Optimización de una implementación JPEG teniendo en cuenta la arquitectura actual de los procesadores

J. A. Padilla
Logic Factory
http://www.logic-factory.com

M. Anguita, F. J. Fernández, A. F. Díaz, A. Cañas, A. Prieto
Dpto. de Arquitectura y Tecnología de Computadores
Universidad de Granada
18071 Granada
e-mail: manguita@atc.ugr.es

Resumen

Creemos conveniente proponer trabajos fin de carrera con el objetivo de que los estudiantes comprueben que pueden *mejorar* ostensiblemente las prestaciones de sus programas en *poco tiempo* aplicando los conocimientos sobre *arquitectura* de computadores que han ido adquiriendo a lo largo de la titulación. Mostrando los resultados obtenidos en estos trabajos a estudiantes de diferentes cursos de la titulación, pretendemos incrementar su *motivación* en las materias de arquitectura. Aquí se muestra el incremento en prestaciones obtenido, aprovechando la arquitectura actual de los computadores, en una implementación de descodificador *JPEG*. Para la mejora de prestaciones, se propone seguir un *proceso iterativo* de varios pasos.

1. Introducción

La *motivación* es uno de los aspectos que más influye en el proceso de aprendizaje, de hecho un estudiante motivado se aproxima al máximo de sus posibilidades. En [12] se define la motivación como "aquellas condiciones o estados del individuo que le activan o dan energía para llevar a cabo una conducta dirigida hacia determinados objetivos, impulsándole a superar los obstáculos que se le presentan". Para favorecer la motivación es esencial proporcionar un *sentido de utilidad* a los contenidos que se imparten. Los estudiantes que prevén dedicar su vida laboral a la programación de aplicaciones, a menudo, no se enfrentan al estudio de la arquitectura y estructura de computadores suficientemente motivados, lo que les puede llevar a fracasar en estas materias. Para incrementar la motivación de los estudiantes, se les podría demostrar, que pueden mejorar las prestaciones de sus programas aplicando los conocimientos de arquitectura de computadores que se adquieren en estas materias. Con el fin de ser más convincentes, deberían obtener las mejoras ellos mismos, o se les debería mostrar resultados que estimen que pueden obtener ellos mismos, como por ejemplo resultados logrados por otros estudiantes de la propia titulación.

Por ello creemos conveniente proponer trabajos fin de carrera, en los que los estudiantes comprueben que pueden *mejorar* en gran medida las prestaciones de sus programas en *poco tiempo* aplicando los conocimientos sobre arquitectura y estructura de computadores que han adquirido. Así, los estudiantes afianzan estos conocimientos, se favorece su utilización posterior, y además obtenemos resultados que se pueden mostrar a otros estudiantes, como se comentó más arriba.

Por otra parte, incrementar las prestaciones de un programa atendiendo a la arquitectura, no sólo resulta actualmente de interés para aplicaciones embebidas, también para aplicaciones ejecutadas en el entorno multitarea de un sistema de propósito general. Las *aplicaciones embebidas* (en impresoras, fotocopiadoras, DVD, cámaras de fotografía o vídeo digital, asistentes personales, teléfonos móviles, etc) requieren altas prestaciones, en muchos casos tiempo real, y están sometidas a otras restricciones como: consumo de potencia, tamaño o precio. Estas restricciones hacen que se opte para estas aplicaciones por procesadores de bajo precio, baja frecuencia de reloj, y también por ejemplo por ajustar el tamaño

y precio de la memoria. Por tanto, para conseguir las prestaciones necesarias para las aplicaciones embebidas, dadas estas restricciones, el programador tiene que *extraer el máximo provecho de la arquitectura*. Los fabricantes de procesadores de propósito general, suelen ofrecer versiones de sus procesadores con consumos de potencia reducidos, para su utilización en aplicaciones incrustadas (Ultrasparc de Sun, PowerPC de IBM-Motorola-Apple, Pentium de Intel, etc).

