre una conversión explícita de un tipo de 32 bits a uno de 64 bits.

```
int n;
float *ptr;
...
ptr = (float *)(n);
```

- **V220:** Este aviso nos informa sobre una secuencia extraña de conversiones. Un tipo de 64 bits es convertido a 32 bits para posteriormente volver a convertirlo a 64 bits. Esto produce una pérdida de los bits más significativos**.**

```
char *p1;
char *p2;
ptrdiff_t n;
...
n = int(p1 - p2);
```

- **V302:** El analizador ha detectado un error potencial al trabajar con clases que contiene el operador []. Si el operador es una tipo de 32 bits, esto puede ser un error ya que no podríamos indexarlo entero.

```
class MyArray {
  std::vector<float> m_arr;
  ...
  float &operator[](int i)  //V302
  {
    DoSomething();
    return m_arr[i];
  }
} A;
...
int x = 2000;
int y = 2000;
int z = 2000;
A[x * y * z] = 33;
```

- **V510:** En funciones con un número variable de argumentos, sólo tipos de datos básicos pueden pasarse como argumentos. Estos datos planos son:

1. Todos los tipos aritméticos por defecto (incluyendo *wchar_t* y *bool*);
2. Tipos definidos como *enum*.
3. Punteros.
4. Estructuras de tipos de datos planos o uniones que cumplan lo siguiente:
    a. No contener constructores, destructores o asignaciones.
    b. No tengan clases base.
    c. No contengan funciones virtuales.
    d. No contengan miembros privados o protegidos que no sean estáticos.
    e. No contengan miembros no estáticos de tipos de datos no básicos y punteros.

```
wchar_t buf[100];
std::wstring ws(L"12345");
swprintf(buf, L"%s", ws);
```

- **V512:** El analizador ha encontrado un error potencial relacionado con el llenado, copia o comparación de un buffer de memoria. Este error puede causar tanto *overflow* como *underflow* del buffer.

```
#define CONT_MAP_MAX 50
int _iContMap[CONT_MAP_MAX];
memset(_iContMap, -1, CONT_MAP_MAX);
```

- **V524:** Este aviso es generado cuando el analizador detecta dos funciones implementadas de la misma manera. Esto no es un error, pero sería deseable que no ocurriese.

```
class Point
{
  ...
  float GetX() { return m_x; }
  float GetY() { return m_x; }
};
```

- **V547:** El analizador ha detectado un error potencial: una condición es siempre verdadera o falsa. Esto no siempre es un error, pero es conveniente revisar la lógica del programa para comprobarlo.

```
LRESULT CALLBACK GridProc(HWND hWnd,
  UINT message, WPARAM wParam, LPARAM lParam)
{
  ...
  if (wParam<0)
  {
    BGHS[SelfIndex].rows = 0;
  }
  else
  {
    BGHS[SelfIndex].rows = MAX_ROWS;
  }
  ...
}
```

- **V550:** El analizador ha detectado un error potencial en una operación de comparación == o !=, usada con número en coma flotante. Estas comparaciones pueden provocar errores. Es preferible usar una expresión del tipo (A-B) > Epsilon.

```
double a = 0.5;
if (a == 0.5) //OK
  x++;

double b = sin(M_PI / 6.0);
if (b == 0.5) //ERROR
  x++;
```

- **V576:** El analizador ha detectado un error potencial con las funciones (*printf(), sprint(),* etc.)

```
int A = 10;
double B = 20.0;
printf("%i %i\n", A, B);
```

- **V595:** El analizador ha detectado un error potencial que puede causar la pérdida de la referencia de un puntero igual a NULL.

```
buf = Foo();
pos = buf->pos;
if (!buf) return -1;
```

- **V601:** El analizador ha detectado una conversión implícita rara. Este tipo de conversión puede ser una señal de código mal escrito.

```
std::string str;
bool bstr;
...
str = true;
```

- **V628:** El analizador ha detectado un error potencial. Dos if seguidos pueden indicar que uno de ellos debería comentarse, sino la lógica del programa podría verse alterada.

```
if(!hwndTasEdit)
//hwndTasEdit = getTask()
if(hwndTasEdit)
{
 ...
}
```

- **V803:** El analizador ha detectado una construcción que puede ser optimizada. Un iterador que se cambia con la operación posfija, ya que no se usa el elemento anterior, se puede usar la operación infija que es más efectiva.

```
std::vector<size_t>::const_iterator it;
for (it = a.begin(); it != a.end(); it++)
{ ... }
```

- **V807:** el analizador ha detectado un código que puede ser optimizado. El código contiene mensajes homogéneos para accede al mismo objeto.

```
Some->getFoo()->doIt1();
Some->getFoo()->doIt2();
Some->getFoo()->doIt3();
```

- **V808:** El analizador ha detectado código que puede se simplificado. Una función que contiene variables locales que no se usan en ningún sitio.

```
void Foo()
{
  int A[100];
  string B[100];
  DoSomething(A);
}
```

- **V808:** El analizador ha detectado código que puede se simplificado. Una función que contiene variables locales que no se usan en ningún sitio.

```
if (pointer != 0)
  delete pointer;
```

# ANEXO 6:

# TABLA LIBRERÍAS APLICACIÓN VISONSURFER

Tabla con las librerías usadas, su versión y disponibilidad para 32/64 bits y Windows/Linux:

| Librería | Versión actual | Versión más reciente | x86 | x64 | Windows | Linux |
|---|---|---|---|---|---|---|
| **ACE** | 5.7.1.1 (5.7.1) | 6.1.0 | X | X | X | X |
| **Cryptopp** | 5.6.0 | 5.6.1 | X | X | X | X (libcryptopp) |
| **ZLib** | 1.2.3 | 1.2.7 | X | X | X | X |
| **Xercesc** | 3.0.1 | 3.1.1 | X | X | X | X |
| **IPP** | 6.1.3.047 | 7.1.1 | X | X | X | X |
| **Boost** | 1.40.0 | 1.52.0 | X | X | X | X |
| **avifile** | 1.0.0.7 | 1.0.0.7 | X | X | X | X |
| **IPPMedia** | 2.2.0 | ??? (IPP 7.1) | X | X | X | X |
| **ijl** | 2.0 | 2.0 | X | ? | X | |
| **SDL** | 1.2.14 | 1.2.15 | X | X | X | X |
| **SQLite** | 3.6.23 | 3.7.15.2 | X | X | X | X |
| **MySQL** | 3.23 (5.1 Ubuntu) | 5.5.29 | X | X | X | X |

