MATEUS KREPSKY LUDWICH

ADAPTAÇÃO DE UM DECODIFICADOR DE VÍDEO DIGITAL A UMA INTERFACE DE PROCESSAMENTO DIGITAL DE SINAIS MULTIPLATAFORMA

Florianópolis – SC

Julho 2008

MATEUS KREPSKY LUDWICH

ADAPTAÇÃO DE UM DECODIFICADOR DE VÍDEO DIGITAL A UMA INTERFACE DE PROCESSAMENTO DIGITAL DE SINAIS MULTIPLATAFORMA

Trabalho de Conclusão de Curso submetido à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciências da Computação.

Professor Dr. Antônio Augusto Medeiros Fröhich

Co-orientador:
Danillo Moura Santos

BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSCA
CENTRO TECNOLÓGICO
UNIVERSIDADE FEDERAL DE SANTA CATARINA

Florianópolis

Julho 2008

Trabalho de Conclusão de Curso sob o título *"Adaptação de um Decodificador de Vídeo Digital a uma Interface de Processamento Digital de Sinais Multiplataforma"*, defendido por Mateus Krepsky Ludwich e aprovado em 11 de Julho de 2008, em Florianópolis, Santa Catarina, pela banca examinadora constituída por:

Prof. Antônio Augusto Medeiros Fröhich, D.Sc.
Orientador

Danillo Moura, B.Sc.
Co-Orientador
Universidade Federal de Santa Catarina

Hugo Marcondes, B.Sc.
Universidade Federal de Santa Catarina

Lucas Wanner, M.Sc.
Universidade Federal de Santa Catarina

Prof. Roberto Willrich, Ph.D.
Universidade Federal de Santa Catarina

À minha mãe Eleonora Kátia Krepsky Ludwich
e ao meu pai Adriano Brognoli Ludwich

AGRADECIMENTOS

Agradeço ao meu orientador Antônio Augusto Medeiros Fröhich e ao meu co-orientador Danillo Moura Santos por acreditarem no meu trabalho, pelas reuniões e esclarecimento de dúvidas e pela oportunidade de trabalhar junto ao Laboratório de Integração de Software e Hardware - LISHA. Agradeço ao pessoal do LISHA pelas sugestões relativas a este trabalho. Aos meus amigos pelas conversas e pela compreensão nos momentos que estive ausente. Aos meus pais que sempre me apoiaram e continuam me apoiando ao longo da minha vida e carreira acadêmia. E finalmente a Deus por proporcionar todas estas coisas e por estar sempre presente na minha vida.

RESUMO

Em codecs de vídeo os estágios de processamento digital de sinais são os mais complexos do ponto de vista computacional. Comumente tais estágios são escritos total ou parcialmente em linguagem assembly de arquiteturas específicas. Isto ao mesmo passo que contribui para um aumento no desempenho, reduz a portabilidade dos codecs. A utilização de uma API que concentre em si funcionalidades de processamento digital de sinais (DSP) e primitivas de processadores digitais de sinais (DSP-HW), pode contribuir no aumento da portabilidade dos codecs. Este trabalho aplicou uma API DSP (DSP/DSP-HW) em um decodificador de vídeo MPEG-2. Modificou-se o estágio da realização da Transformada Inversa Discreta de Cosseno (IDCT), para usar a API. Executou-se sobre esta modificação o teste de conformidade descrito no padrão MPEG-2 parte 4. O decodificador modificado demonstrou-se correto de acordo com o teste. Realizou-se também uma análise de desempenho comparando o decodificador modificado com o original. O desempenho do decodificador modificado mostrou-se satisfatório. Outro estágio do MPEG-2 trabalhado foi a compensação de movimento (MC), sobre o qual foram gerados apontamentos de onde é possível a utilização da API.

ABSTRACT

In video codecs the stages of digital signal processing are the more complex in the computational sense. Commonly these stages are written total or partially in assembly language of specific architectures. At the same time that this contributes for a performance increase, this reduces the portability of the codecs. The utilization of a API that provides features of digital signal processing (DSP) and primitive of digital signal processors, can contribute in increasing the portability of codecs.
This work applied a DSP API (DSP/DSP-HW) in a MPEG-2 video decoder. The stage of Inverse Discrete Cosine Transform (IDCT) was modified to use the API. The MPEG-2 part 4 conformance test for IDCT was applied on this modification. The modified decoder passed in this test. A performance analysis also was made, where the modified decoder was compared with the original one. The performance of modified decoder was satisfactory. Another stage of MPEG-2 that was worked was the motion compensation (MC). Then was generated notes where the utilization of the API is possible in this stage.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

Advanced Audio Coding (AAC) Application Programming Interface (API)
Concealment Motion Vector (CMV)
Discrete Cosine Transform (DCT)
Differential Motion Vector (DMV)
Digital Signal Processing (DSP)
Digital Storage Media Command and Control (DSM-CC) Digital Video Disc (DVD)
Discrete Wavelet Transform (DWT)
Forward Discrete Cosine Transform (FDCT)
Fast Fourier Transform (FFT)
Field-Programmable Gate Array (FPGA)
frames per second (fps)
Group of Pictures (GOP)
Inverse Discrete Cosine Transform (IDCT)
International Electrotechnical Commission (IEC)
Institute of Electrical and Electronic Engineers (IEEE)
International Organization for Standardization (ISO)
MAC Multiplication Accumulator (MAC)
Motion Compensation (MC)
Moving Picture Experts Group (MPEG)
Motion Vector (MV)
Variable Length Coding (VLC)

SUMÁRIO

1 INTRODUÇÃO

Atualmente a utilização de vídeo em formato digital é ampla. Tanto em termos de dispositivos de obtenção de vídeo digital como câmeras e dispositivos reprodutores como DVD players e o próprio computador. Entretanto a transmissão e o armazenamento de vídeos digitais em seu formato bruto é na maioria dos casos inviável devido as altas taxas de transmissão necessárias e as grandes quantidades de memória exigidas para seu armazenamento. Neste contexto se faz necessário a compressão e descompressão dos vídeos digitais, o que ocorre na prática nos processos de codificação e decodificação dos mesmos. Estes processos por lidarem com sinais de vídeo utilizam determinadas funções matemáticas conhecidas como funções de Processamento Digital de Sinais (DSP). O componentes que implementam tais processos são chamados de codificadores/decodificadores (codecs).

O Moving Picture Experts Group, ou simplesmente MPEG, é um grupo de trabalho ISO/IEC que ao longo de sua história tem desenvolvido diversos padrões de codificação tanto de áudio como de vídeo digital. Dentre estes padrões temos o MPEG-2, cuja parte 2, especifica como codificar e decodificar vídeo digital.

Codecs podem ser implementados como sendo software, ou como sendo hardware, ou como ambos. No caso das implementações em software é interessante que estas sejam portáveis para as diversas plataformas de hardware existentes, como processadores de uso geral, sistemas dedicados desenvolvidos em lógica programável e processadores digitais de sinais com instruções dedicadas. É importante entretanto que o desempenho do codec seja satisfatório nas diversas plataformas.

No presente trabalho buscou-se adaptar um decodificador de vídeo padrão MPEG-2 previamente existente a uma Interface de Programação de Aplicativos (API) que provê funções DSP e que contempla os requisitos de portabilidade e desempenho desejados pelo decodificador.

1.1 OBJETIVOS

1.1.1 OBJETIVO GERAL

Adaptar um decodificador de vídeo digital padrão MPEG-2 a uma API DSP multiplataforma, mapeando as funções DSP do decodificador nas funções providas pela API.

1.1.2 OBJETIVOS ESPECÍFICOS

- Identificar em um decodificador MPEG-2 as primitivas de DSP.
- Modificar o decodificador para que use as funções providas pela API, no lugar de suas implementações originais.
- Verificar a corretude das modificações realizadas.

1.2 ESTRUTURA DO TRABALHO

Os capítulos que seguem estão organizados da seguinte maneira. Capítulo 2 traz a a fundamentação teórica do trabalho, sendo dividido em duas seções principais. A seção 2.1, dá uma rápida visão dos conceitos gerais de vídeo digital, assim como evidencia a necessidade de compressão do mesmo. A seção 2.2, menciona o grupo MPEG, focando na explicação do padrão MPEG-2 parte 2 (vídeo).

O capítulo 3 descreve os pontos trabalhados do decodificador MPEG-2 escolhido. E o capítulo 4 faz as considerações finais e conclui o trabalho.

2 REVISÃO TEÓRICA

2.1 VÍDEO DIGITAL

Como consta em (SALOMON, 2004), vídeo digital é, em princípio, uma seqüência de quadros (frames) apresentados em uma determinada taxa (frame rate), geralmente medida em quadros por segundo (fps), criando assim a ilusão de animação. Vídeos digitais podem ser obtidos a partir de dispositivos como câmeras, que capturam sinais analógicos do mundo externo e os transformam para sinais digitais, ou podem ser gerados artificialmente, com o auxílio do computador.

Cada quadro em um vídeo é formado por um conjunto de pontos chamados de pixels. Pixel (aglutinação de Pix - abreviatura de Picture e Element), neste caso, significa a menor unidade da imagem (quadro). Um pixel pode ser visto como um pequeno ponto que possui uma determinada cor. Para representar a cor de um pixel de modo digital é necessária uma determinada quantidade de bits. A esta propriedade da-se o nome de profundidade de cor. Em princípio quanto mais bits forem usados para representar um pixel, uma maior variedade de cores pode ser representada na imagem. Os pixels em um quadro estão dispostos na horizontal e na vertical, formando uma imagem bidimensional retangular. O produto da quantidade de pixels na largura e da quantidade de pixels na altura define a chamada resolução do quadro (pixels na largura X pixels na altura).

Vídeos digitais podem obviamente conter áudio e outros elementos como legendas. Neste trabalho está sendo considerado apenas a imagem do vídeo digital.

2.1.1 COMPRESSÃO

Com estas três propriedades básicas: resolução, profundidade de cor e taxa de apresentação, mais o tempo de duração do vídeo é possível calcular o tamanho lógico de um arquivo de vídeo. O tamanho é dito lógico por não considerar a unidade mínima de alocação no sistema de arquivos e conseqüentemente no disco. O tamanho lógico de um arquivo de vídeo (em bits) é

calculado da seguinte forma:

$$tamanho\_logico = resolucao \times profundidade\_de\_cor \times taxa\_de\_apresentacao \times duracao \quad (2.1)$$

Lembrando que a resolução é expressa em pixels, a profundidade de cor em bits, a taxa de apresentação em bits por segundo e a duração em segundos. Um vídeo de uma hora, por exemplo, com uma taxa de apresentação de 24 fps, resolução de 640x480 pixels, e profundidade de cor de 24 bits, gera um arquivo de 637009920000 bits, ou seja, aproximadamente 74 giga bytes, o que representa um arquivo relativamente grande mesmo nos atuais discos rígidos convencionais. Se considerarmos a duração deste mesmo vídeo como um segundo, temos que a taxa necessária para a transmissão do mesmo em tempo real deve ser de no mínimo 169 MBps, que é impraticável na grande maioria das redes atuais. Este pequeno exemplo indica a necessidade de compressão de vídeo digital, para que o mesmo não ocupe grandes quantidades de espaço de armazenamento e para que seja viável sua transmissão em tempo real.