Resulta también de interés mejorar prestaciones para aplicaciones ejecutadas en el entorno multitarea de un PC u otro *sistema de propósito general*, donde el usuario desea ejecutar varios programas simultáneamente y con buenas prestaciones. Además, para la ejecución de aplicaciones multimedia en estos entornos, evitando en lo posible el uso de hardware específico, se ha de explotar al máximo la arquitectura ([3], [6], [13], [14]).

Adicionalmente, hay que tener en cuenta, que las *prestaciones* y el *tiempo de desarrollo del software* son esenciales, entre otras cosas, debido a la competencia entre empresas, y que aprovechando los conocimientos de arquitectura que ya se tienen, se puede conseguir una buena relación prestaciones-tiempo de desarrollo. Para mejorar adicionalmente esta relación, el programador puede utilizar el software de ayuda a la programación y optimización que proporcionan los fabricantes para sus arquitecturas. Software tradicionalmente ofrecido por los fabricantes de DSP (*Digital Signal Processor*), y que ahora se ha extendido a procesadores de propósito general, especialmente a raíz de la incorporación de unidades SIMD (MMX, SSE y SSE2 de Intel, VIS de Sun, 3Dnow! de AMD o Altivec para PowerPC), por ejemplo el paquete Vtune y los compiladores de Intel [17], CodeAnalyst de AMD [1], o VSDK de Sun [16].

Para mejorar prestaciones en una aplicación, se pueden utilizar los conocimientos adquiridos sobre la *arquitectura abstracta* del procesador o ISA (*Instruction Set Architecture*): repertorio de instrucciones, conjunto de registros y su gestión (pila o acceso directo), tipos de datos y modos de direccionamiento; y sobre la *arquitectura concreta*: arquitectura segmentada, superescalar, VLIW, SIMD, jerarquía de memoria. También se puede aplicar los conocimientos sobre las optimizaciones que actualmente pueden realizar (y las que no son capaces de realizar) los compiladores ([9], [18], [2], [10]).

En estos trabajos fin de carrera, se propone al estudiante la implementación completa de alguna aplicación, se eligen aplicaciones que puedan atraer al estudiante por su amplia utilización (se están desarrollando trabajos para jpeg, mpeg-2 y mp3), o por su interés en investigación (actualmente se está desarrollado un trabajo sobre procesamiento de flujo óptico para evaluación de movimiento).

En la Figura 1 se muestra el proceso que se puede seguir para obtener la versión optimizada de la implementación, objetivo del trabajo:

1. *Programar aplicación.* Partiendo de una descripción detallada de la aplicación, se pretende que el estudiante desarrolle un programa tal y como lo haría habitualmente. Simultáneamente buscaría entradas al programa representativas que le permitan comprobar que éste funciona correctamente, y que serán utilizadas posteriormente para analizar las prestaciones del código.
2. Incluir en puntos del código *medidas de prestaciones,* con el fin de analizar el tiempo de ejecución que suponen los diferentes bloques funcionales de la aplicación.
3. *Ejecutar el programa* con el conjunto de entradas seleccionado y *recolectar*, el tiempo de ejecución total y los tiempos obtenidos para cada bloque funcional.
4. Si se ha llegado a las *prestaciones* que se tenían como *objetivo* o se ha agotado el *tiempo de desarrollo propuesto* termina el proceso.
5. *Localizar el bloque o bloques funcionales* que destaquen por su *mayor tiempo* de ejecución.
6. *Analizar* la implementación de los bloques seleccionados en el paso 5, y *modificar* esta implementación, teniendo en cuanta la arquitectura, con el objetivo de *mejorar* prestaciones. La posibilidad que ofrecen los compiladores de generar el código ensamblador de un programa, se utiliza en esta etapa para ver aquellos aspectos de la arquitectura concreta o aquellas instrucciones del repertorio máquina que el compilador no ha utilizado en estos bloques, con el fin de que sean aprovechados explícitamente por el programador. A continuación, se procedería nuevamente a recolectar tiempos, es decir se vuelve al punto 3.