# ANEXO 7:

# HERRAMIENTAS DE ANALISIS DE INTEL

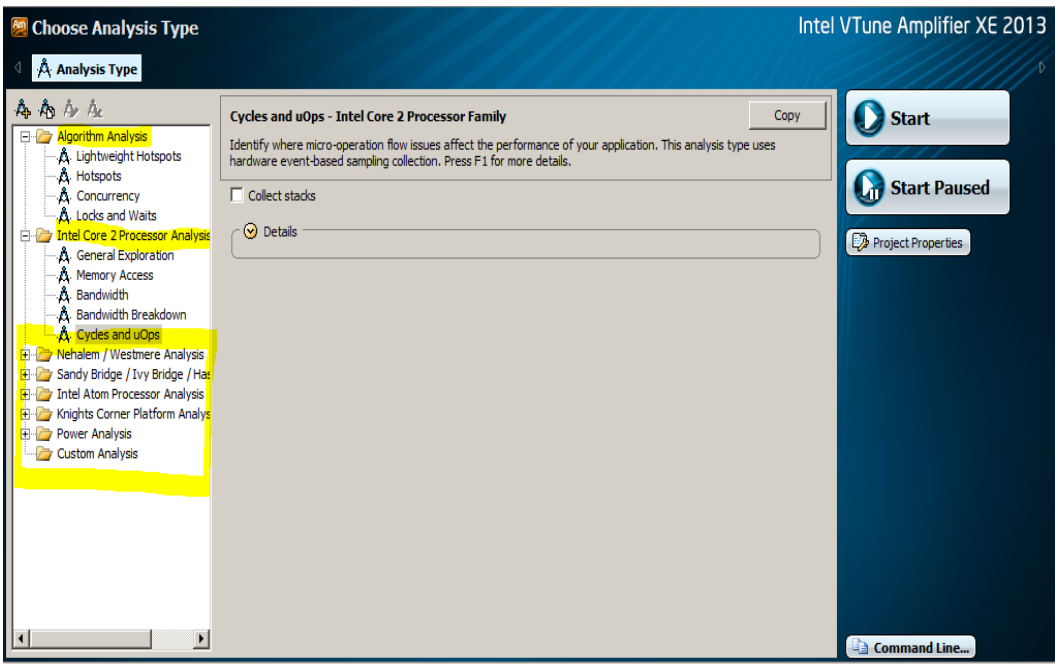
## Intel Composer XE

Con la librería de las *IPP (Integrated Performance Primitives)* contenidas en la suite *Composer XE* (Windows y Linux), vienen una herramientas para el análisis de código que son: *Advisor XE*, *Inspector XE y VTune Amplifier XE*. Estas herramientas durante su instalación quedan integradas en Visual Studio en Windows, pero también se pueden usar por separado en ambos sistemas. Sirven para los lenguajes C++, C#y Fortran, en nuestro caso nos centraremos en C++ que es el que usamos.

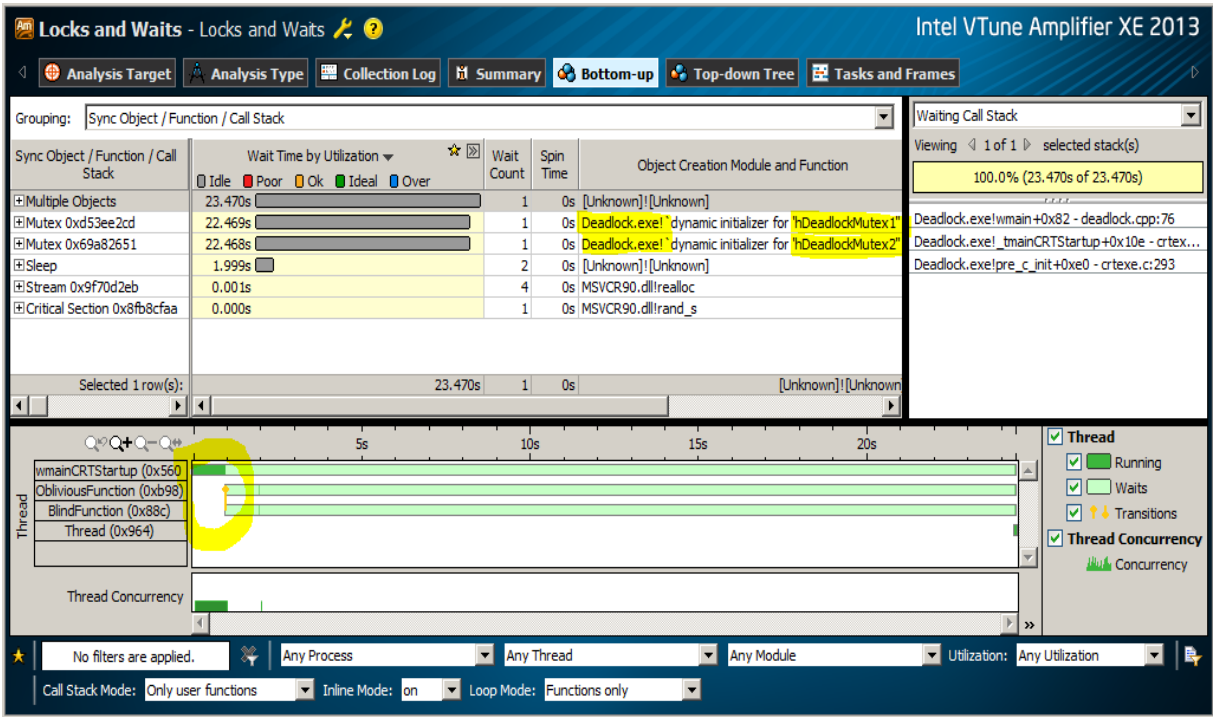
Para todas las herramientas es recomendable hacer el análisis para la versión *Debug* ya que nos permitirá acceder a más información sobre los problemas que nos indique, salvo que se diga que es mejor en *Release*.

## VTune Amplifier XE

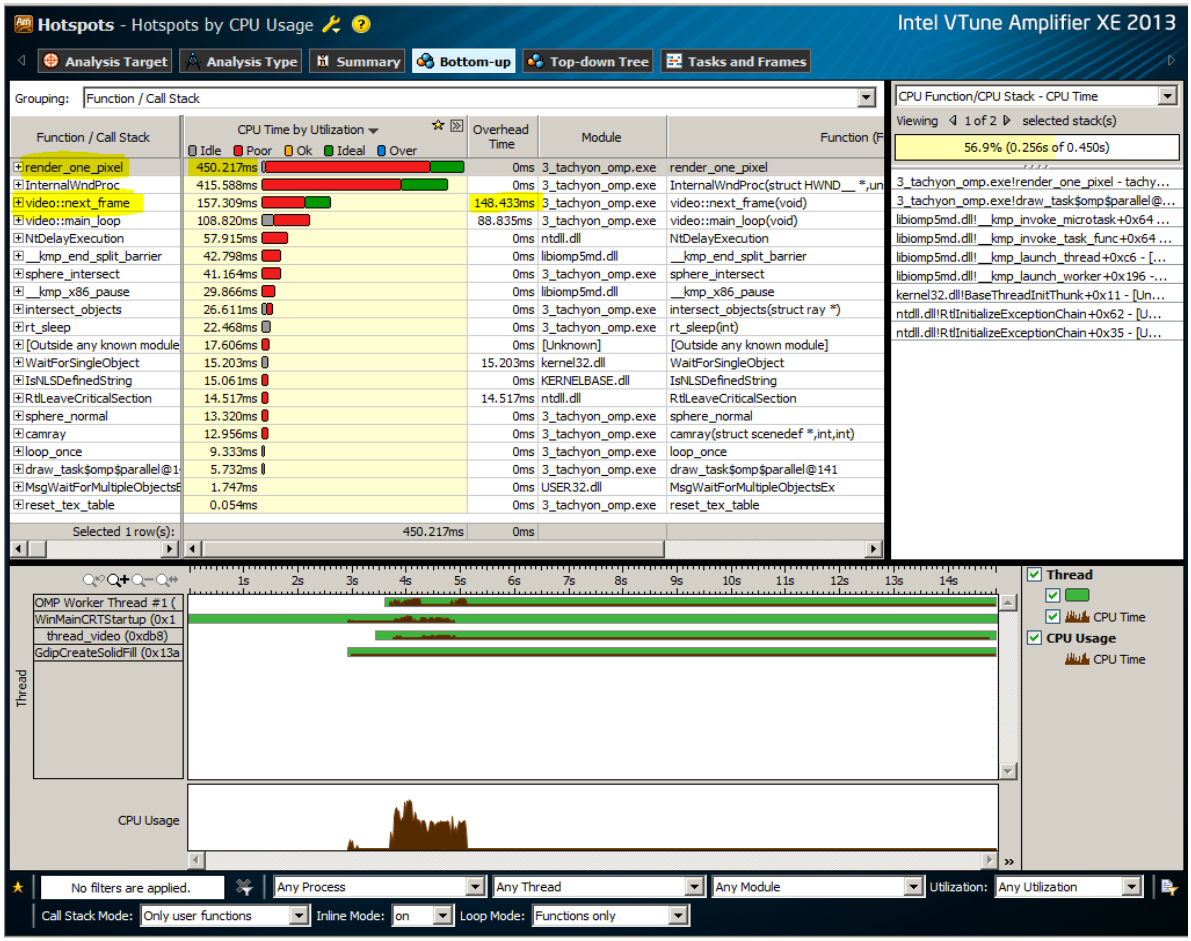
Esta herramienta hay que ejecutarla con permisos de administrador del sistema para poder realizar todos los análisis que ofrece. Estos análisis nos dan datos de muy bajo nivel, como pueden ser los puntos que más tiempo están en ejecución, sincronización y concurrencia entre hilos. Estas son comunes a todas las arquitecturas, pero también ofrece otras medidas especificas de cada arquitectura como pueden ser: acceso a la memoria, ancho de banda de memoria y número de ciclos.