De acordo com (SALOMON, 2004) a compressão de vídeo está baseada em dois princípios: a redundância espacial e a redundância temporal contida nos vídeos digitais. Redundância espacial é entendida como a semelhança em cor entre um pixel e seus vizinhos (tanto horizontal como vertical). Esta característica é bastante presente em imagens fotográficas, proveniente de câmeras de vídeo, em oposição as imagens geradas por computador onde, dependendo do que se cria, pode não existir semelhança entre pixels vizinhos. Já a redundância temporal está associada ao fato de que quadros vizinhos em um vídeo, ou seja, quadros obtidos em instantes de tempo próximos, são bastante semelhantes entre si. Explorando estes dois tipos de redundância são desenvolvidas as várias técnicas de compressão de vídeo digital. Cada uma destas técnicas pode ser constituída em um ou mais algoritmos e o conjunto delas foma uma especificação ou padrão de compressão de vídeo. O padrão de compressão de vídeo trabalhado neste projeto foi o MPEG-2, que será explanado mais adiante no texto.

2.2 MPEG

MPEG é a sigla usada para Moving Picture Experts Group, que trata-se de um grupo de trabalho ISO/IEC encarregado do desenvolvimento de padrões de codificação de áudio e vídeo. Até hoje diversos padrões foram desenvolvidos por este grupo, padrões que tratam de codificação de áudio e vídeo propriamente dita (e.g: MPEG-1, MPEG-2, MPEG-3, MPEG-4), padrões de descrição de conteúdo multimídia (MPEG-7) e padrões relacionados a propriedade intelectual e controle de acesso (MPEG-21). A descrição de todos estes padrões está fora do escopo deste trabalho, mas pode ser consultada, por exemplo, em (WOOTTON, 2005)

e (MPEG. . . , 2008).

O padrão MPEG utilizado no desenvolvimento deste trabalho é o MPEG-2, mas especificamente as partes condizentes ao vídeo e aos testes de conformidade. A seguir é apresentado o MPEG-2, com enfase nas partes trabalhadas.

2.2.1 MPEG-2

O MPEG-2, assim como outros padrões MPEG, está descrito em um documento constituído de várias partes, cada uma delas abordando um aspecto do padrão. Como conta em (MPEG. . . , 2008), atualmente o MPEG-2 é constituído de nove partes. São elas:

- Parte 1 - Sistemas
- Parte 2 - Vídeo
- Parte 3 - Áudio
- Parte 4 - Teste de conformidade
- Parte 5 - Simulação de Software
- Parte 6 - Extensões para DSM-CC
- Parte 7 - Codificação Avançada de Áudio
- Parte 9 - Extensão para interface de tempo real para sistemas decodificadores
- Parte 10 - Extensões de conformidade para DSM-CC

Como este projeto trata-se de uma adaptação de um decodificador de vídeo, foram trabalhadas as partes 2 - vídeo (ISO/IEC, 1995) e 4 - teste de conformidade (ISO/IEC, 1998) do padrão. Conceitos da parte 2 do padrão são tratados a seguir. A parte 4 é mencionada em 3.3.3.

HIERARQUIA DO FLUXO DE DADOS (BISTREAM)

Um vídeo MPEG-2 pode ser constituído de uma ou mais camadas (layers) de fluxo de dados (bitstream). Um vídeo que seja constituído de apenas uma camada de bitstream é chamado de não-escalável. A idéia de escalabilidade, implementada com o mecanismo de diversas camadas está associada à qualidade do vídeo. Um vídeo escalável possui na primeira camada de bitstream um mínimo de qualidade, que vai sendo aprimorada com os dados provenientes das camadas superiores. Cada bistream obedece uma estrutura sintática semelhante a esta:

$$sequencia\_de\_video \rightarrow grupo\_de\_imagens \rightarrow imagem \rightarrow slice \rightarrow macrobloco \rightarrow bloco \rightarrow amostra \tag{2.2}$$

- Seqüência de vídeo (Video sequence)
  Trata-se da estrutura de mais alto nível do bistream. Uma seqüência de vídeo inicia-se com um cabeçalho que pode ser seguido de um cabeçalho de grupo de imagens e então por um ou mais imagens codificadas. A ordem em que as imagens codificadas estão no bistream é a ordem em que o decodificador as processa, mas não é necessariamente a ordem de apresentação das mesmas. Uma seqüência de vídeo termina com um código de fim de seqüência.

- Grupo de Imagens (Group of Pictures)
  Grupo de Imagens (Group of Pictures - GOP) é a estrutura contida na seqüência de vídeo, composto por um cabeçalho e por imagens (pictures) que podem ser, no MPEG-2, de três tipos: I, P, B (explicados em seguida). Na realidade a noção formal de GOP está presente apenas no MPEG-1, sendo que o MPEG-2 não exige esta estrutura. Mas, caso seja desejado, é possível no MPEG-2 obter o efeito do GOP utilizando os chamados cabeçalhos GOP.
  De acordo com (WOOTTON, 2005) um GOP é formado por mais de 10 e menos de 30 frames. Um GOP muito pequeno reduz a taxa de compressão, já que cada GOP começa com uma imagem I (como será visto mais adiante uma imagem I é maior que uma imagem P ou B). Por outro lado um GOP muito grande, se tiver algum erro, este erro se propagará/durará mais tempo. Quando se cria vídeo MPEG-2 para DVDs, o comprimento dos GOPs deve estar de acordo com os padrões de vídeo PAL ou NTSC. Para o sitema PAL um GOP não deve ter mais de 15 quadros, e para um sistema NTSC este valor não deve ser maior que 18.

- Imagem (Picture)
  Uma imagem fonte ou reconstruída por um processo de decodificação consiste em três

matrizes retangulares de números de 8 bits. Uma matriz representa o sinal de luminância (Y) e as outras duas os sinais de crominância (Cb, Cr). Luminância é usada para descrever o brilho de um sinal de vídeo relativo a um valor de pico de branco (de luz branca). Crominância por sua vez está associada ao valor das cores do sinal de vídeo. Existe também o conceito de imagem codificada, que é a imagem de fato presente no bistream. Neste texto quando for mencionado apenas a palavra "imagem" está se referindo a imagem codificada.

Uma imagem pode ser de dois tipos, tipo quadro (frame picture) ou tipo campo (field picture). Os conceitos de quadro e campo estão associados com modo como o vídeo é desenhado na tela. No chamado esquadrinhamento progressivo a tela é preenchida desenhando-se todas as linhas, de cima para baixo, formando o que é conhecido como quadro. Entretanto quando uma varredura na tela do vídeo não desenha um quadro e sim metade dele por vez, desenhando-se primeiro todas as linhas ímpares do quadro e depois todas as linhas pares (ou todas as linhas pares e depois as ímpares) temos o chamado esquadrinhamento entrelaçado. Este meio-quadro, formado pelas linhas pares ou ímpares de um quadro, é chamado de campo.

Cada imagem, seja ela do tipo quadro ou campo, ainda segue uma outra classificação, relativa ao grau de dependência com outras imagens (codificadas ou fontes) no momento de sua codificação (e conseqüentemente decodificação). De acordo com esta classificação, o MPEG-2 determina os seguintes tipos de imagem:

- Tipo I (Intra-coded)
  Uma imagem do tipo I é codificada (e portanto decodificada) de forma independente de qualquer outra imagem.
- Tipo P (Predictive)
  Uma imagem do tipo P é codificada usando-se predição da compensação do movimento em cima de uma imagem de referência passada. Predição da compensação do movimento é o mesmo que estivativa de movimento, conceito explicado em 2.2.3. O seguinte exemplo pode ilustrar melhor este tipo de imagem.
  Dado três imagens fontes A, B, C. Considerando que A seja usado na integra, então ela será a primeira imagem I, chamada por exemplo de I1. Supondo que é calculada a diferença B - A; o resultado é a primeira imagem P, chamado por exemplo de P1. Supondo que é calculada a diferença C - B; o resultado é a segunda imagem P, e assim sucessivamente. Assim teria-se:
  Fonte: A, B, C
  Resultado: A, (B - A), (C - B) == I1, P1, P2

– Tipo B (Birectional)

  Uma imagem do tipo B, assim como uma imagem do tipo P é codificada usando predição da compensação do movimento. Entretanto neste caso é usada uma imagem de referência passada e uma imagem de referência futura, por isto o termo bidirecional. O seguinte exemplo pode ilustrar melhor este tipo de imagem.

  Dado quatro imagens fontes A, B, C, E. Considerando I1 uma imagem I obtida a partir de A, e (E - A) uma imagem P chamada de P1. Os frames B's serão:

  B1 == diff(P1, B, I1)

  B2 == diff(P1, C, I1)

  B3 == diff(P1, D, I1)

  Fonte: A, B, C, D, E

  Resultado: I1, B1, B2, B3, P1 As imagens B's codificam as diferenças entre a última imagem I ou P anterior, com a próxima imagem I ou P e com uma imagem original de vídeo que fique entre estas duas imagens "última anterior" e "próxima". Assim, as imagens B's podem estar entre duas imagens P's. No exemplo mostrado as imagens B's ficam entre uma imagem do tipo I e uma imagem do tipo P.

A seguinte figura ilustra a geração de imagens P e B.

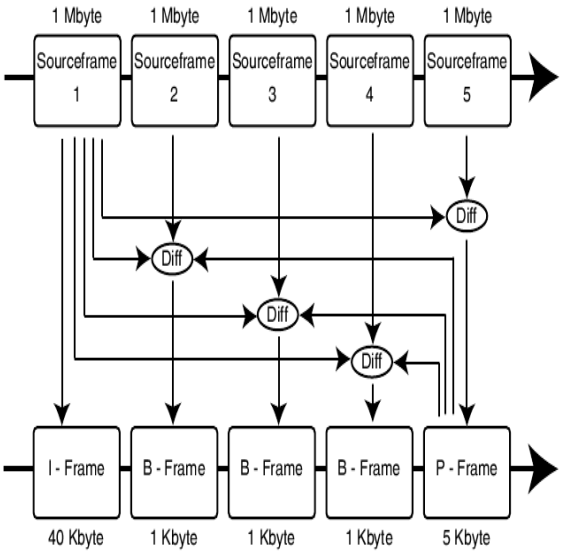

Figura 1: Geração de imagens P e B (WOOTTON, 2005).