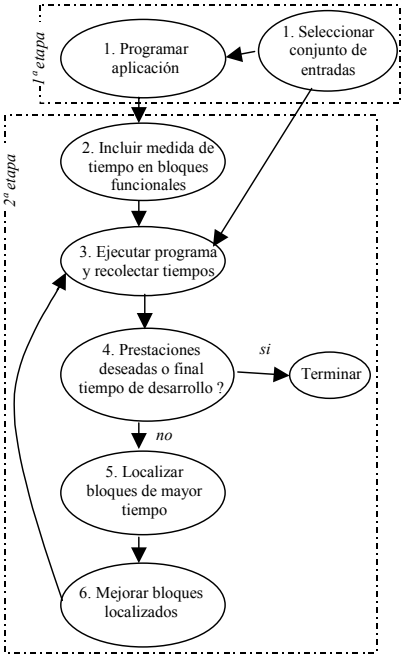

Figura 1. Proceso que se puede seguir para optimizar la implementación.

En el paso 6, para mejorar prestaciones en cada bloque funcional atendiendo a la arquitectura, se pueden abordar los siguientes puntos:

- Mejorar la construcción de los bucles. El conocimiento de la aplicación nos puede llevar a realizar construcciones de bucles más eficientes aplicando por ejemplo un "desenrollado de bucles" ([9], [18]) más apropiado o realizando una reordenación de instrucciones dentro del bucle para optimizar la utilización de las unidades funcionales.
- Utilización de instrucciones del procesador apropiadas a las operaciones realizadas en el programa y que el compilador no haya generado. Hay varios motivos por los que los compiladores no generan instrucciones que tiene el procesador. Puede por ejemplo, que el compilador no detecte todas las situaciones en las que resulta útil la instrucción, o puede que la desconozca por tratarse de una instrucción incluida en un procesador de reciente aparición. Por ejemplo, resulta complicado para los compiladores generar instrucciones SIMD, incorporadas en los noventa en DSPs (TMS320C80 de Texas Instruments en 1992) y procesadores de propósito general (MAX en PA de HP en 1993; VIS en Sparc de Sun en 1995; MMX en x86 de Intel en 1996). En [3], [6], [13] y [14] se pueden ver ejemplos del incremento en prestaciones obtenido en aplicaciones multimedia aprovechando principalmente la arquitectura SIMD.
- Analizar la penalización por predicción errónea en saltos. Se podría optar por cambiar el sentido de algún salto, o por sustituir construcciones if-then-else por instrucciones de movimiento condicional (*cmov*), si el compilador no las ha generado.
- Analizar la penalización por falta de alineación en los datos. Habrá que alinear los datos explícitamente en caso de ser necesario.
- Optimizar el acceso a memoria. El conocimiento de la aplicación nos permite detectar fácilmente si es eficiente almacenar los datos en memoria principal saltando las caches, y si resulta rentable precaptar (con la suficiente antelación) los datos con los que se va a operar.

Un análisis detallado sobre las optimizaciones que se pueden realizar para procesadores x86, se encuentra en [2] y [10].

Aquí se muestra como ejemplo, la mejora de una implementación de descodificador JPEG con pérdida "*baseline*" teniendo en cuenta la arquitectura. En la Sección 2 se describe brevemente la aplicación JPEG. La Sección 3 muestra las mejoras que se han ido obteniendo aplicando el proceso de la Figura 1. Por último, la Sección 5 muestra algunas conclusiones.

2. Descripción del descodificador JPEG

JPEG es un estándar internacional (1992) ISO (*International Organization for Standardization*) y ITU (*International Telecommunication Union*) para la compresión de imágenes [11]. Se utiliza con el fin de disminuir espacio de almacenamiento o tiempo de transmisión, en aplicaciones que

necesitan almacenar o transferir imágenes. Un descodificador JPEG, a partir de los datos de una imagen comprimida y de unas especificaciones de tablas (de cuantificación, Huffman), genera como salida la imagen reconstruida correspondiente. En la Figura 2 se pueden ver los bloques funcionales del descodificador JPEG: preprocesamiento, descodificador de entropía, descuantificación, IDCT (*Inverse Discrete Cosine Transform*) y postprocesamiento.