Si sobre un ejemplo que contiene un *deadlock* ejecutamos el análisis de *Locks and Waits*, obtenemos el siguiente resultado:




Otra de las medidas que podemos obtener con esta herramienta son los *Hotspots,* o sitios donde más tiempo está el programa ejecutándose, para un ejemplo que renderiza y muestra una imagen este análisis nos ha dado el siguiente resultado:




Observamos que en la función más costosa es *render_one_pixel()*, puesto que se ejecuta para cada pixel de la imagen. Esta sería la función que deberíamos mejorar si queremos obtener una mejora sustancial en el rendimiento de nuestro programa, ya que al ser la que más tiempo lleva, una pequeña mejora aquí sería más efectiva que una gran mejora en otra función que no use tanto tiempo.

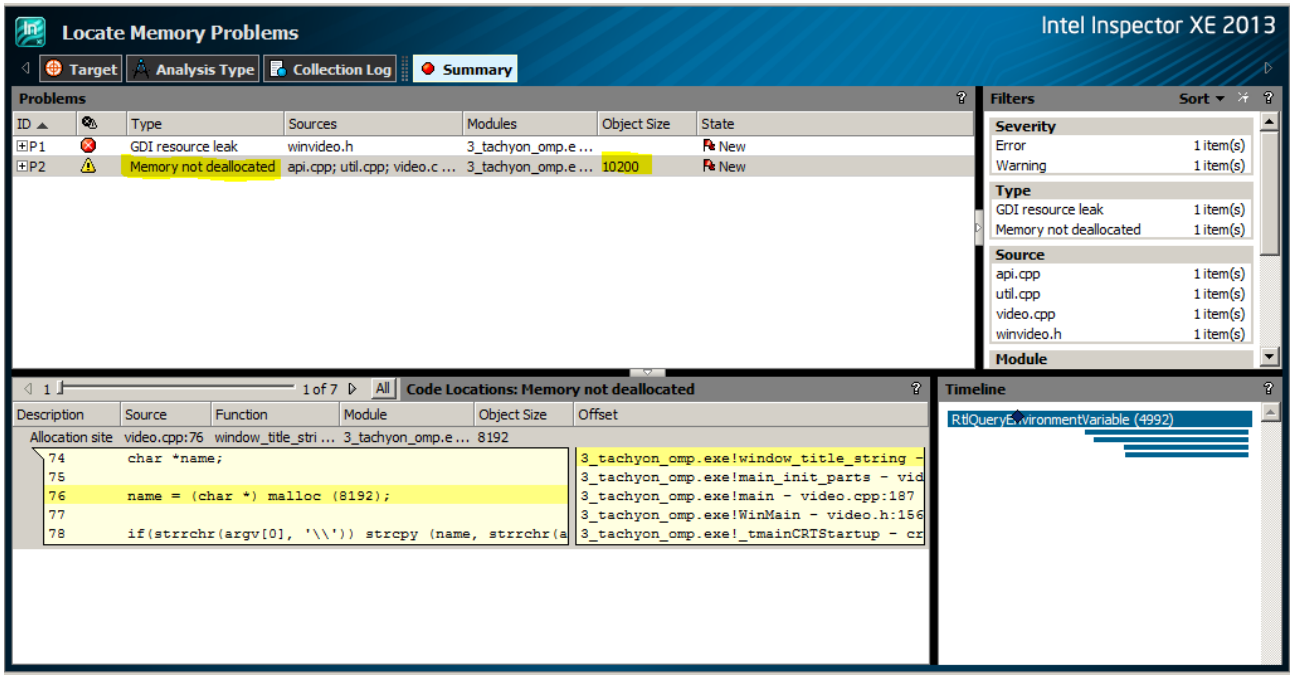
Por otro lado observamos que la función *next_frame()* tiene un tiempo de sobrecarga casi igual al total del tiempo que tarda en realizarse, esto es debido a que se accede desde todos los hilos y no hemos protegido la variable que hay en la función (*g_updates*) con un mutex, por lo que aparecen problemas de sincronización con esta variable llevándonos a una sobrecarga de tiempo grande.

El resto de tipos de análisis diferentes que ofrece quizás no nos interesen tanto a nivel general, ya que dan información muy detallada de por ejemplo: accesos a memoria, tiempo de accesos a memoria, ciclos de CPU, etc.

## Inspector XE

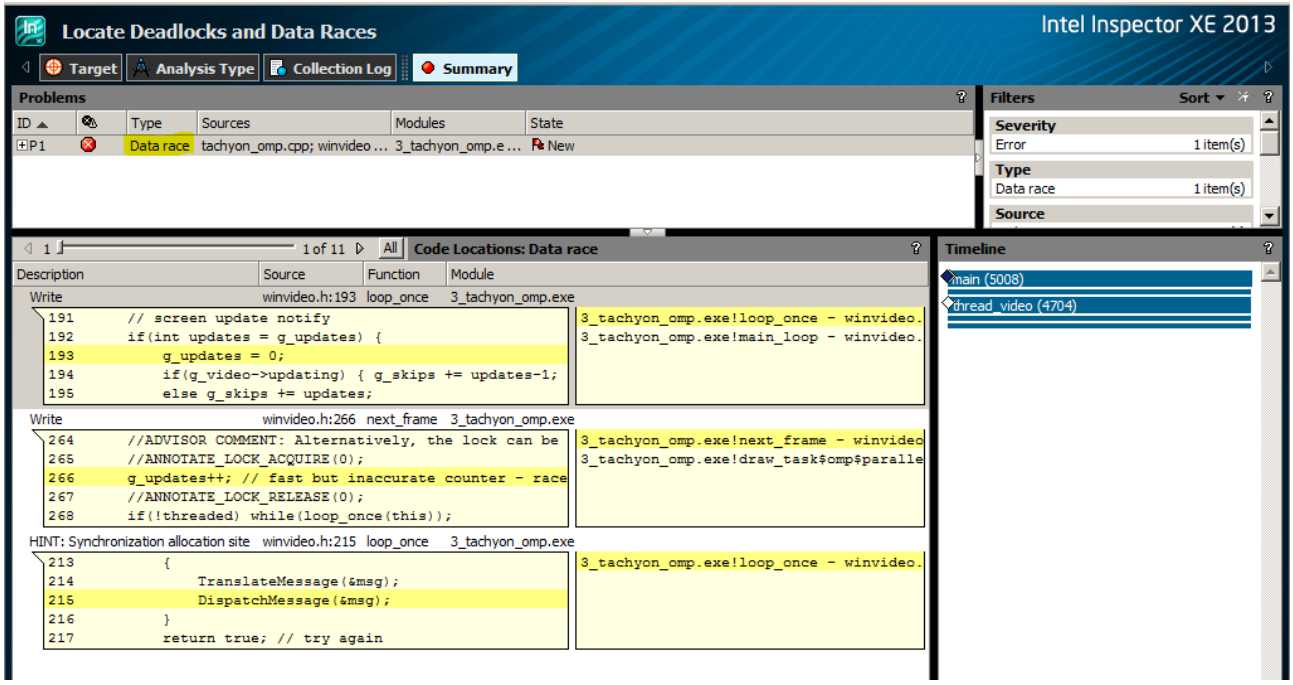
Esta es otra de las herramientas que tenemos a nuestra disposición. En esta herramienta básicamente se pueden hacer dos tipos de análisis, de fugas y otros problemas de memoria y problemas con los hilos como pueden ser condiciones de carrera y *deadlocks*.