Pode-se observar que uma imagem I é maior que uma imagem P ou B, já que não necessita de outras imagens para ser codificada. De forma semelhante uma imagem P é maior que uma imagem B, pois é formada a partir de menos imagens.
Imagens do tipo I são também chamadas de "imagens intra codificadas" (intra coding pictures) e imagens do tipo B e P são chamadas de "imagens não-intra codificadas" (non-intra coding pictures) ou ainda "imagens inter codificadas". Esta mesma classificação de intra coding e non-intra coding também se aplica para macroblocos e blocos, já que eles fazem parte da imagem.

- Slice
  Um slice é um pedaço da imagem. Ele é constituído de macroblocos. No MPEG-2 um slice não podem ocupar mais que uma linha da imagem e não pode estar parte em uma linha e parte em outra. Exitem duas estruturas de slice, uma chamada de geral e outra chamada de restrita. A estrutura restrita exige que toda a área de uma imagem seja dividida em slices, já na estrutura mais geral, é possível haver apenas alguns slices cobrindo a imagem, sendo que apenas estes são codificados.

- Macrobloco (Macroblock)
  Um macrobloco é uma estrutura formada por blocos que contém informações de luminância do sinal de vídeo e por blocos, espacialmente correspondentes, que contém informações de crominância. Existem três formatos de crominância para um macrobloco, chamados de 4:2:0, 4:2:2 e 4:4:4. São chamados de formatos de crominância pois é este componente que se altera, o componente de luminância permanece o mesmo em todos estes formatos, isto é, em todos eles existem quatro blocos de luminância, sendo estes sempre os quatro primeiros blocos do macrobloco. Abaixo segue o número de blocos para cada formato e as respectivas ordens dos blocos.
  Um macrobloco 4:2:0 é constituído de quatro blocos de luminância, 4*Y e dois de crominância, 1*Cb mais 1*Cr, totalizando seis blocos. A ordem dos blocos é a seguinte: 0 Y, 1 Y, 2 Y, 3 Y, 4 Cb, 5 Cr, ou seja, primeiro (índice zero) até quarto de luminância, quinto crominância relativa ao azul e sexto crominância relativa ao vermelho. A seguinte figura ilustra estrutura de um macrobloco 4:2:0.

Figura 2: Estrutura de um macrobloco 4:2:0 (ISO/IEC, 1995).

Um bloco 4:2:2 é constituído de 4*Y + 2*Cb + 2*Cr, totalizando oito blocos. A seguinte figura ilustra estrutura de um macrobloco 4:2:2.

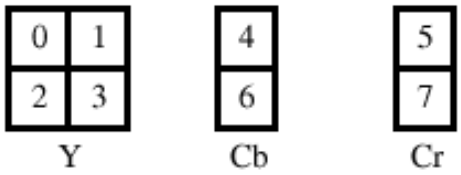

Figura 3: Estrutura de um macrobloco 4:2:2 (ISO/IEC, 1995).

Um bloco 4:4:4 é constituído de 4*Y + 4*Cb + 4*Cr, totalizando doze blocos. A seguinte figura ilustra estrutura de um macrobloco 4:4:4.

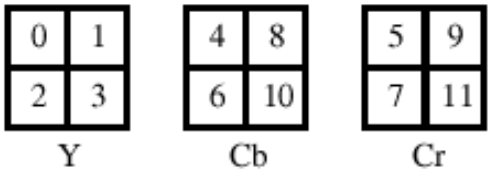

Figura 4: Estrutura de um macrobloco 4:4:4 (ISO/IEC, 1995).

É interessante notar que na ordem dos blocos existe sempre uma alternância entre os blocos de crominância Cb e Cr.
O termo macrobloco pode ser usado tanto para se referir a dados fonte (ou reconstruído), quanto a dados codificados.

- Bloco (Block)
  Assim como o termo macrobloco, "bloco" pode ser usado tanto para se referir a dados fonte (ou reconstruído), quanto a dados codificados. Um bloco no MPEG-2 possui oito linhas e oito colunas, totalizando um número de sessenta e quatro elementos. Em um bloco

fonte/reconstruído cada elemento representa uma amostra do sinal de vídeo. Um bloco fonte/reconstruído pode conter ou amostras de luminância ou amostras de crominância. O primeiro elemento de um bloco (i.e. índice zero para linha e coluna) é chamado de DC, os outros elementos que não o primeiro são chamados de AC.

- Amostra (sample)
  Em um bloco fonte ou reconstruído, uma amostra significa a representação digital de uma parte do sinal de vídeo, sendo da componente de luminância ou de crominância. Como os dispositivos de visualização de vídeo (i.e. monitores) trabalham com pixels codificados em RGB (Red Green Blue) os decodificadores de MPEG-2 geralmente possuem a capacidade de converter uma imagem constituída de lunimância e crominância YCbCr para uma equivalente em RGB. É possível também armazenar em arquivo a imagem RGB obtida. O inverso, ou seja, a conversão de RGB para YCbCr também é possível, e pode ser necessária em dispositivos que capturam vídeo e o codificam em RGB. Por esta razão é comum codificadores MPEG-2 disponibilizarem a conversão RGB para YCbCr.

DECODIFICAÇÃO

Como este trabalho lida com um decodificador MPEG-2, a compreensão do processo de decodificação é de grande importância. Aqui é feito um resumo de como este processo ocorre para um vídeo não-escalável (constituído de apenas uma camada de bitstream). Em (ISO/IEC, 1995) seções 7.7 a 7.11 são contempladas as extensões de escalabilidade da decodificação MPEG-2, não relevantes para este trabalho.

A decodificação de um bitstream MPEG-2 é composta dos seguintes estágios:

1. Decodificação de comprimento variável (Variable Length Decoding)
2. Esquadrinhamento inverso (Inverse Scan)
3. Quantização inversa (Inverse Quantisation)
4. IDCT
5. Compensação de movimento (Motion Compensation)

Cada um destes estágios é brevemente explicado a seguir, sendo que os estágios IDCT e Compensação de movimento, por terem uma importância maior neste trabalho, são explicados em mais detalhes, respectivamente, em 2.2.2 e 2.2.3.

Nesta parte do texto e em 2.2.2 e 2.2.3 utiliza-se a mesma notação presente em (ISO/IEC,

1995), no que diz respeito as entradas/saídas de cada estágio da decodificação. A notação diz o seguinte: vetores bidimensionais são representados por *nome[q][p]*, onde "q" é o índice da dimensão vertical e "p" é o índice da dimensão horizontal.

1. Decodificação de comprimento variável

   A unidade a ser processada por este estágio é o bloco. O inicio deste é identificado através de seu cabeçalho e dos cabeçalhos das respectivas estruturas superiores a bloco (i.e. macrobloco, slice, etc). Ao entrar neste estágio o bloco está compactado, codificado com a chamada "Codificação de comprimento variável", do original Variable Length Coding, ou simplesmente, VLC. VLC é uma espécie de codificação Huffman (SALOMON, 2004, 2.8), no sentido de que os eventos que ocorrem com maior freqüência são mapeados para códigos de menor comprimento, e os eventos que ocorrem com menor freqüência são mapeados para códigos de maior comprimento. Eventos, neste caso, são cada um dos 64 elementos do bloco. O MPEG-2 especifica tabelas que são utilizadas neste processo (ISO/IEC, 1995). A (de)codificação do coeficiente DC de um bloco intra utiliza um conjunto de tabelas (tabelas B-12 e B13). A (de)codificação dos coeficientes AC ou do coeficiente DC de um bloco não intra utiliza um outro conjunto de tabelas (tabelas B-14, B-15, B-16). A saída deste estágio trata-se do bloco "expandido", porém ainda em um formato unidimensional. Cada elemento do bloco na saída desta etapa é denotado como QFS[n], onde n vai de 0 a 63.

2. Esquadrinhamento Inverso

   É neste estágio que um bloco recupera seu formato bidimensional. A entrada deste estágio é o QFS[n], e a saída é denotada como QF[v][u], onde v e u vão de 0 a 7. A matriz QF é montada a partir de uma varredura no vetor QFS, onde o primeiro elemento do vetor corresponderá ao valor da quina superior esquerda da matrix (QF[0][0]) e o último elemento do vetor resultará na quina direita inferior da matrix (QF[7][7]). No esquadrinhamento "não inverso" a matriz QF é varrida de acordo com um ziguezague (no caso de imagens progressivas), ou ziguezague modificado (no caso de imagens entrelaçadas) e então o vetor unidimensional QFS é gerado. No esquadrinhamento inverso, é justamente o oposto que acontece. As figuras abaixo ilustram, respectivamente, o ziguezague padrão e o ziguezague modificado.

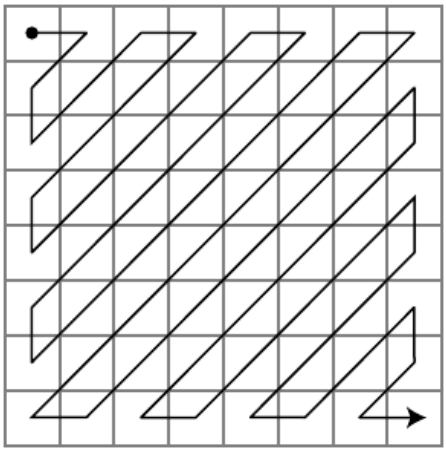

Figura 5: Ziguezague padrão (WOOTTON, 2005).

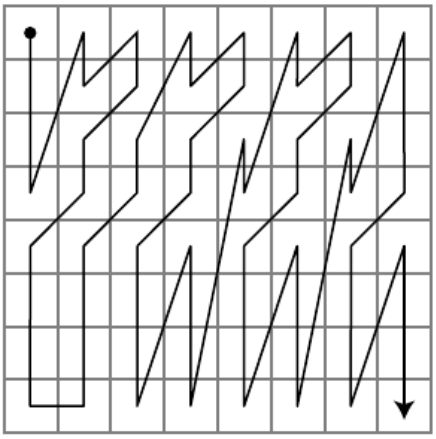

Figura 6: Ziguezague modificado (WOOTTON, 2005).