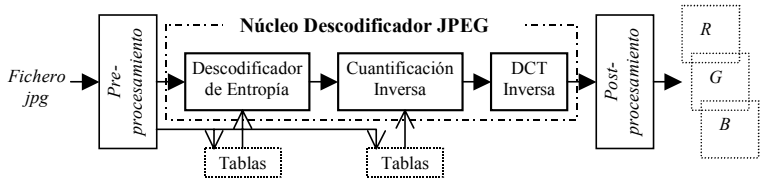

Figura 2. Bloques funcionales del descodificador JPEG

El núcleo del descodificador (entropía, descuantificación, IDCT) se aplica a nivel de bloques de 8x8 píxeles de la imagen. La imagen se irá formando a partir de los bloques 8x8 reconstruidos por este núcleo. En la práctica, una imagen a color se representa por varias componentes. Por ejemplo, en los computadores se utilizan imágenes representadas típicamente en formato RGB, que tiene tres componentes: rojo, verde y azul. JPEG, en lugar de RGB, prefiere trabajar con imágenes en color en formato YCbCr, porque mejora la razón de compresión (ver [11] y Capítulo 5 en [5]). Cada componente se codifica por separado, aplicando el proceso a bloques de 8x8. Para una imagen a color, el descodificador irá reconstruyendo cada componente de color a partir de los bloques 8x8 que va descodificando. A continuación se comenta brevemente cada etapa de procesamiento de la figura, una descripción detallada se encuentra en [5] y [11]:

1. *Preprocesamiento.* Esta etapa ha de conocer el formato de los ficheros JPEG. Debe extraer del fichero, además de las secuencias de bits que representa los bloques de imagen 8x8 comprimidos, información necesaria para el proceso de descodificación, como el tamaño de la imagen (o de las diferentes componentes de que consta la imagen) y las especificaciones de tablas utilizadas en el descodificador de entropía y descuantificador.
2. *Descodificador de entropía.* Deshace la codificación por longitud de ráfagas y codificación Huffman realizada por el codificador. Genera un bloque de 8x8 datos.
3. *Descuantificación*. Multiplica cada dato del bloque 8x8 generado en la etapa anterior, por un valor extraído de la tablas de descuantificación.
4. *IDCT*. Genera un bloque 8x8 (s_{yx}) aplicando al bloque 8x8 de entrada (S_{vu}) la transformada del coseno inversa:

$$s_{yx} = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} C_u C_v S_{vu} \cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16}$$

$$C_u, C_v = \begin{cases} 1/\sqrt{2} & u,v = 0 \\ 1 & u,v \neq 0 \end{cases} \qquad y,x = 0,1,...,7 \qquad (1)$$

5. *Postprocesamiento*. Ya que JPEG prefiere la codificación YCbCr y el computador el formato RGB, típicamente se debería realizar en esta etapa una conversión de formato YCbCr a RGB.

3. Mejora JPEG

En el proceso de mejora se han seguido los pasos de la Figura 1. La implementación de la aplicación se ha realizado en gcc, utilizándose en la compilación la última versión de djgpp v2.03 [8], con gcc 3.1 que incluye optimizaciones para Pentium. Estas optimizaciones se aplican si se compila con la opción *–mcpu=pentium*. Se han utilizado también las opciones de optimización *–O6* y *–fast-math*, recomendadas en la FAQ [8], y las opciones *–finline* y *–finline-function*. Para probar el funcionamiento del programa y para las posteriores medidas de prestaciones, se han buscado imágenes JPEG de distintos tamaños. Las medidas de prestaciones presentadas en esta

sección, se han generado con las tres imágenes que se pueden ver en las Figuras 3, 4 y 5, en un procesador Pentium de la familia P6 a 750 MHz con cache de nivel 2 de 256KB.