Hemos realizado el análisis de problemas de memoria con el ejemplo anterior del programa que renderiza y muestra una imagen, estos son los resultados que nos ha dado:




Vemos que el problema aquí es que reservamos espacio para la variable *name*, pero nunca lo liberamos, la solución sería hacer un *dealloc()* de la memoria asignada a *name* justo antes de salir del programa para liberar la memoria.

El otro tipo de análisis que podemos hacer con esta herramienta es el relacionado con los hilos y sus posibles problemas como pueden ser las condiciones de carrera por algún recurso o los *deadlocks*. Estos son los resultados del análisis para el mismo ejemplo:




El resultado es que tenemos un problema con el recurso *g_updates*. Este error es el que hemos visto anteriormente con la sobrecarga de la función *next_frame()*, que por dentro usa este recurso. La solución es usar un mutex para acceder a la variable, de esta manera los procesos se organizan para acceder cada uno a su tiempo y no se produce tanta sobrecarga en esperar.

## Advisor XE

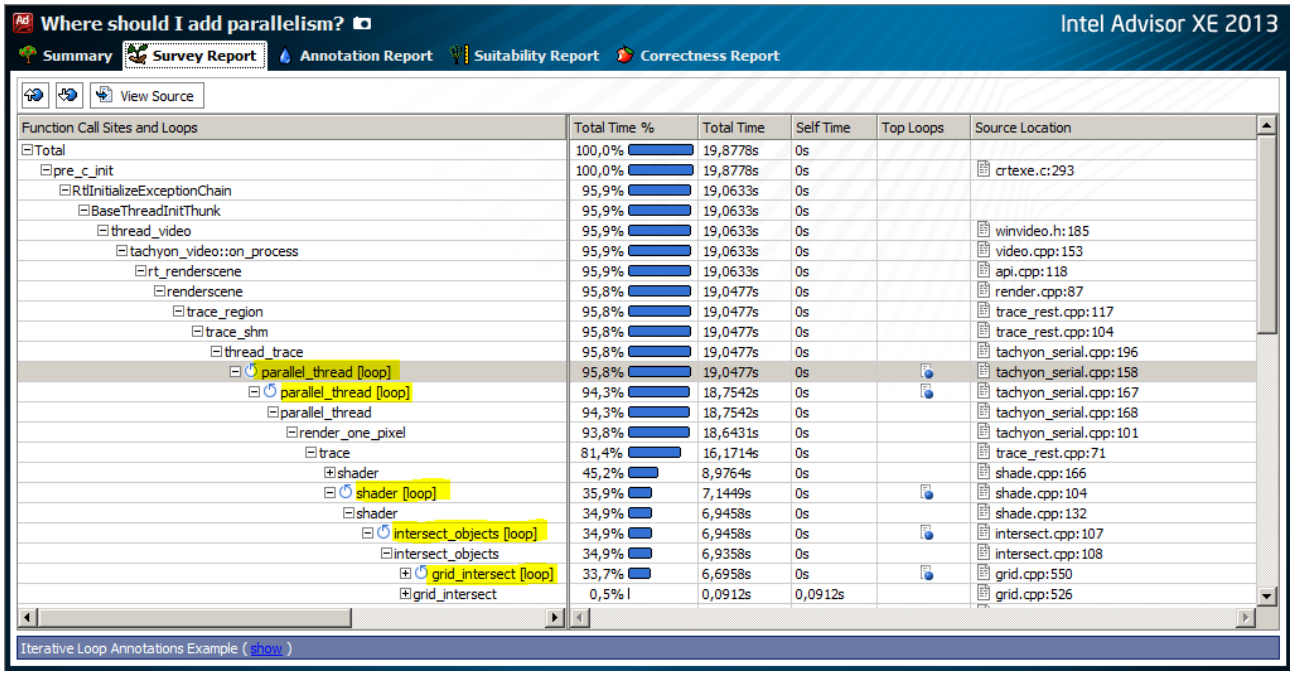
Esta es la tercera y última herramienta. Esta herramienta se usa básicamente para detectar partes del código que se pueden paralelizar aprovechando los recursos que nos ofrecen los procesadores modernos con cada vez más capacidad de procesamiento en paralelo.

El proceso a seguir es compilar al aplicación que queremos analizar en *Release*, y con las siguientes opciones en el proyecto activadas:

- C++ → General → Debug Information Format: **Program Database (/Zi)**
- C++ → Optimization → Optimization: **Maximize speed (/O2)** | Inline Function Expansion: **Only __inline (/Ob1)** | Enable Intrinsic Functions: **No**
- C++ → Code Generation → Runtime Library: **Multi-threaded DLL (/MD)**
- Linker → Debugging → Generate Debug Info: **Yes (/DEBUG)**

El proceso de análisis tiene cinco fases bien diferenciadas.

La primera consiste en recoger información sobre el programa para poder ver que partes del código son buenas candidatas para paralelizar. El resultado de este análisis para el ejemplo de la imagen es:




En la pestaña del análisis se nos muestra los bucles más importantes y el tiempo de cada uno, así como la línea de código donde se encuentra cada uno. Con esto podemos ir a cada bucle y poner unas instrucciones especiales que requieren incluir*: #include <advisor-annotate.h>* y que se encuentra en la carpeta donde se instala el programa en el directorio *include*.

Las anotaciones que podemos hacer para el *parallel_thread()* son:

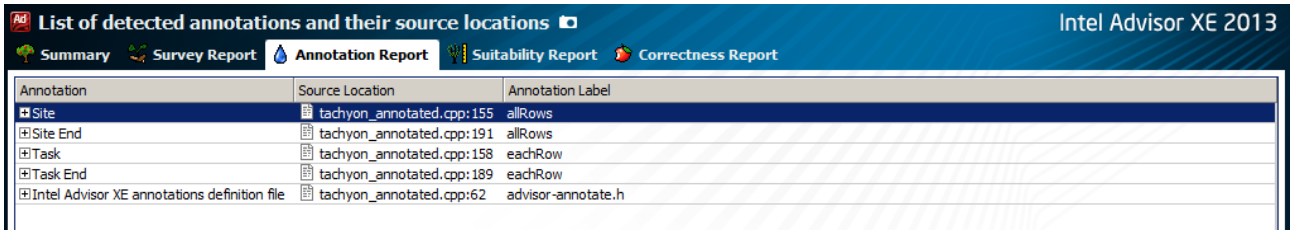
```
static void parallel_thread (void)
{
	ANNOTATE_SITE_BEGIN(allRows);
	for (int y = starty; y < stopy; y++)
	{
		ANNOTATE_TASK_BEGIN(eachRow);
		// Instrucciones
		ANNOTATE_TASK_END(eachRow);
	}
	ANNOTATE_SITE_END(allRows);

}
```

Para hacer estas anotaciones, seleccionamos el bucle y hacemos click derecho → Intel Advisor XE 2013 → Annotate Site, y para las instrucciones Annotate Task y les damos un nombre a cada uno. Haríamos lo mismo para los demás bucles.