3. Quantização inversa

   A entrada neste estágio é uma matriz do tipo QF[v][u] e a saída é outra matriz, com as mesmas dimensões, denotada como F[v][u], que serve de entrada para a IDCT. Quantização inversa, como o próprio nome indica é o processo inverso da quantização. O objetivo da quantização é fazer com que os valores ao redor da quina superior esquerda da matriz sejam mantidos e que os valores ao redor da quina inferior direita tendam a zero. Isto é obtido submetendo todos os elementos do bloco a uma matriz chamada de "matriz de quantização". O efeito da quantização é basicamente o da divisão dos elementos do bloco pelos elementos da matriz de quantização. Já o efeito da quantização inversa é basicamente o efeito da multiplicação de todos os valores da matriz QF pelos valores de mesmo índice da matriz de quantização. No MPEG-2 existem duas matrizes padrão de quantização, uma para blocos intra e outra para blocos não intra. É na etapa

de quantização que ocorre a maior perda de informações, o que caracteriza o MPEG-2 como uma compressão com perdas (lossy compression). A quantização é viável pois a DCT (estágio anterior) garante que os elementos mais importantes do bloco estão ao redor da quina superior esquerda da matriz e os menos importantes, próximos a quina inferior direita. As figuras abaixo mostram as matrizes padrão de quantização, respectivamente, para blocos intra e blocos não intra.

$$\begin{pmatrix} 8 & 16 & 19 & 22 & 26 & 27 & 29 & 34 \\ 16 & 16 & 22 & 24 & 27 & 29 & 34 & 37 \\ 19 & 22 & 26 & 27 & 29 & 34 & 34 & 38 \\ 22 & 22 & 26 & 27 & 29 & 34 & 37 & 40 \\ 22 & 26 & 27 & 29 & 32 & 35 & 40 & 48 \\ 26 & 27 & 29 & 32 & 35 & 40 & 48 & 58 \\ 26 & 27 & 29 & 34 & 38 & 46 & 56 & 69 \\ 27 & 29 & 35 & 38 & 46 & 56 & 69 & 83 \end{pmatrix}$$

Figura 7: Matriz padrão de quantização intra.

$$\begin{pmatrix} 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 \\ 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 \end{pmatrix}$$

Figura 8: Matriz padrão de quantização não intra.

4. IDCT

   IDCT é a função inversa da DCT (transformada discreta de cosseno). Enquanto a DCT tem por objetivo transpor os valores de um bloco do domínio do espaço, para o domínio da freqüência, a IDCT tem o efeito contrário, reconstruindo os elementos de luminância ou crominância que o bloco contém. A entrada da IDCT é a matriz F[v][u] e a saída é a matriz f[y][x]. "x" e "y" são usados justamente para indicar que o bloco agora está no domínio do espaço.

   Mais informações sobre a IDCT podem ser vistas em 2.2.2.

5. Compensação de movimento

   Na compensação de movimento os vetores de movimento são usados para gerarem os chamados preditores de movimento, denotados por p[y][x]. Estes preditores são combinados com as saídas de IDCT, f[y][x], para gerarem os blocos decodificados d[y][x].

   Os vetores de movimento são gerados na codificação do vídeo, em uma etapa chamada de estimativa de movimento. Estes vetores correspondem basicamente ao deslocamento calculado a partir de macroblocos de imagens diferentes para os quais um casamento é encontrado.

   Mais informações sobre a compensação de movimento podem ser vistas em 2.2.3.

2.2.2 IDCT

De acordo com (SALOMON, 2004, sec. 4.6.4) a função que é conhecida como "A DCT", trata-se da Transformada Discreta de Cosseno tipo II (DCT-II), referida neste texto simplesmente por DCT e também por FDCT - Transformada discreta de cosseno "para Frente". A inversa da DCT-II é a DCT tipo III (DCT-III). A DCT-III é mais conhecida como Transformada Inversa Discreta de Cosseno (IDCT). Abaixo são apresentadas, respectivamente, as equações da DCT e IDCT para uma dimensão.

DCT unidimencional para N elementos:

$$F[u] = \sqrt{\frac{2}{N}} C(u) \sum_{x=0}^{N-1} f[x] \cos \frac{(2x+1)u\pi}{2N} \tag{2.3}$$

IDCT unidimencional para N elementos:

$$f[x] = \sqrt{\frac{2}{N}} \sum_{u=0}^{N-1} C(u) F[u] \cos \frac{(2x+1)u\pi}{2N} \tag{2.4}$$

Em ambas as equações temos que:

u e *x* = 0, 1, 2, ... *N - 1*

Onde:

x representa a coordenada de um valor no domínio do espaço.

u representa a coordenada de um valor no domínio da freqüência.

f e *F* são vetores de tamanho *N* .

C(u) é a seguinte função:

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & : \quad u = 0 \\ 1 & : \quad u > 0 \end{cases} \tag{2.5}$$

O MPEG-2 aplica DCT e IDCT na unidade bloco do bitstream. Um bloco MPEG-2 é bidimensional, possuindo 8x8 elementos. Por esta razão o MPEG-2 deve, em princípio, utilizar versões bidimensionais das equações 2.3 e 2.4. Entretanto na prática é possível utilizar as versões unidimensionais das equações, obtendo exatamente o mesmo resultado que se conseguiria usando as versões bidimensionais de DCT e IDCT. Isto pode ser obtido da seguinte forma: aplica-se a (I)DCT de uma dimensão em cada linha do bloco e, em seguida, a (I)DCT de uma dimensão em cada coluna do resultado. Devido a esta propriedade a DCT e a IDCT são ditas separáveis em duas dimensões.

O objetivo da DCT é transpor os valores de um bloco do domínio do espaço, para o domínio da freqüência, gerando uma matriz com a seguinte estrutura:

A quina superior esquerda (0,0) da matriz contém um valor que indica a quantidade da freqüência fundamental do bloco. Indo-se em direção à quina inferior direita, tanto pelo eixo horizontal qual pelo vertical, as posições da matriz contém valores das quantidades referentes as harmônicas superiores à fundamental. Quanto maior é a freqüência que uma posição representa, menor em magnitude será o valor que está naquela posição.

O valor da quina superior esquerda representa uma freqüência contínua (DC), todas as demais posições da matriz representam freqüências alternadas (AC).

Já a IDCT tem o objetivo oposto da DCT, ou seja, transpor os valores de um bloco do domínio da freqüência, para o domínio do espaço, reconstruindo os elementos de luminância ou crominância que o bloco contém. Para se restaurar um pixel, na hora da decodificação, toda a área do bloco a ser reconstruído é preenchida com o valor contínuo da matriz resultante da DCT e então os valores AC's são adicionados ao DC para perturbar os valores do pixel.

2.2.3 COMPENSAÇÃO DE MOVIMENTO

Estimativa e compensação de movimento são técnicas que exploram a redundância temporal existente em um vídeo (i.e. a semelhança entre imagens do vídeo obtidas em instantes de tempo próximos). A estimativa do movimento é realizada na codificação do vídeo, e a compensação de movimento (o inverso da estimativa) é usado na decodificação do vídeo.

Na estimativa de movimento as imagens são primeiramente divididas em blocos de tamanho fixo, o que no caso do MPEG-2 corresponde aos macroblocos. Em seguida um casamento para cada macrobloco é procurado em uma ou mais imagens chamadas de imagens de referência, já armazenadas. O deslocamento entre o macrobloco para o qual está se procurando casamento (que é o macrobloco para o qual está se fazendo uma predição) e o(s) macrobloco(s) da(s) imagem (imagens) de referência é chamado vetor de movimento. Para auxiliar os vetores de movimento, uma diferença que é calculada em função de subtrações pixel a pixel entre o macrobloco para o qual está se fazendo uma predição e o(s) macrobloco(s) de referência é codificada. Esta diferença calculada é também conhecida como resíduo de movimento.

Na compensação de movimento os vetores de movimento juntamente com o resíduo de movimento, são utilizados na reconstrução dos macroblocos e conseqüentemente da imagem original.

No MPEG-2 a utilização dos vetores de movimento para a reconstrução da imagem original é um processo que ocorre em várias etapas. São elas:

1. Seleção de quais serão as imagens de referência
2. Decodificação dos vetores de movimento
3. Formação das predições usando as imagens de referência e os vetores de movimento
4. Combinação das predições e geração dos preditores
5. Soma de preditor com saída da IDCT e saturação

SELEÇÃO DE QUAIS SERÃO AS IMAGENS DE REFERÊNCIA

Imagens de referência são imagens já decodificadas, a partir das quais (juntamente com a utilização dos vetores de movimento) serão reconstruídas imagens não intra, ou seja, imagens P e B. A seleção de quais serão as imagens de referência usadas na reconstrução de uma imagem não intra, depende do modo de predição que está sendo empregado. No MPEG-2 existem quatro modos de predição, dois considerados modos principais e dois modos chamados de especiais.

Os dois modos principais são predição de campo (field prediction) e predição de quadro (frame prediction). Os dois modos especiais são compensação de movimento 16x8 (16x8 motion compensation) e dual-prime. Nos próximos parágrafos, para efeito de ilustração, será mostrado como é feita a seleção de imagens de referência na predição de quadro. O leitor que desejar se aprofundar no processo de seleção de imagens de referência para os outros modos de predição pode consultar (ISO/IEC, 1995).

Na predição de quadro para reconstrução de imagens P, a imagem de referência selecionada será o frame mais recentemente decodificado. Este frame ocorre antes da imagem P. Isto é ilustrado na figura 9.

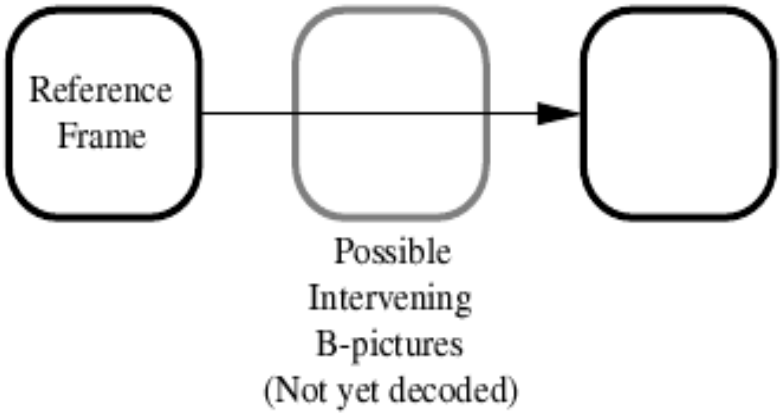

Figura 9: Predição de quadro em imagens P (ISO/IEC, 1995).