Figura 3. Sensation, de 651x507=330.057 píxeles

Figura 4. Gladiator, de 2044x1603=3.276.532 píxeles

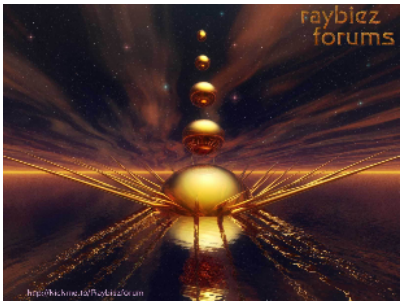

Figura 5. Rabiez, de 4800x3600=17.280.000 píxeles

El *paso 1* (Figura 1) acaba una vez que el programa generado, siguiendo el método habitual del estudiante, funciona correctamente. En el programa concreto implementado, la entrada es un fichero en formato jpg y la salida es un fichero (.raw) con la imagen RGB correspondiente a la imagen de entrada comprimida ([15]). En la transformada del coseno inversa y en la conversión a formato RGB se opera con datos punto flotante de simple precisión (32 bits), en la cuantificación se opera con datos enteros de 16 bits.

A continuación, en el *paso 2*, se incorporan en el código medidas que permiten determinar el tiempo que suponen los diferentes bloques funcionales de la aplicación (Figura 2): pre-procesamiento, descodificador de entropía, descuantificación, IDCT y postprocesamiento. Concretamente, se mide para cada bloque funcional y para el total del programa, el número de *ciclos de reloj del procesador* que consume su ejecución. Este número de ciclos se obtiene utilizando la instrucción ensamblador *rdtsc*, que devuelve el número de ciclos transcurridos desde que se reinició el sistema. Así se obtiene una precisión a nivel de ciclo de reloj, y las medidas son independientes de la frecuencia de reloj del procesador.

En las Tablas 1, 2 y 3, se muestran los ciclos de reloj generados a partir distintas versiones del programa ejecutable JPEG. Las versiones etiquetadas *0* y *1*, se han obtenido con el programa fuente inicial generado en la 1ª etapa (Figura 1). En la *versión 0* no se ha usado ninguna optimización del compilador, para *1* se han utilizado las opciones del compilador indicadas más arriba. Estas opciones se han empleado también para el resto de versiones, que han sido generadas a partir de los fuentes derivados en las sucesivas iteraciones del ciclo de la Figura 1.

La *versión* 1 del ejecutable presenta una ganancia en prestaciones de aproximadamente 1.36 frente a la versión 0. Para facilitar la comparación entre las diferentes versiones de código y entre los diferentes bloques funcionales en cada versión, se muestra en la Figura 6 la ganancia en prestaciones obtenida con las versiones 2, 3 y 4 sobre la versión inicial no mejorada 1, y en la Figura 7 un gráfico de barras apiladas para una de las imágenes. Destacar que en la etapa de postprocesamiento se incluye no sólo la conversión de formato YCbCr a formato RGB, sino el tiempo que supone almacenar la imagen RGB en un fichero. En las tablas se indica entre paréntesis para la versión 4, el tiempo que supone exclusivamente la conversión a RGB.

En la *primera iteración* del ciclo de la Figura 1, se genera la *versión 2*. En el *paso 3*, se obtienen

los ciclos de reloj para la *versión 1*. Como se puede observar en las Tablas 1, 2 y 3, y en la Figura 7, para la versión 1, destaca especialmente el tiempo que supone la IDCT. El código de la IDCT implementa la ecuación (1) mediante dos bucles. Analizando el código de la IDCT en el *punto 6* encontramos, que este bucle realiza además de una conversión de entero a punto flotante, cuatro multiplicaciones y una suma punto flotante en secuencia. Dado la gestión como pila de los registros punto flotante, habrá dependencia de datos entre una instrucción punto flotante y la siguiente, lo que impide que se puedan ejecutar solapadas. Para mejorar el código, se ha eliminado de los bucles la multiplicación de Cu y Cv. Esta reducción de las operaciones punto flotante con dependencias, ha mejorado sustancialmente las prestaciones, como ilustra la entrada para la *versión 2* (IDCT,Total) de las tablas y la Figura 7.