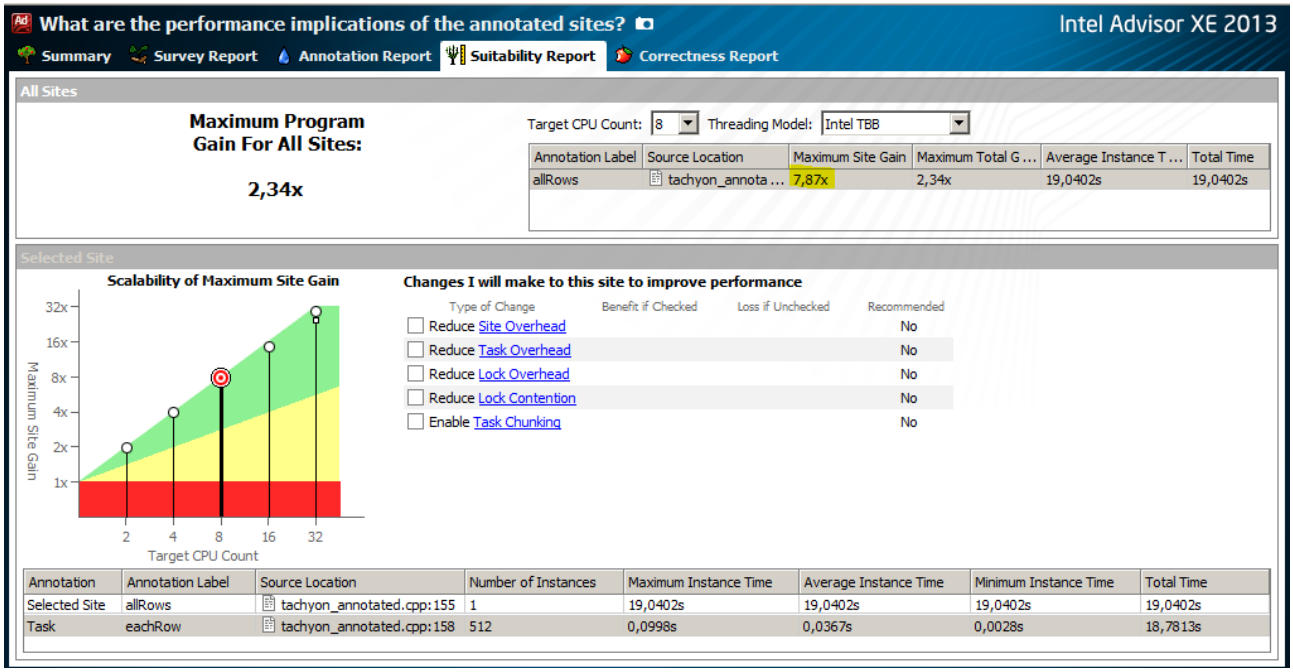
La segunda fase es ver las anotaciones que hemos hecho en la fase 1 y

asegurarnos que están todas correctas y que hemos incluido la cabecera necesaria para que el programa las detecte y podamos así avanzar a la siguiente fase.




En nuestro caso solo hemos hecho anotaciones en el bucle *`parallel_thread()`*, los demás los hemos dejado pero podríamos haberlos incluido también.

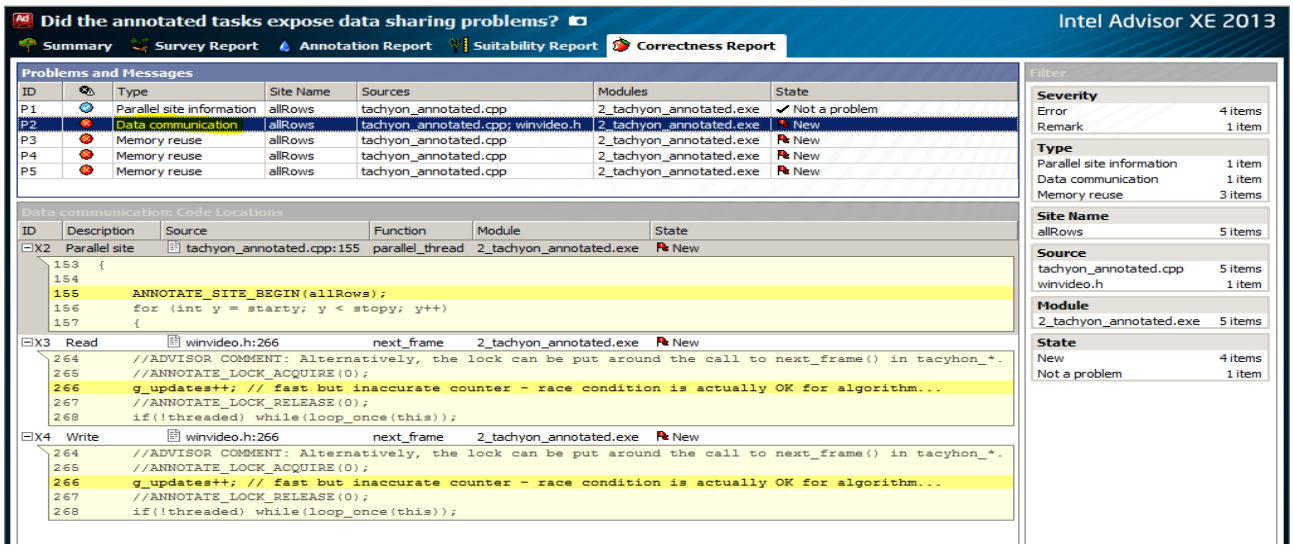
La tercera fase nos dice, ayudándose en las anotaciones que hemos hecho, que podríamos interpretar como intenciones de paralelización que tenemos, la ganancia que obtendríamos por hacer lo que pone en las anotaciones para diversos números de núcleos, métodos de sincronización, etc.

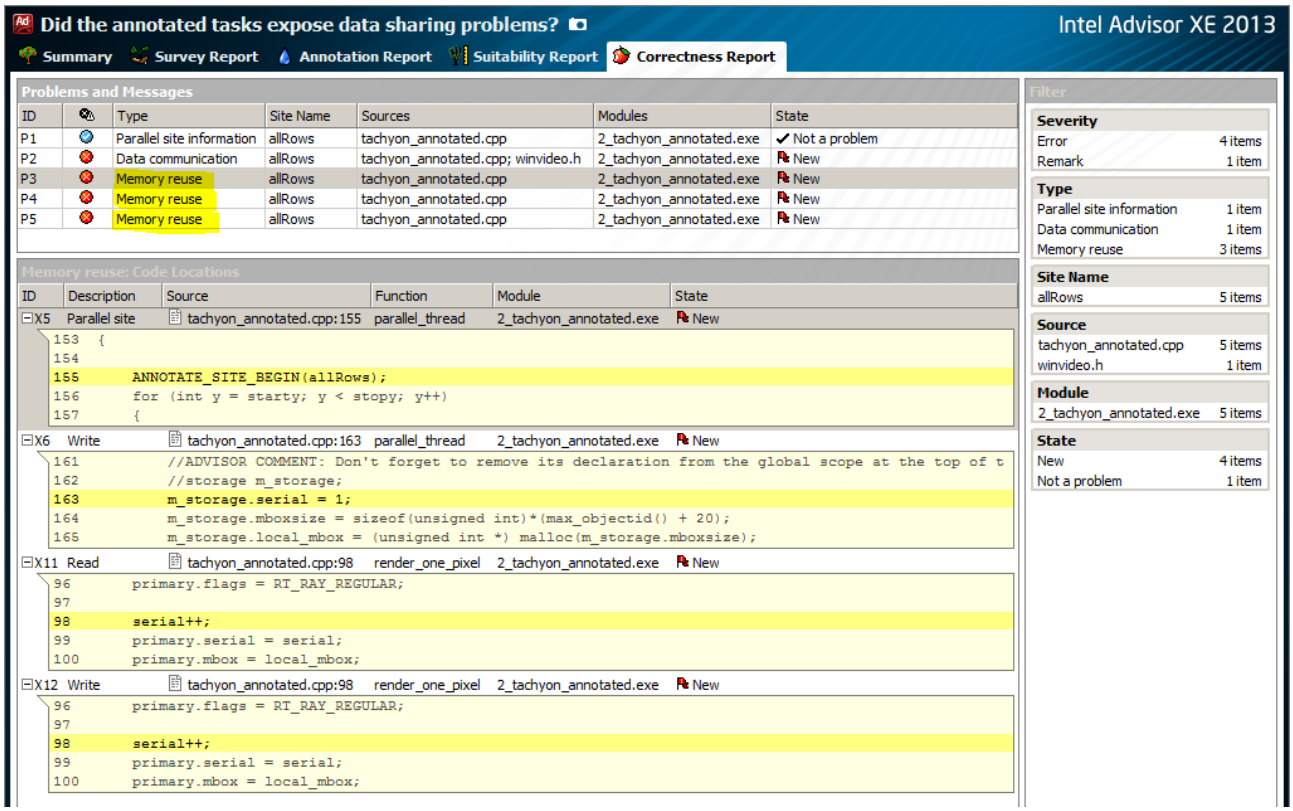



Este es un ejemplo muy paralelizable, y podemos observar que para 8 núcleos paralelizando el bucle que hemos anotado en la fase anterior, podríamos llegar a obtener una ganancia de 7,87 o sea de casi 8 veces más rápido. Llegando a ser de 29,22 para 32 núcleos, lo que nos indica que en este caso aprovecharíamos todos los núcleos que pudiésemos tener disponibles para ejecutar nuestra aplicación.