Na predição de quadro para reconstrução de imagens B, existirão duas imagens de referência, correspondendo aos dois frames mais recentemente decodificados, sendo que um dos frames ocorre antes da imagem B e outro depois. Isto é ilustrado na figura 10.

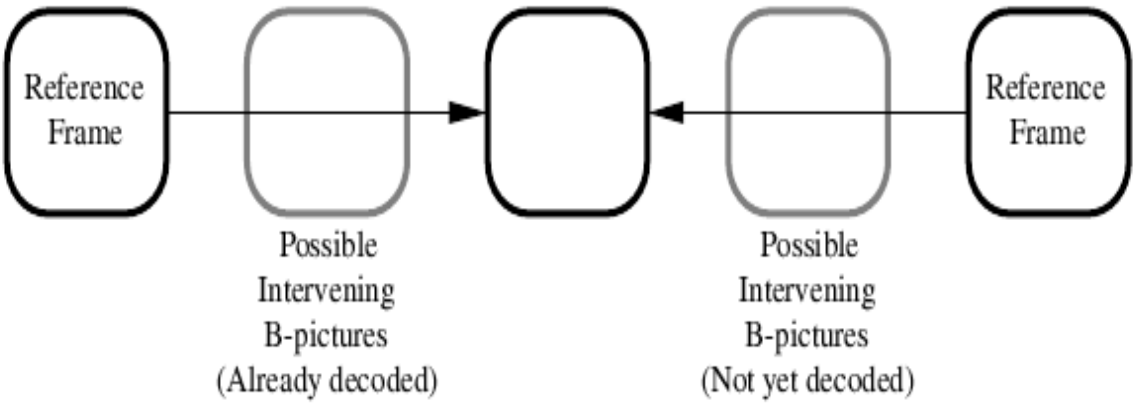

Figura 10: Predição de quadro em imagens B (ISO/IEC, 1995).

DECODIFICAÇÃO DOS VETORES DE MOVIMENTO

Vetores de movimento (Motion vectors - MVs), como mencionado anteriormente, representam o deslocamento entre macroblocos de imagens diferentes, para o qual um casamento é encontrado. Estes vetores são gerados na fase de estimativa de movimento, no processo de codificação de um vídeo. No decodificação, para que os vetores de movimento possam ser utilizados na reconstrução de imagens, eles precisam ser primeiramente decodificados.

Vetores de movimento podem ser codificados integralmente (full format), ou de forma diferencial (Differential Motion Vector - DMV). Na codificação diferencial, apenas as diferenças de acordo com os outros vetores de movimento previamente decodificados são armazenadas, o que reduz o número de bits necessários para representar-los.

Um vetor de movimento codificado pode ser denotado como motion_vector[r][s][t]. O índice "r" pode indicar: se é o primeiro ou o segundo vetor de movimento do macrobloco, ou ainda se é um vetor full ou diferencial usado no tipo de predição dual-prime. O índice "s" pode indicar: se o vetor é do tipo forward ou backward de predição. O índice "t" pode indicar: se o valor em questão corresponde a componente vertical ou horizontal do vetor. Um motion_vector[r][s][t] é proveniente do bitstream, e consiste basicamente de três elementos:

- motion_code[r][s][t]

  Este elemento, que é codificado usando-se VLC, indica qual a magnitude da componente

vertical ou horizontal do vetor, podendo esta assumir qualquer valor inteiro no intervalo de [-16, 16]. 16 é relativo ao tamanho da largura e altura da componente de luminância de um macrobloco. É interessante notar que a decodificação de um MV se dá primeiro para a componente de luminância do macrobloco, e somente depois e em função desta, são decodificados os componentes de crominância.

- motion_residual[r][s][t]
  Este elemento é um número inteiro que representa o resíduo de movimento existente entre o macrobloco do qual a predição foi feita e os correspondentes macroblocos de referência; para a componente vertical ou horizontal do vetor.

- dmvector[t]
  Este elemento, que é codificado usando-se VLC, é usado nos vetores codificados de forma diferencial. Ele pode assumir os valores -1, 0, ou 1 para uma dada componente vertical ou horizontal do vetor.

A grande maioria dos MVs é codificada de forma diferencial. O processo geral de codificação funciona basicamente da seguinte maneira.

1. Obtem-se o motion_vector[r][s][t] do bitstream e a partir deste um delta que corresponde ao valor do vetor diferencial.

2. Somando este delta com um preditor de vetor de movimento, obtém-se o um vetor de movimento denotado como vector'[r][s][t]. Preditores de vetores de movimento, representam valores bases de uma componente horizontal ou vertical de um vetor, a partir dos quais, um vetor de movimento pode ser calculado. Estes preditores são denotados como PMV[r][s][t].

3. Colocando vector'[r][s][t] em escala, dependendo do formato do macrobloco (4:2:0, 4:2:2 or 4:4:4), gera-se vetor[r][s][t], que o vetor de movimento decodificado.

Em princípio não deveriam existir vetores de movimento associados a macroblocos intra, já que eles contém em si toda a informação necessária para serem decodificados. E de fato não existem vetores de movimento "regulares" para tais macroblocos. Entretanto existe um tipo de vetor de movimento especial chamado de vetor de cancelamento de movimento (concealment motion vector - CMV) associado a macroblocos intra. O objetivo deste tipo de vetor é auxiliar na recuperação de um macrobloco intra perdido (o que pode ocorrer na transmissão do bitstream). CMVs podem ser enviados junto dos macroblocos intra, ou em outras partes do bitstream.

No primeiro caso, quando um macrobloco intra é perdido o CMV dele se perde junto e algo semelhante ao macrobloco perdido pode ser recuperado usando-se os CMVs dos macroblocos anteriores e posteriores a ele e um macrobloco de referência. No segundo caso, é possível reconstruir o macrobloco perdido a partir de seu próprio CMV e de um macrobloco de referência que pode ser o macrobloco anterior ao perdido.

FORMAÇÃO DAS PREDIÇÕES USANDO AS IMAGENS DE REFERÊNCIA E OS VETORES DE MOVIMENTO

Com os vetores decodificados e as imagens de referência selecionadas é possível iniciar a construção das chamadas predições. Predições são formadas a partir de predições de amostras. Uma predição de amostra é feita lendo se uma amostra de uma imagem de referência e deslocando-se esta amostra utilizando o vetor de movimento. Um valor positivo na componente horizontal do vetor de movimento, indica que a predição é feita a partir de amostras (da imagem de referência) que se encontram a direita das amostras cujas predições estão sendo calculadas. Um valor positivo na componente vertical do vetor de movimento, indica que a predição é feita a partir de amostas que se encontram abaixo das amostras cujas predições estão sendo calculadas.

Todos os vetores de movimento possuem uma precisão de meia amostra (half-sample) ou, em outras palavras, de meio pixel. Entretanto a precisão das imagens de referência está em pixel. Então se um componente do vetor de movimento (seja ele horizontal ou vertical) for ímpar, as amostras serão lidas de um ”meio-caminho”entre as atuais amostras da imagem de referência. Estas meias-amostras serão calculadas usando interpolação linear simples a partir das duas amostras de referências atuais.

COMBINAÇÃO DAS PREDIÇÕES E GERAÇÃO DOS PREDITORES

As várias predições calculadas devem ser combinadas para gerar um bloco final de predição, conhecido como preditor. Este bloco tem dimensão 8x8, que é mesma dimensão dos blocos que saem da IDCT. Isto é proposital, visto que na no final da compensação de movimento estes blocos serão somados. Em geral um preditor é gerado pela combinação de quatro predições.

SOMA DE PREDITOR COM SAÍDA DA IDCT E SATURAÇÃO

Um bloco decodificado d[y][x] é gerado somando-se elemento por elemento de um bloco preditor p[y][x] com um bloco de dados de coeficiente (saída da IDCT) f[y][x] e pela saturação de cada elemento resultante da soma. A saturação é a etapa final da compensação de movimento,

ela tem por objetivo objetivo remover os valores negativos e os valores muito altos (maiores que 255), da soma de p[y][x] com f[y][x]. Para isto os os valores acima de 255 são saturados em 255 e os valores negativos em 0.

3 PROJETO

Este capítulo descreve o projeto desenvolvido. Primeiramente são apresentados a API DSP e decodificador MPEG-2 utilizados. Em seguida são explicadas as modificações que foram realizadas na IDCT do decodificador. Por fim, é apresentado como a implementação da compensação de movimento está estruturada no decodificador escolhido e são apontados os pontos do mesmo que podem fazer uso da API.

3.1 API DSP

Neste projeto foi utilizada uma API de processamento digital de sinais e funções de suporte a compressão de áudio e vídeo. Esta é o trabalho do mestrando Danillo Moura Santos, co-orientador deste projeto. A principal idéia desta API é de prover funcionalidades de DSP e correlatas de forma transparente para seus clientes (e.g. codificadores e decodificadores), possuindo implementações otimizadas para para diversas arquiteturas heterogêneas, como por exemplo processadores digitais de sinais com instruções dedicadas, processadores de propósito geral, sistemas dedicados desenvolvidos em FPGA. Vale lembrar que a API está em desenvolvimento, portanto as funções providas ainda não se encontram nas suas versões definitivas.

3.2 LIBMPEG2

Utilizou-se neste projeto a biblioteca libmpeg2 (LIBMPEG2..., 2008). Esta implementa um decodificador que possui conformidade com o padrão MPEG-2 (“main profile”) e trata-se de um software livre, ou seja, com código fonte disponível e passível de ser alterado. A libmpeg2 é amplamente utilizada e depurada. Em (LIBMPEG2..., 2008) existe uma lista de projetos que usam a libmpeg2; dentre eles podemos citar os seguintes players multimídia: VLC (VideoLAN), MPlayer e Xine.

A libmpeg2 é escrita na linguagem C. Entretanto, alguns trechos da biblioteca possuem versões em assembly (além da versão em C), trechos estes associados a estágios do processo de

decodificação que podem aproveitar-se de otimizações específicas de determinadas plataformas de hardware. Os estágios da libmpeg2 que possuem otimizações em assembly são a IDCT e a compensação de movimento (Motion Compensation - MC). As plataformas para as quais atualmente existem otimizações em assembly são: x86 (instruções MMX) e Power PC - ppc (instruções AltiVec).

3.3 IDCT

Nesta seção são descritas as modificações realizadas no decodificador no que diz respeito a IDCT. Primeiramente é apresentada a função da API utilizada, em seguida quais as modificações foram realizadas no decodificador e onde elas ocorreram. Para assegurar a corretude das modificações foram realizados testes, cujas considerações para a elaboração e resultados são aqui descritos. Além da corretude, outro aspecto verificado foi o desempenho, onde a implementação de IDCT provida pela API foi comparada com a original. A descrição de como foi feita esta análise assim como seus respectivos resultados também são apresentados nesta seção.