Versión	PreProc.	Entropía	Cuantifica.	IDCT	PostProc.	Total	
0	22.231.304	31.949.307	24.986.927	1.103.095.183	238.439.287	1.420.702.008	1,8943 seg.
1	11.819.328	22.927.176	5.251.895	804.966.456	188.421.065	1.033.385.920	1,3778 seg.
2	11.998.307	23.056.586	5.264.861	476.696.138	185.279.810	702.295.702	0,9364 seg.
3	12.581.971	22.713.347	5.299.239	22.417.991	80.313.005	143.325.553	0,1911 seg.
4	12.196.677	23.201.289	5.515.107	24.209.187	58.861.170 (45.594.759)	123.983.430	0,1653 seg.

Tabla 1. Ciclos consumidos en la ejecución de distintas versiones del programa ejecutable jpeg para la imagen *sensation.jpg* de tamaño 651x507 píxeles. El total se muestra también en segundos.

Versión	PreProc.	Entropía	Cuantifica.	IDCT	PostProc.	Total	
0	199.447.421	210.624.617	248.061.269	10.887.416.445	2.268.030.990	13.813.580.742	18,418 seg.
1	91.872.159	153.737.675	52.452.921	7.941.436.233	1.880.995.067	10.120.494.055	13,494 seg.
2	91.907.311	151.360.028	52.527.640	4.687.296.770	1.864.541.342	6.847.633.091	9,130 seg.
3	102.988.656	158.353.327	56.500.800	223.694.530	840.953.735	1.382.491.048	1,843 seg.
4	93.466.460	156.428.114	53.190.553	224.931.647	675.947.272 (431.555.659)	1.203.964.046	1,605 seg.

Tabla 2. Ciclos consumidos en la ejecución de distintas versiones del programa ejecutable jpeg para la imagen *gladiator.jpg* de tamaño 2044x1603 píxeles. El total se muestra también en segundos.

V	PreProc.	Entropía	Cuantifica.	IDCT	PostProc.	Total	
0	1.058.033.739	1.245.022.425	1.300.937.763	56.790.821.647	12.149.408.467	72.544.224.041	96,73 seg.
1	494.082.363	911.295.916	273.857.193	41.375.100.757	10.245.955.535	53.300.291.764	71,07 seg.
2	491.510.464	914.454.782	275.542.000	24.308.083.918	10.070.236.280	36.059.827.444	48,08 seg.
3	526.428.886	932.158.849	280.805.647	1.159.061.793	4.575.981.152	7.474.436.327	9,97 seg.
4	510.515.417	919.902.202	280.914.782	1.169.168.836	4.454.250.089 (2.456.779.992)	7.269.502.442	9,69 seg.

Tabla 3. Ciclos consumidos en la ejecución de distintas versiones del programa ejecutable jpeg para la imagen *rabiez.jpg* de tamaño 4800x3600 píxeles. El total se muestra también en segundos.