La cuarta fase también hay que hacerla en *Debug*, y nos informa si los cambios

introducidos por la anotaciones que hemos hecho nos llevarían a problemas relacionados con compartir datos entre procesos.




Vuelve a indicarnos el problema con la variable *g_updates* que ya hemos visto antes y luego nos dice también que hay problemas con la variable global *m_storage*, que es usada en el bucle `parallel_thread()` y que cuando se paraleliza cada ejecución accede a la misma variable sin restricciones, la solución es declarar esa variable local al bucle ya que no la usamos fuera y de esta manera ya no ocurre este problema.

La quinta y última fase consiste en sustituir las anotaciones por código que de verdad se ejecute en paralelo, *mutex* que realmente hagan el acceso exclusivo a las variables, etc. Esto es un proceso que debemos hacer por nuestra cuenta y en el que el programa no puede ayudarnos. Lo que sí que tenemos son unos enlaces a distintos métodos para hacer este cambio y enlaces a cada uno de los tipos para aprender a usarlos si no sabemos.

La documentación se encuentra en: *Composer_XE_2013/documentation/en/tutorials/C++/index.htm*

# ANEXO 8:

# TABLAS PRUEBAS RENDIMIENTO SCATIRTPVIDEOCLIENT

## H264

Sin activar ninguna opción del cliente de vídeo.

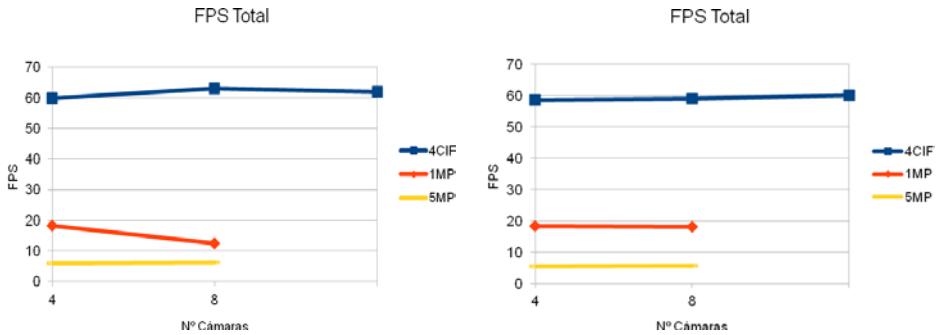
| 32/64 bits | PC | Códec | Resolución | Cámaras | FPS totales | FPS cámara | Bitrate total | Bitrate cámara | CPU total | CPU cámara | Memoria total | Memoria cámara |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 64 | Celeron E3400 2.60GHz | H.264 | 4CIF | 4 | 59,86 | 14,965 | 434,9 | 108,725 | 90 | 22,5 | 78 | 19,5 |
| 64 | Celeron E3400 2.60GHz | H.264 | 4CIF | 8 | 63,05 | 7,88125 | 458,2 | 57,275 | 93 | 11,625 | 146 | 18,25 |
| 64 | Celeron E3400 2.60GHz | H.264 | 4CIF | 16 | 61,94 | 3,87125 | 456,7 | 28,54375 | 95 | 5,9375 | 281 | 17,5625 |
| 64 | Celeron E3400 2.60GHz | H.264 | 1MP | 4 | 18,19 | 4,5475 | 895,1 | 223,775 | 92 | 23 | 204 | 51 |
| 64 | Celeron E3400 2.60GHz | H.264 | 1MP | 8 | 12,53 | 1,56625 | 656,6 | 82,075 | 94 | 11,75 | 387 | 48,375 |
| 64 | Celeron E3400 2.60GHz | H.264 | 5MP | 4 | 6,06 | 1,515 | 748 | 187 | 86 | 21,5 | 708 | 177 |
| 64 | Celeron E3400 2.60GHz | H.264 | 5MP | 8 | 6,2 | 0,775 | 904,1 | 113,0125 | 90 | 11,25 | 1340 | 167,5 |
| 32 | Celeron E3400 2.60GHz | H.264 | 4CIF | 4 | 58,55 | 14,6375 | 427,7 | 106,925 | 93 | 23,25 | 76 | 19 |
| 32 | Celeron E3400 2.60GHz | H.264 | 4CIF | 8 | 59,01 | 7,37625 | 431,4 | 53,925 | 93 | 11,625 | 144 | 18 |
| 32 | Celeron E3400 2.60GHz | H.264 | 4CIF | 16 | 60,12 | 3,7575 | 431,7 | 26,98125 | 96 | 6 | 278 | 17,375 |
| 32 | Celeron E3400 2.60GHz | H.264 | 1MP | 4 | 18,39 | 4,5975 | 918,6 | 229,65 | 95 | 23,75 | 198 | 49,5 |
| 32 | Celeron E3400 2.60GHz | H.264 | 1MP | 8 | 18,26 | 2,2825 | 898,9 | 112,3625 | 95 | 11,875 | 384 | 48 |
| 32 | Celeron E3400 2.60GHz | H.264 | 5MP | 4 | 5,57 | 1,3925 | 741,8 | 185,45 | 94 | 23,5 | 708 | 177 |
| 32 | Celeron E3400 2.60GHz | H.264 | 5MP | 8 | 5,76 | 0,72 | 749,9 | 93,7375 | 98 | 12,25 | 1405 | 175,625 |