3.3.1 FUNÇÃO DA API UTILIZADA

Na modificação do decodificador no estágio de IDCT, onde a implementação de IDCT original da libmpeg2 foi trocada pela implementação fornecida pela API DSP, utilizou-se a seguinte função da API:
*void idct2D(DCTELEM *dctWeights);*
onde *DCTELEM* é um *short* (inteiro de 16 bits sinalizados) e *dctWeights* é o bloco de entrada (e saída) da IDCT. Esta função, opera uma IDCT bidimensional sobre um bloco de 8x8 elementos.

3.3.2 MODIFICAÇÕES REALIZADAS

Na libmpeg2, existem dois ponteiros de função que são associados a funções relacionadas com a IDCT. São eles: mpeg2_idct_copy e mpeg2_idct_add, utilizados, respectivamente, na decodificação intra e inter de quadros/campos MPEG-2. Em tempo de compilação são disponibilizadas as funções com implementação em assembly para plataformas específicas (caso existam para a plataforma em questão) e as funções escritas em C (sempre disponíveis). As funções C que podem ser associadas aos ponteiros mpeg2_idct_copy e mpeg2_idct_add são, mpeg2_idct_copy_c e mpeg2_idct_add_c (nesta ordem). Os nomes das funções que são implementadas em assembly dependem da plataforma: mpeg2_idct_copy_altivec, por exemplo, é a

função para a plataforma Power PC (com instruções AltiVec) que pode ser associada ao ponteiro mpeg2_idct_copy. Em tempo de execução, se houverem funções otimizadas, é possível escolher entre elas e as versões em C.

No presente trabalho as funções mpeg2_idct_copy_c e mpeg2_idct_add_c foram modificadas para utilizarem a função de IDCT provida pela API. Não houve preocupação em alterar as funções com implementação em assembly, pois as implementações da IDCT da própria API é que poderão prover versões em assembly, otimizadas para plataformas específicas. Aqui, é demonstrado como a função mpeg2_idct_copy_c foi alterada. A função mpeg2_idct_add_c foi alterada de forma muito semelhante, por isto não é mostrada aqui. Mas ambas as alterações podem ser observadas na íntegra no anexo A.

A função mpeg2_idct_copy_c possui a seguinte assinatura:
*void mpeg2_idct_copy_c (int16_t * block, uint8_t * dest, const int stride);* onde: *block* é o bloco de entrada da IDCT, *dest* é o bloco de saída *stride* indica o quanto se avança no bloco de saída, a medida em que ele está sendo reconstruído. A implementação original é basicamente a seguinte:

```
static void mpeg2_idct_copy_c (int16_t * block, uint8_t * dest,
const int stride)
{
int i;

/* Realização da IDCT */
for (i = 0; i < 8; i++)
idct_row (block + 8 * i);
for (i = 0; i < 8; i++)
idct_col (block + i);

// ...

}
```

Onde é aplicado uma IDCT unidimensional nas linhas do bloco e, em seguida, outra IDCT de uma dimensão em cada coluna do bloco alterado. Esta função pode ser observada na íntegra nos fontes da libmpeg2 (versão mpeg2dec-0.4.1), que podem ser obtidos em (LIBMPEG2..., 2008).

A versão alterada de mpeg2_idct_copy_c, utilizando-se a API, fica da seguinte forma:

```
static void mpeg2_idct_copy_c (int16_t * block, uint8_t * dest,
const int stride)
{
int i = 8;
libmpeg2ToNormal(block);

/* Realização da IDCT */
c_idct2D(block);

// ...

}
```

Como a função *idct2D* , provida pela API é escrita em C++, e a libmpeg2 em C, é necessário a utilização da função wrapper *c_idct2D* . A implementação de *c_idct2D* é trivial, e pode ser observada no apêndice A.

Utilizando-se a IDCT da libmpeg2 de forma isolada percebeu-se que, para um mesmo conjunto de entradas, a saída da IDCT da libmpeg2 era diferente de uma IDCT convencional. Verificou-se então que a IDCT da libmpeg2 exige que sua entrada esteja em um formato específico para operar corretamente e gerar uma saída igual a de uma IDCT convencional. Esta formatação acontece quando os blocos que entrarão na IDCT ainda estão sendo reconstruídos (estágios de Decodificação de comprimento variável, Esquadrinhamento inverso e Quantização inversa). A formatação consiste em uma reordenação dos elementos do bloco e na **multiplicação** de cada elemento por 16. A matriz abaixo mostra a ordem dos índices dos elementos em um bloco libmpeg2 reordenado.

$$\begin{pmatrix} 0 & 4 & 1 & 5 & 2 & 6 & 3 & 7 \\ 32 & 36 & 33 & 37 & 34 & 38 & 35 & 39 \\ 8 & 12 & 9 & 13 & 10 & 14 & 11 & 15 \\ 40 & 44 & 41 & 45 & 42 & 46 & 43 & 47 \\ 16 & 20 & 17 & 21 & 18 & 22 & 19 & 23 \\ 48 & 52 & 49 & 53 & 50 & 54 & 51 & 55 \\ 24 & 28 & 25 & 29 & 26 & 30 & 27 & 31 \\ 56 & 60 & 57 & 61 & 58 & 62 & 59 & 63 \end{pmatrix}$$

Figura 11: Índices em um bloco libmpeg2.

Considerando que a IDCT provida pela API é uma IDCT convencional, exigindo blocos "não formatados" como entrada, duas opções existiam: a primeira consistiria em alterar a reconstrução dos blocos na libmpeg2 nos estágios anteriores à IDCT para que não efetuassem nenhuma formatação, a segunda seria a de reverter tal formatação. Optou-se pela segunda opção, pois apesar de ela gerar um pequeno overhead (já que todos os blocos na antes de entrarem na IDCT são "reformatados"), ela não é uma modificação tão intrusiva já que não altera as diversas funções da libmpeg2 responsáveis pela montagem do bloco, e ela é mais localizada, ocorrendo apenas na entrada da IDCT e em mais nenhum outro lugar. A implementação de reformatação foi relizada pela função *libmpeg2ToNormal* , que pode ser observada no apêndice B. Esta função consiste basicamente na "reordenação inversa" dos elementos do bloco de acordo com a matriz da figura 11 e na **divisão** de cada elemento por 16.

Como mencionado anteriormente, as implementações de *void idct2D(DCTELEM *dctWeights);* é que poderão prover versões em assembly, otimizadas para plataformas específicas. Portanto tais otimizações específicas a cada plataforma, e a própria portabilidade (neste sentido), passam a residir não mais nas várias versões das funções associadas a mpeg2_idct_copy e mpeg2_idct_add e sim na função provida pela API. Desta forma a granularidade das otimizações para plataformas se torna mais fina e focada na API, assim o desenvolvedor do decodificador passa a se preocupar apenas na interface das funções DSP que utilizará e não nas implementações das mesmas.

3.3.3 CONSIDERAÇÕES PARA ELABORAÇÃO DOS TESTES

O padrão MPEG-2 parte 4 (testes de conformidade)(ISO/IEC, 1998) estabelece dois critérios que uma IDCT deve seguir para que ela seja considerada de acordo com o mesmo. O padrão também descreve como elaborar uma IDCT de referência e uma série de conjuntos de dados que a IDCT de referência e a IDCT a ser avaliada devem processar para que possa ser julgada a conformidade.

Os critérios estabelecidos pelo padrão MPEG-2 para avaliação da IDCT são os seguintes:

- De modo geral o erro de pico, definido como o módulo da diferença entre cada valor esperado e obtido, deve ser menor ou igual a 2. Um valor é um número obtido na saída da IDCT depois de formatado (clip) para ficar no intervalo [-256, +255].
- Considerando F como o conjunto de 4096 blocos Bi[y][x] (i = 0 .. 4095), definidos da seguinte maneira:
    1. Bi[0][0] = i - 2048

2. Bi[7][7] = 1 se Bi[0][0] é par, senão (Bi[0][0] impar) Bi[7][7] = 0.
3. Todos os outros coeficientes Bi[y][x] que não sejam Bi[0][0] ou Bi[7][7] valem 0.

Para cada bloco Bi[y][x] de entrada da IDCT pertencente ao conjunto F definido acima, o erro de pico deve ser menor ou igual a 1.

3.3.4 TESTES E AVALIAÇÃO DOS RESULTADOS

Construi-se uma IDCT e uma FDCT de referência e o teste de conformidade para IDCT de acordo com o padrão MPEG2 ISO/IEC 13818-4 (ISO/IEC, 1998) e IEEE Std 1180 (IEEE, 1990-1991). A implementação das funções de referência e do teste de conformidade pode ser vista no apêndice C. Aplicou-se este teste na IDCT original da libmpeg2 para confirmar se esta era realmente compatível com o padrão e como uma forma de testar o teste desenvolvido. Para testar a IDCT da libmpeg2 foi necessário utilizar a função *normalToLibmpeg2* , que faz o inverso da função *libmpeg2ToNormal* , ou seja, altera um bloco "comum" para o formato esperado pela IDCT da libmpeg2. *normalToLibmpeg2* pode ser vista no apêndice B. O teste da IDCT da libmpeg2 pode ser visto no apêndice D. Como era de se esperar a IDCT original da libmpeg2 passou no teste. Testou-se então a implementação padrão da IDCT provida pela API, e esta também demonstrou-se de acordo com os requisitos do padrão. Este teste pode ser visto no apêndice E.

Como teste de integração da IDCT na libmpeg2 executou-se a libmpeg2 adaptada e a original, sobre os mesmos vídeos, imprimindo as saídas das IDCTs e compararando os resultados. Neste caso considerou-se a saída da libmpeg2 original como a referência. Também neste caso, como era de se esperar, as diferenças entre os resultados se encaixam nos requisitos do padrão MPEG-2. Para executar a libmpeg2, utilizou-se de um programa exemplo (sample1) que vem com a mesma. Este programa tem como entrada um arquivo de vídeo em MPEG-2, e sua saída são os arquivos de imagem não compactada, que formam os vários quadros do vídeo decodificado. O código fonte do programa sample1, pode ser visto no anexo A. Ele aparece com a "aceleração" desligada justamente para fazer uso de *mpeg2_idct_copy_c* e *mpeg2_idct_add_c* .

3.3.5 ANÁLISE DE DESEMPENHO

A implementação de IDCT provida pela API foi comparada com a IDCT original da libmpeg2 com relação a tempo médio de computação de um determinado número de blocos. Estes blocos foram gerados pelo mesmo mecanismo empregado no teste de corretude da IDCT (descrito nas subseções anteriores) e especificado pelo MPEG-2 parte 4.