En la *segunda iteración* del ciclo de optimización, se parte de la *versión 2* y se genera la *3* mejorando los bloques funcionales más lentos: IDCT y Postprocesamiento. En esta iteración, para mejorar prestaciones se ha optado por operar con aritmética entera en lugar de punto flotante de simple precisión y por la utilizado instrucciones SIMD del repertorio MMX de la familia x86. En

las operaciones se ha utilizado precisión de 16 bits, con esta precisión se obtiene una imagen reconstruida de calidad similar a la generada con aritmética punto flotante de simple precisión (ver Capítulo 5 en [5]). Utilizando enteros se evita el coste de las operaciones de conversión entre tipos de datos, y utilizando SIMD se pueden llegar a realizar cuatro operaciones en paralelo. Tanto en la IDCT, como en la conversión YCbCr a RGB, se emplean sumas de multiplicaciones. Para acelerar estos bloques se ha utilizado la instrucción MMX *pmaddwd*, que realiza cuatro multiplicaciones en paralelo seguidas de dos sumas en paralelo. Estas mejoras han supuesto una ganancia en prestaciones superior a 7 respecto a la *versión 1* no mejorada (Figura 6). La menor reducción de tiempo observada en el postprocesamiento respecto a la IDCT, es debido a que éste no sólo incluye conversión a RGB también el almacenamiento en un fichero de esta imagen RGB resultante.

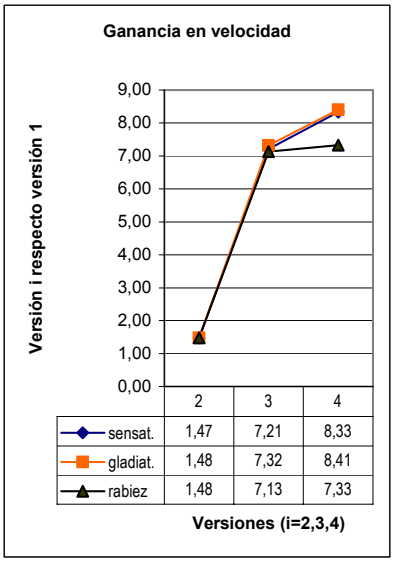

Figura 6. Ganancia en velocidad de las versiones 2, 3 y 4 del programa respecto a la *versión 1* para las tres imágenes

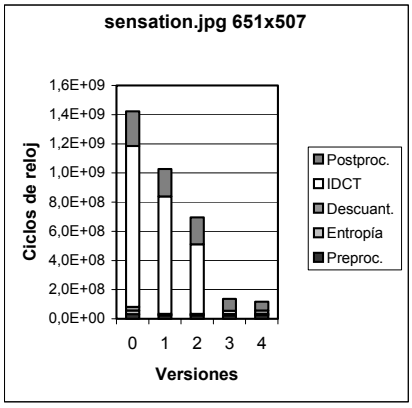

Figura 7. Ciclos de reloj para *sensation.jpg*

En la *tercera iteración* se parte de la *versión 3* generándose la *versión 4*. En la versión 3, la etapa de postprocesamiento domina en tiempo con respecto a las demás. Para mejorar esta etapa se ha separado la conversión a RGB del almacenamiento en un fichero, que se realizaban de forma solapada. Esto ha supuesto una reducción en tiempo, llegándose a ganancias superiores a 8. Con imágenes de gran tamaño, se incrementa la influencia del almacenamiento en disco en el postprocesamiento. Lo que se puede comprobar, comparando en la entrada 4 de las tablas, el tiempo de la conversión RGB (entre paréntesis) con el tiempo de postprocesamiento. Por este motivo, para rabiez.jpg, la ganancia es sólo de 7,33.

4. Conclusiones

Algunos de los trabajos fin de carrera que proponemos dentro del área de Arquitectura y Tecnología de Computadores, persiguen que los estudiantes comprueben que pueden emplear los conocimientos de arquitectura que han adquirido, para mejorar las prestaciones de sus programas. Las mejoras en prestaciones conseguidas en estos trabajos fin de carrera, se muestran a estudiantes que actualmente cursan asignaturas de arquitectura y estructura de computadores, con el objetivo de motivarles en el estudio de estas materias.

En estos trabajos se propone la programación de una aplicación, que despierte interés para el estudiante, en dos etapas (Figura 1). En la primera etapa, el estudiante genera un programa que resuelve la aplicación siguiendo su método habitual. En la segunda etapa, el código generado por el estudiante se mejora teniendo en cuenta la arquitectura actual de los computadores. En esta etapa el estudiante aplica conocimientos que ha adquirido sobre arquitectura y estructura de computadores durante la titulación. Para este proceso de mejora proponemos utilizar los pasos especificados en la Figura 1.