Descomprimiendo sólo los *keyframes*.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 64 | Celeron E3400 2.60GHz | H.264 | 4CIF | 4 | 99,42 | 24,855 | 719 | 179,75 | 8 | 2 | 75 | 18,75 |
| 64 | Celeron E3400 2.60GHz | H.264 | 4CIF | 8 | 199,83 | 24,97875 | 1455 | 181,875 | 16 | 2 | 141 | 17,625 |
| 64 | Celeron E3400 2.60GHz | H.264 | 4CIF | 16 | 400,89 | 25,055625 | 2923 | 182,6875 | 35 | 2,1875 | 275 | 17,1875 |
| 64 | Celeron E3400 2.60GHz | H.264 | 4CIF | 32 | 800,2 | 25,00625 | 5839,7 | 182,490625 | 70 | 2,1875 | 541 | 16,90625 |
| 64 | Celeron E3400 2.60GHz | H.264 | 4CIF | 64 | 932,63 | 14,5723438 | 6622,6 | 103,478125 | 74 | 1,15625 | 1070 | 16,71875 |
| 64 | Celeron E3400 2.60GHz | H.264 | 1MP | 4 | 99,69 | 24,9225 | 4906,5 | 1226,625 | 53 | 13,25 | 196 | 49 |
| 64 | Celeron E3400 2.60GHz | H.264 | 1MP | 8 | 161 | 20,125 | 7885,9 | 985,7375 | 82 | 10,25 | 382 | 47,75 |
| 64 | Celeron E3400 2.60GHz | H.264 | 1MP | 16 | 153 | 9,5625 | 7522,1 | 470,13125 | 82 | 5,125 | 751 | 46,9375 |
| 64 | Celeron E3400 2.60GHz | H.264 | 1MP | 32 | 153,4 | 4,79375 | 7538,8 | 235,5875 | 88 | 2,75 | 1478 | 46,1875 |
| 64 | Celeron E3400 2.60GHz | H.264 | 5MP | 4 | 50,08 | 12,52 | 6442,9 | 1610,725 | 86 | 21,5 | 710 | 177,5 |
| 64 | Celeron E3400 2.60GHz | H.264 | 5MP | 8 | 50,71 | 6,33875 | 6537,5 | 817,1875 | 90 | 11,25 | 1400 | 175 |
| 32 | Celeron E3400 2.60GHz | H.264 | 4CIF | 4 | 99,79 | 24,9475 | 726,3 | 181,575 | 8 | 2 | 74 | 18,5 |
| 32 | Celeron E3400 2.60GHz | H.264 | 4CIF | 8 | 198,99 | 24,87375 | 1448,8 | 181,1 | 19 | 2,375 | 142 | 17,75 |
| 32 | Celeron E3400 2.60GHz | H.264 | 4CIF | 16 | 398,72 | 24,92 | 2885,5 | 180,34375 | 36 | 2,25 | 275 | 17,1875 |
| 32 | Celeron E3400 2.60GHz | H.264 | 4CIF | 32 | 799,35 | 24,9796875 | 5833,1 | 182,284375 | 75 | 2,34375 | 542 | 16,9375 |
| 32 | Celeron E3400 2.60GHz | H.264 | 4CIF | 64 | 883,33 | 13,8020313 | 6414,9 | 100,2328125 | 86 | 1,34375 | 1071 | 16,734375 |
| 32 | Celeron E3400 2.60GHz | H.264 | 1MP | 4 | 99,57 | 24,8925 | 4900,6 | 1225,15 | 50 | 12,5 | 192 | 48 |
| 32 | Celeron E3400 2.60GHz | H.264 | 1MP | 8 | 108,29 | 13,53625 | 5311,7 | 663,9625 | 60 | 7,5 | 378 | 47,25 |
| 32 | Celeron E3400 2.60GHz | H.264 | 1MP | 16 | 140,76 | 8,7975 | 6931,2 | 433,2 | 76 | 4,75 | 747 | 46,6875 |
| 32 | Celeron E3400 2.60GHz | H.264 | 5MP | 4 | 99,96 | 24,99 | 4922,9 | 1230,725 | 50 | 12,5 | 795 | 198,75 |
| 32 | Celeron E3400 2.60GHz | H.264 | 5MP | 8 | 65,23 | 8,15375 | 8527,4 | 1065,925 | 80 | 10 | 1410 | 176,25 |

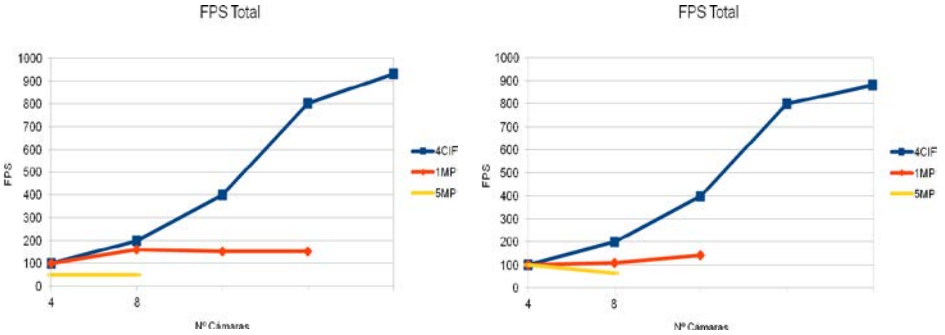
Sin opciones seleccionada en el cliente

Comparación FPS versión de 64-bits (arriba.) y 32-bits (abajo) son prácticamente iguales.




Sólo teniendo en cuenta los *keyframes*.

Resultan casi idénticos los resultados entre las versiones de 32-bits (abajo) y 64-bits (arriba) pero son valores muy superiores a los anteriores como cabría esperar.




## MPEG4

Sin activar ninguna opción del cliente de vídeo.

| 32/64 bits | PC | Códec | Resolución | Cámaras | FPS totales | FPS cámara | Bitrate total | Bitrate cámara | CPU total | CPU cámara | Memoria total | Memoria cámara |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 64 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 4 | 98,62 | 24,655 | 628,5 | 157,125 | 53 | 13,25 | 33 | 8,25 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 8 | 154,52 | 19,315 | 996,8 | 124,6 | 88 | 11 | 61 | 7,625 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 16 | 155,49 | 9,718125 | 935,3 | 58,45625 | 90 | 5,625 | 111 | 6,9375 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 32 | 159,02 | 4,969375 | 982,6 | 30,70625 | 93 | 2,90625 | 208 | 6,5 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 64 | 147,72 | 2,308125 | 1201,4 | 18,771875 | 90 | 1,40625 | 406 | 6,34375 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 1MP | 4 | 48,43 | 12,1075 | 778,3 | 194,575 | 87 | 21,75 | 82 | 20,5 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 1MP | 8 | 54,48 | 6,81 | 860 | 107,5 | 88 | 11 | 152 | 19 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 5MP | 4 | 12,3 | 3,075 | 1321,7 | 330,425 | 94 | 23,5 | 290 | 72,5 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 5MP | 8 | 13,06 | 1,6325 | 1438,2 | 179,775 | 96 | 12 | 575 | 71,875 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 4 | 98,93 | 24,7325 | 635,5 | 158,875 | 60 | 15 | 51 | 12,75 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 8 | 155,13 | 19,39125 | 1012,2 | 126,525 | 90 | 11,25 | 77 | 9,625 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 16 | 149,62 | 9,35125 | 1010,9 | 63,18125 | 90 | 5,625 | 130 | 8,125 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 32 | 156,84 | 4,90125 | 952,5 | 29,765625 | 94 | 2,9375 | 235 | 7,34375 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 1MP | 4 | 49,28 | 12,32 | 791,5 | 197,875 | 89 | 22,25 | 93 | 23,25 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 1MP | 8 | 51,61 | 6,45125 | 814,8 | 101,85 | 91 | 11,375 | 167 | 20,875 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 5MP | 4 | 12,53 | 3,1325 | 1379,9 | 344,975 | 95 | 23,75 | 307 | 76,75 |