O procedimento para análise de desempenho, cujo código fonte pode ser visto no apêndice F, foi basicamente o seguinte:

1. Gerar um bloco IDCT
2. Aplicar este bloco na IDCT da libmpeg2 M vezes e computar o tempo médio de execução deste bloco
3. Aplicar o mesmo bloco na implementação de IDCT provida pela API, M vezes e computar o tempo médio de execução deste bloco
4. Repetir este processo para N blocos e no fim computar o tempo médio geral de cada IDCT utilizando o tempo médio de cada bloco.

As funções utilizadas para se obter o tempo tinham precisão de microsegundos, mas utilizando um M = 1000, conseguiu-se calcular um tempo médio de execução para cada bloco com precisão de nanosegundos.

Em uma execução com N = 60000 (sessenta mil) blocos distintos, o tempo médio de execução das IDCTs foi o seguinte:

- IDCT libmpeg2: 869 nanosegundos (ns)
- IDCT API: 1326 nanosegundos (ns)

A diferença entre os tempos de execução é de 457 nanosegundos. A relação tempo médio execução IDCT API / tempo médio execução IDCT libmpeg2, ou seja, 1326 ns / 869 ns, é de aproximadamente 1.5. Isto significa que a IDCT da libmpeg2 se apresentou cerca de 1.5 vezes mais rápida do que a implementação de IDCT provida pela API. No procedimento descrito acima, desejou-se verificar o desempenho das IDCTs (libmpeg2 e API) de forma isolada. Portanto o tempo de mapeamento de blocos de entrada de IDCT formato libmpeg2 para blocos IDCT convencionais, realizado por libmpeg2ToNormal, não foi considerado. Repetindo-se este procedimento, considerando o tempo de mapeamento, a diferença entre os tempos de execução sobe para cerca de 2.5 vezes.

Atualmente nem o decodificador original nem o modificado se encontram integrados a um player de vídeo. Esta integração seria interessante para se avaliar de forma visual qual a representatividade desta diferença de 2.5 vezes entre o tempo de computação das duas IDCTs (considerando também o tempo de mapeamento entrada IDCT libmpeg2 $\rightarrow$ entrada IDCT convencional). Entretanto realizou-se um pequeno experimento que forneceu uma indicação aproximada da representatividade desta diferença. O experimento foi o seguinte: executou-se alguns

vídeos MPEG-2 no player VLC (VLC..., 2008) o qual utiliza a libmpeg2. Mediu-se a carga da CPU durante a execução do VLC utilizando-se o programa TOP (TOP..., 2008). Em média a carga da CPU fica abaixo de 5%, o que representa de forma aproximada a carga utilizando-se o decodificador original. Considerando uma relação diretamente proporcional e linear entre o tempo de execução da IDCT e a carga média da CPU, a utilização do decodificador modificado no VLC resultaria em uma carga de 12.5% (2.5 x 5%). Como 12.5% é uma carga relativamente baixa de utilização de CPU, pode-se inferir que a diferença de 2.5 vezes entre os tempos médios de execução das IDCTs é aceitável.

Convém lembrar que a IDCT da libmpeg2 (mesmo em C) é otimizada, por isto mais rápida que a implementação provida pela API. Em contrapartida não é uma IDCT convencional já que exige entradas alteradas. Por outro lado a IDCT da API é genérica e pode ser usada em vários decodificadores. Além disto foi verificada apenas uma das implementações que a API proverá, o que significa que esta análise de desempenho não é abrangente, pois analisou apenas duas implementações específicas.

3.4 COMPENSAÇÃO DE MOVIMENTO

A compensação de movimento (Motion Compensation - MC), é um importante estágio do processo de decodificação MPEG-2, pois é neste momento em que a redundância temporal é tratada, e as imagens codificadas em função de outras, reconstruídas. Esta importância se reflete na libmpeg2, pois além de uma implementação genérica escrita em C, existem também implementações escritas em assembly, otimizadas para plataformas específicas.

Nesta seção é primeiramente apresentado como o estágio de compensação de movimento está organizado na libmpeg2. Em seguida, indica-se quais pontos da MC podem fazer uso da API DSP.

3.4.1 COMPENSAÇÃO DE MOVIMENTO NA LIBMPEG2

Segundo afirma Michel Lespinasse em mensagem para lista de discussão da libmpeg2 (LIBMPEG2-DEVEL..., 2008) em 13/08/2003, o que foi confirmado inspecionando-se o código fonte da libmpeg2, a compensação de movimento está organizada da seguinte maneira:

No nível mais alto existe a macro MOTION_CALL, utilizada no processamento dos macroblocos que fazem parte de um determinado slice. Está macro executará a rotina de MC desejada de acordo com a estrutura da imagem e os modos de macrobloco.

As funções que MOTION_CALL poderá chamar são específicas para frames ou fields, para o modo de predição a ser utilizado, e para cada tipo de macrobloco. Seguem alguns exemplos:

- *motion_fr_field_420* utilizada em frames que utilizam predição do modo field, em macroblocos do tipo 4:2:0

- *motion_fr_frame_422* utilizada em frames que utilizam predição do modo frame, em macroblocos do tipo 4:2:2

- *motion_fi_16x8_444* utilizada em frames que utilizam predição do modo 16x8 MC, em macroblocos do tipo 4:4:4

As funções do tipo *motion_fr_** e *motion_fi_** , em suas implementações utilizam macros que são expandidas em operações relacionadas ao processamento dos vetores de movimento. Estas macros possuem nomes como MOTION_420, MOTION_FIELD_444, MOTION_DMV_422, ou seja, estão relacionadas ao modo de predição utilizado (field, frame, dual-prime, 16x8 MC) e ao tipo do macrobloco (4:2:0, 4:2:2, 4:4:4).

Existem funções que atuam na etapa de formação das predições (2.2.3) da MC, operando sobre blocos de referência e vetores de movimento já decodificados. Estas funções são implementadas basicamente com o uso de laços e operações de deslocamento (shifts). Tais funções possuem implementações genéricas escritas em C e versões escritas em assembly, otimizadas para plataformas específicas. Alguns exemplos destas funções são:

- *MC_put_x_16_c*
  Realiza operação de atribuição (put), atuando na componente horizontal do macrobloco (x) e sobre toda sua extensão (16 pixels). A terminação "_c"indica que trata-se da versão da função escrita em C.

- *MC_put_x_16_mmx*
  Versão assembly utilizando instruções MMX da função acima.

- *MC_avg_x_16_c*
  Semelhante as anteriores, porém este realiza uma média (average - avg), utilizada no calculo das meias-amostras.

3.4.2 PONTOS EM QUE A API PODE SER UTILIZADA

A API DSP deverá prover além de funcionalidades de processamento digital de sinais (DSP) como IDCT, FFT, DWT, entre outras; primitivas de processadores digitais de sinais (DSP-HW) como acumuladores de multiplicação (MAC), multiplicações de matrizes, laços em hardware, e deslocadores em barra (barrel shifters).

Como já foi afirmado em 3.4.1, as funções da libmpeg2 que lidam com a formação das predições fazem uso de laços e operações de deslocamento em sua implementações. A implementação da função *MC_put_x_16_c* ilustra esta afirmação:

```
// From motion_comp.c with macros extended
static void MC_avg_xy_16_c (uint8_t * dest, const uint8_t * ref,
                                const int stride, int height)
{
    do
    {
        dest[0] = (((((ref[0]+ref[0 +1]+(ref+stride)[0]+
                       (ref+stride)[0 +1]+2)>>2))+dest[0]+1)>>1);
        dest[1] = (((((ref[1]+ref[1 +1]+(ref+stride)[1]+
                       (ref+stride)[1 +1]+2)>>2))+dest[1]+1)>>1);
        dest[2] = (((((ref[2]+ref[2 +1]+(ref+stride)[2]+
                       (ref+stride)[2 +1]+2)>>2))+dest[2]+1)>>1);
        dest[3] = (((((ref[3]+ref[3 +1]+(ref+stride)[3]+
                       (ref+stride)[3 +1]+2)>>2))+dest[3]+1)>>1);
        dest[4] = (((((ref[4]+ref[4 +1]+(ref+stride)[4]+
                       (ref+stride)[4 +1]+2)>>2))+dest[4]+1)>>1);
        dest[5] = (((((ref[5]+ref[5 +1]+(ref+stride)[5]+
                       (ref+stride)[5 +1]+2)>>2))+dest[5]+1)>>1);
        dest[6] = (((((ref[6]+ref[6 +1]+(ref+stride)[6]+
                       (ref+stride)[6 +1]+2)>>2))+dest[6]+1)>>1);
        dest[7] = (((((ref[7]+ref[7 +1]+(ref+stride)[7]+
                       (ref+stride)[7 +1]+2)>>2))+dest[7]+1)>>1);
        dest[8] = (((((ref[8]+ref[8 +1]+(ref+stride)[8]+
                       (ref+stride)[8 +1]+2)>>2))+dest[8]+1)>>1);
        dest[9] = (((((ref[9]+ref[9 +1]+(ref+stride)[9]+
                       (ref+stride)[9 +1]+2)>>2))+dest[9]+1)>>1);
        dest[10] = (((((ref[10]+ref[10 +1]+(ref+stride)[10]+
                        (ref+stride)[10 +1]+2)>>2))+dest[10]+1)>>1);
        dest[11] = (((((ref[11]+ref[11 +1]+(ref+stride)[11]+
                        (ref+stride)[11 +1]+2)>>2))+dest[11]+1)>>1);
        dest[12] = (((((ref[12]+ref[12 +1]+(ref+stride)[12]+
                       (ref+stride)[12 +1]+2)>>2))+dest[12]+1)>>1);
```

```
        dest[13] = (((((ref[13]+ref[13 +1]+(ref+stride)[13]+
                        (ref+stride)[13 +1]+2)>>2))+dest[13]+1)>>1);
        dest[14] = (((((ref[14]+ref[14 +1]+(ref+stride)[14]+
                        (ref+stride)[14 +1]+2)>>2))+dest[14]+1)>>1);
        dest[15] = (((((ref[15]+ref[15 +1]+(ref+stride)[15]+
                        (ref+stride)[15 +1]+2)>>2))+dest[15]+1)>>1);
        ref += stride;
        dest += stride;
    } while (--height);
}
```

Estas funções que lidam com a formação das predições representam pontos onde a utilização da API é possível. As primitivas da API a serem utilizadas, neste caso, seriam os deslocadores em barra e os laços em hardware. Os deslocadores em barra permitem a realização das operações de deslocamento em um único ciclo de relógio. E laços em hardware, poderiam substituir os laços em software presentes nestas funções.