Aquí se muestran como ejemplo, la mejora en prestaciones obtenida a partir de un programa generado por un estudiante para descodificar imágenes en formato jpeg "*baseline*" ([15]). Se ha obtenido en un único procesador, una ganancia en prestaciones entre 7 y 8 con respecto al programa original, pasando el tiempo de procesamiento para una imagen con más de tres millones de píxeles de 13,49 segundos a 1,6 segundos en un Pentium de la familia P6 a 750MHz. Estas mejoras se han conseguido en menos de una semana, mientras que el estudio de la aplicación y su programación han llevado cerda de dos meses. Los estudiantes se sorprenden al ver que en poco tiempo, se pueden conseguir ganancias de 8 con un único procesador.

Actualmente se están desarrollando otros trabajos fin de carrera de optimización y se está aplicando esta idea en una práctica de 2 horas en la asignatura Estructura de los Computadores. En esta práctica, se optimiza el cálculo del mínimo de una lista de números usando una instrucción de movimiento condicional. La ganancia que se obtiene es superior a 2.

Referencias

[1] CodeAnalyst, http://www.amd.com/us-en/Processors/DevelopWithAMD/0,,30_2252_3604,00.html

[2] "AMD Athlon™ Processor x86 Code Optimization Guide", http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_1274_3734^3748,00.html

[3] Akramullah, S. M. ; Ahmad, I. ; Liou, M. L. "Optimization of H.263 Video Encoding Using a Single Processor Computer: Performance Tradeoffs and Benchmarking", IEEE Transactions on Circuits and Systems for Video Technology, vol. 11, n. 8, pp. 901-915, August 2001.

[5] Bhaskaran, V. ; Konstantinides K., "Image and Video Compression Standars", Kluwer.1999.

[6] Casalino, F. et al. "MPEG-4 Video Decoder Optimization", in Proc. IEEE Int. Conf. Multimedia Computing and Systems, vol.1, pp. 363-368, 1999.

[7] CodeAnalyst de AMD, ttp://www.amd.com/us-en/Processors/DevelopWithAMD/0,,30_2252_3604,00.html

[8] Djgpp home, http://www.delorie.com/djgpp/

[9] Dowd, K.; Severance C.R., Hihg Performance Computing", O'Really, 1998.

[10] "Intel® Pentium® 4 Processor Optimization Reference Manual", http://www.intel.com/design/pentium4/manuals/248966.htm

[11] ITU-T Recommendation T.81 y T.84, "Digital Compression and Coding of Continous-Tone Still Images", 1992.

[12] de Juan Herrero, J. "Introducción a la enseñanza universitaria. Didáctica para la formación del profesorado", Dykinson S.L., Madrid, 1995.

[13] Lappalainen, V. ; "Peformance of an Advanced Video Codec on a General-Purpose Processor with Media ISA Extensions", in IEEE Transactions on Consumer Electronics, vol. 46, no. 3, pp. 706-716, August 2000.

[14] Lappalainen, V. ; Hämäläinen, T. D. ; P. Liuha. "Overview of Research Efforts on Media ISA Extensions and Their Usage in Video Coding", in IEEE Transactions on Circuits and Systems for Video Technology, vol. 12, n. 8, pp. 660-670, August 2002.

[15] Padilla Pérez, J.A. "JPEG con MMX", Proyecto Fin de Carrera, ETSI Informática de Granada. Curso 2001-2002.

[16] VSDK de sun, http://www.sun.com/processors/vis/vsdk.html

[17] Vtune de Intel, http://developer.intel.com/software/products/vtune/

[18] Wadleigh, K.R., Crawford, I. L.; "Software optimization for High Performance Computing", Hewlett-Packard Professional Books, 2000.