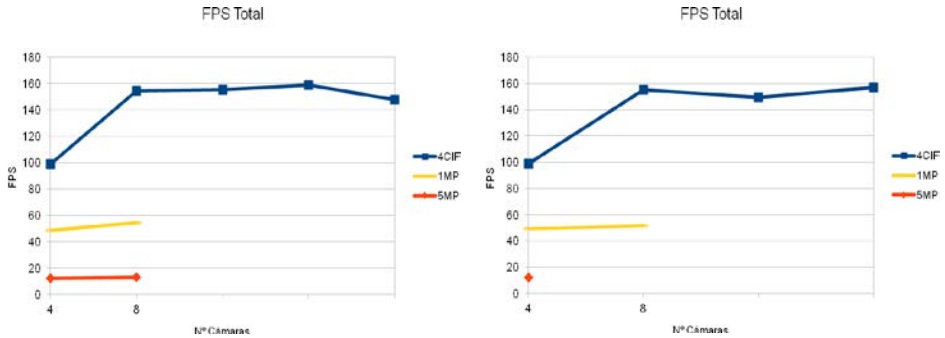
Descomprimiendo sólo los *keyframes*.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 64 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 4 | 99,69 | 24,9225 | 649,6 | 162,4 | 4 | 1 | 32 | 8 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 8 | 197,98 | 24,7475 | 1247,2 | 155,9 | 7 | 0,875 | 57 | 7,125 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 16 | 399,83 | 24,989375 | 2562,6 | 160,1625 | 15 | 0,9375 | 108 | 6,75 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 32 | 806,81 | 25,2128125 | 5185,3 | 162,040625 | 33 | 1,03125 | 207 | 6,46875 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 64 | 1617,11 | 25,2673438 | 10504,4 | 164,13125 | 63 | 0,984375 | 401 | 6,265625 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 100 | 1600,54 | 16,0054 | 10347,4 | 103,474 | 70 | 0,7 | 622 | 6,22 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 1MP | 4 | 99,25 | 24,8125 | 1592,9 | 398,225 | 15 | 3,75 | 83 | 20,75 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 1MP | 8 | 199,57 | 24,94625 | 3195,9 | 399,4875 | 32 | 4 | 151 | 18,875 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 1MP | 16 | 398,21 | 24,888125 | 6381,7 | 398,85625 | 62 | 3,875 | 292 | 18,25 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 1MP | 32 | 516,12 | 16,12875 | 8285,9 | 258,934375 | 75 | 2,34375 | 573 | 17,90625 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 1MP | 64 | 455,28 | 7,11375 | 7327,4 | 114,490625 | 88 | 1,375 | 1019 | 15,921875 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 5MP | 4 | 24,22 | 6,055 | 3435,5 | 858,875 | 83 | 20,75 | 294 | 73,5 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 5MP | 8 | 24,26 | 3,0325 | 3436,5 | 429,5625 | 89 | 11,125 | 575 | 71,875 |
| 64 | Celeron E3400 2.60GHz | MPEG4 | 5MP | 16 | 80,9 | 5,05625 | 8775,4 | 548,4625 | 91 | 5,6875 | 1135 | 70,9375 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 4 | 99,81 | 24,9525 | 629,5 | 157,375 | 3 | 0,75 | 48 | 12 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 8 | 199,63 | 24,95375 | 1292,8 | 161,6 | 9 | 1,125 | 75 | 9,375 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 16 | 401,39 | 25,086875 | 2595,5 | 162,21875 | 19 | 1,1875 | 128 | 8 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 32 | 796,4 | 24,8875 | 5189,5 | 162,171875 | 40 | 1,25 | 232 | 7,25 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 64 | 1612,84 | 25,200625 | 10501,2 | 164,08125 | 68 | 1,0625 | 437 | 6,828125 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 4CIF | 100 | 1537,31 | 15,3731 | 9873,5 | 98,735 | 79 | 0,79 | 665 | 6,65 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 1MP | 4 | 99,68 | 24,92 | 1597,5 | 399,375 | 17 | 4,25 | 92 | 23 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 1MP | 8 | 202,07 | 25,25875 | 3247,1 | 405,8875 | 31 | 3,875 | 165 | 20,625 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 1MP | 16 | 398,58 | 24,91125 | 6384,7 | 399,04375 | 64 | 4 | 307 | 19,1875 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 1MP | 32 | 487,61 | 15,2378125 | 7882 | 246,3125 | 80 | 2,5 | 590 | 18,4375 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 1MP | 64 | 510,43 | 7,97546875 | 8117,3 | 126,8328125 | 85 | 1,328125 | 1149 | 17,953125 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 5MP | 4 | 77,29 | 19,3225 | 8339,4 | 2084,85 | 82 | 20,5 | 306 | 76,5 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 5MP | 8 | 76 | 9,5 | 8168,1 | 1021,0125 | 81 | 10,125 | 574 | 71,75 |
| 32 | Celeron E3400 2.60GHz | MPEG4 | 5MP | 16 | 79,67 | 4,979375 | 8621,5 | 538,84375 | 91 | 5,6875 | 1122 | 70,125 |

Los resultados son mejores que para H264 como era de esperar, puesto que el códec h264 ofrece un tamaño menor al comprimir, lo que hace que se necesiten más recursos a la hora de realizar la compresión.

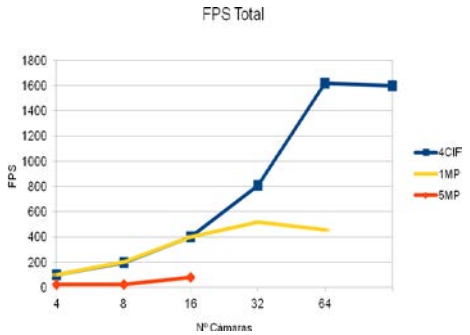
Sin opciones seleccionada en el cliente

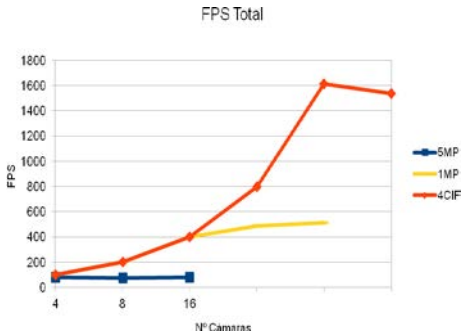
Comparación FPS versión de 64-bits (arriba.) y 32-bits (abajo) son prácticamente iguales.




Sólo teniendo en cuenta los *keyframes*.

Resultan casi idénticos los resultados entre las versiones de 32-bits (abajo) y 64-bits (arriba) pero son valores muy superiores a los anteriores como cabría esperar.


FPS Total
1800
1600
1400
1200
1000
800
600
400
200
0
FPS
4
8
16
32
64
Nº Cámaras
4CIF
1MP
5MP


FPS Total
1800
1600
1400
1200
1000
800
600
400
200
0
FPS
4
8
16
Nº Cámaras
5MP
1MP
4CIF

# BIBLIOGRAFIA

[1] Microsoft, *Best Practices for WOW64.*
[2] Bill Graham y Edwin Verplanke, *Optimizing Intel Multi-core Embedded Platforms*, Intel Corporation.
[3] Agner Fog, *Optimizing software in C++. An optimization guide for Windows, Linux and Mac platforms,* Copenhagen University College of Engineering.
[4] *Intel Integrated Performance Primitives for Windows OS,* Intel Software Development Products.
[5] *Intel Integrated Performance Primitivesreference manual, Volume 2: Image and Video Processing,* Intel Software Development Products.
[6] *Quick-Reference Guide to Optimization with Intel Compilers version 10.x,* Intel Software Development Products.
[7] " http://software.intel.com/en-us/articles. Web site que contiene artículos relacionados con las arquitecturas de Intel".
[8] " http://blogs.msdn.com. Blog de Microsoft que contiene explicaciones sobre las distintas opciones disponibles en Visual Studio 2008 C++".
[9] " http://msdn.microsoft.com/en-us/library. Web site con artículos relacionados con la compilación y opciones de las herramientas de desarrollo de Microsoft".
[10] " http://www.codemachine.com/article_x64deepdive.html. Artículo sobre los problemas más comunes a la hora de realizar una migración a 64 bits".
[11] " http://eli.thegreenplace.net/2011/09/06/stack-frame-layout-on-x86-64/. Artículo que describe la estructura y uso de la pila en arquitecturas de 64 bits".
[12] " http://mark.masmcode.com/. Artículo con consejos a la hora de optimizar código ensamblador, Mark Larson".
[13] " http://graphics.stanford.edu/~seander/bithacks.html. Trucos para mejorar el rendimiento en operaciones con bits, Sean Eron Anderson, Stanford University".
[14] " http://www.gamedev.net/page/resources/_/technical/general-programming/100-bugs-in-open-source-cc-projects-r2886. 100 ejemplos de problemas a la hora de migrar un aplicación de 32 a 64 bits en código abierto, Andrey Karpov".
[15] " http://www.tantalon.com/pete/cppopt/final.htm. Web site que describe posibles optimizaciones, Pete Isensee".
[16] " http://www.viva64.com/. Web site donde se puede obtener el analizador estático utilizado para el análisis de código, además contiene muchos artículos relacionados con el tema, Evgeniy Ryzhkov y Andrey Karpov"