No presente momento deslocadores em barra e os laços em hardware ainda não são providos pela API. Porém, quando eles estiverem disponíveis, a estratégia para utiliza-los a partir da libmpeg2 será a mesma que foi usada para substituir a IDCT, isto é, a alteração da versão em C das devidas funções. Assim como no caso da IDCT não haverá necessidade de alterar as versões assembly das funções, uma vez que as otimizações ficarão a cargo das implementações providas pela API.

4 CONCLUSÕES

É desejável que implementações em software de codecs sejam portáveis para as diversas plataformas de hardware existentes, mantendo um desempenho satisfatório nas diversas plataformas. O processamento de sinais de áudio e vídeo costuma consumir uma parcela significativa no tempo de processamento de tais codecs. Em virtude disto diversas implementações de codecs possuem otimizações para processamento de sinais escritas em linguagem assembly, focadas em plataformas específicas. Com a disponibilidade de uma API que concentre em si funcionalidades de DSP e DSP-HW, provendo implementações para diversas arquiteturas distintas, a construção ou adaptação de um codec para atender o requisito de ser multiplataforma se torna facilitada. Este projeto escolheu um decodificador de vídeo padrão MPEG-2 para ser adaptado para tal API. Esta API é o trabalho do mestrando Danillo Moura Santos, co-orientador deste projeto. Os estágios do decodificador trabalhados foram a transformada inversa discreta de coseno (IDCT) e a compensação de movimento (MC). O decodificador escolhido para ser adaptado foi a libmpeg2 , que possui código fonte disponível e passível de ser alterado e apresenta conformidade com o padrão MPEG-2.

Pesquisou-se na libmpeg2 onde a IDCT era utilizada e como era utilizada. A partir daí modificou-se a libmpeg2 para utilizar a IDCT provida pela API DSP. A compensação de movimento também foi analisada e gerou-se apontamentos de onde a utilização da API é possível.

Como a IDCT da libmpeg2 exige que sua entrada esteja em um formato específico, diferente do esperado para uma IDCT convencional e como a API provê uma IDCT que funciona com entradas convencionais, foi necessário realizar um mapeamento de “entradas formato libmpeg2” para “entradas convencionais”, para poder fazer uso da IDCT fornecida pela API dentro da libmpeg2. Aplicou-se o teste de conformidade descrito no padrão MPEG-2 parte 4 (ISO/IEC, 1998) na implementação original da IDCT da libmpeg2 e na implementação provida pela API. Ambas as IDCTs passaram no testes. Comparou-se as saídas de IDCT geradas na decodificação de vídeos do decodificador original e do decodificador modificado, analisando-as sob os requisitos do mesmo teste de conformidade aplicado nas IDCTs de forma isolada. Constatou-se que os resultados também estão de acordo com os requisitos do teste. Fez-se também uma análise de desempenho comparando a IDCT original com a provida pela API. Também neste quesito

os resultados foram satisfatórios.

Foram identificados os pontos da libmpeg2 relativos a compensação de movimento que podem usar a API DSP. Atualmente as primitivas da API a serem utilizadas pela MC ainda não estão disponíveis já que a mesma se encontra em desenvolvimento. Porém, quando elas estiverem disponíveis, a estratégia para utiliza-los a partir da libmpeg2 será a mesma que foi usada para substituir a IDCT.

Através deste trabalho pode ser demonstrada a aplicabilidade de uma API DSP multiplataforma em um problema real, que é a decodificação de vídeo digital. Para tanto um decodificador padrão MPEG-2 foi adaptado à API, se beneficiando de todas as características providas pela mesma. Um outro fruto deste trabalho foi a implementação do teste de conformidade MPEG-2 parte 4 no que diz respeito a teste de IDCT. Como parte deste teste foram implementadas também uma FDCT e uma IDCT de referência. Este teste pode ser aplicado a cada nova implementação de IDCT que se desejar verificar conformidade com o padrão MPEG-2.

REFERÊNCIAS

IEEE. *IEEE Standard Specifications for the Implementations of 8x8 Inverse Discrete Cosine Transform*. [S.l.], 1990–1991.

ISO/IEC. *International Standard ISO/IEC 13818-2 (Video), Recommendation ITU-T H.262*. [S.l.], 1995.

ISO/IEC. *International Standard ISO/IEC 13818-4 (Conformance testing)*. [S.l.], 1998.

LIBMPEG2 - a free MPEG-2 video stream decoder. 2008. Disponível em: <http://libmpeg2.sourceforge.net>. Acesso em: 10 mar. 2008.

LIBMPEG2-DEVEL - Mailing list archive. 2008. Disponível em: <http://sourceforge.net/mailarchive/forum.php?forum_name=libmpeg2-devel>. Acesso em: 04 jun. 2008.

MPEG Home Page. 2008. Disponível em: <http://www.chiariglione.org/mpeg/index.htm>. Acesso em: 12 mai. 2008.

SALOMON, D. *Data Compression - The Complete Reference*. [S.l.]: Springer, 2004.

TOP (Unix). 2008. Disponível em: <http://en.wikipedia.org/wiki/Top_(Unix)>. Acesso em: 04 jun. 2008.

VLC media player - Overview. 2008. Disponível em: <http://www.videolan.org/vlc/>. Acesso em: 04 jun. 2008.

WOOTTON, C. *A Practical Guide to Video and Audio Compression - From Sprockets and Rasters to Macro Blocks*. [S.l.]: Elsevier, 2005.

APÊNDICE A – CÓDIGO FONTE DA IMPLEMENTAÇÃO DA API UTILIZADA

Arquivo idct_api_dsp.h

```
#ifndef IDCT2D
#define IDCT2D

typedef short DCTELEM;

void idct2D(DCTELEM *dctWeights);

extern "C" {
void c_idct2D(DCTELEM *dctWeights);
}

#endif // IDCT2D
```

Arquivo idct_api_dsp.cc

```
/* Copyright (C) 2007, 2008, Mateus Krepsky Ludwich.
Contact: mateus@lisha.ufsc.br
*/

#include "idct_api_dsp.h"
#include "block.h"

void idct2D(DCTELEM *dctWeights) {
```

```
CoeffBlock::j_rev_dct (dctWeights);
}

void c_idct2D(DCTELEM *dctWeights) {
idct2D(dctWeights);
}
```

Arquivo block.h

```
/* block.h - Adapted from
http://www.uow.edu.au/~nabg/MPEG
(A C++ implementation of the MPEG-1 decoding algorithms) by

Mateus Krepsky Ludwich - mateus@lisha.ufsc.br

*/

#ifndef BLOCK_H
#define BLOCK_H

/* #include <sys/types.h> */

const int DCTSIZE = 8;  /* The basic DCT block is 8x8 samples */
const int DCTSIZE2 = 64; /* DCTSIZE squared; # of elements in a block */


typedef short DCTELEM;
typedef DCTELEM DCTBLOCK[DCTSIZE2];

class CoeffBlock {
public:
static void j_rev_dct_sparse (DCTBLOCK data,int pos);
static void j_rev_dct (DCTBLOCK data);
};
```

```
#endif // BLOCK_H
```

APÊNDICE B – CÓDIGO FONTE DA REFORMATAÇÃO DE BLOCOS NA ENTRADA DA IDCT

Arquivo libmpeg2_to_normal.h

```
/*
 libmpeg2_to_normal - Maps libmpeg2 IDCT entry to a normal IDCT entry.

 Copyright (C) 2008 Mateus Krepsky Ludwich.

 This program is free software; you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation; either version 2 of the License, or
 (at your option) any later version.

 This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU Library General Public License for more details.

 You should have received a copy of the GNU General Public License
 along with this program; if not, write to the Free Software
 Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

 Contact: mkludwich@gmail.com

 A copy of GNU General Public License can be found in copying.txt file.

*/
```

```
#ifndef LIBMPEG2_TO_NORMAL
#define LIBMPEG2_TO_NORMAL

void libmpeg2ToNormal(short * block);
void normalToLibmpeg2(short * block);

#endif
```

Arquivo libmpeg2_to_normal.c

```
/*
 libmpeg2_to_normal - Maps libmpeg2 IDCT entry to a normal IDCT entry.

 Copyright (C) 2008 Mateus Krepsky Ludwich.

 This program is free software; you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation; either version 2 of the License, or
 (at your option) any later version.

 This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU Library General Public License for more details.

 You should have received a copy of the GNU General Public License
 along with this program; if not, write to the Free Software
 Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

 Contact: mkludwich@gmail.com

 A copy of GNU General Public License can be found in copying.txt file.

*/
#include "libmpeg2_to_normal.h"
```

```
#include <assert.h>

#define LIBMPEG2_TO_NORMAL_MAP 55
#define NORMAL_TO_LIBMPEG2_MAP 66

#define L2N_TRACE 0

#if L2N_TRACE
#include <stdio.h>
#endif

void l2n_trace(char * msg) {
#if L2N_TRACE
printf("%s\n", msg);
#endif
}

int libmpeg2Order[64] = {0, 4, 1, 5, 2, 6, 3, 7,
32, 36, 33, 37, 34, 38, 35, 39,
8, 12, 9, 13, 10, 14, 11, 15,
40, 44, 41, 45, 42, 46, 43, 47,
16, 20, 17, 21, 18, 22, 19, 23,
48, 52, 49, 53, 50, 54, 51, 55,
24, 28, 25, 29, 26, 30, 27, 31,
56, 60, 57, 61, 58, 62, 59, 63
};

void oneToAnother(short * block, int mode) {
short aux[64];

int i;

for (i = 0; i < 64; ++ i) {
if (mode == LIBMPEG2_TO_NORMAL_MAP) {
aux[i] = block[libmpeg2Order[i]];
```

```
l2n_trace("LIBMPEG2_TO_NORMAL_MAP");
#if L2N_TRACE
printf("aux[%i] = block[%i];", i, libmpeg2Order[i]);
#endif
}
else {
assert(mode == NORMAL_TO_LIBMPEG2_MAP);
aux[libmpeg2Order[i]] = block[i];

l2n_trace("NORMAL_TO_LIBMPEG2_MAP");
}
}

for (i = 0; i < 64; ++ i) {
if (mode == LIBMPEG2_TO_NORMAL_MAP) {
block[i] = aux[i] >> 4; // block[i] = aux[i] / 16;
}
else {
assert(mode == NORMAL_TO_LIBMPEG2_MAP);
block[i] = aux[i] << 4; // block[i] = aux[i] * 16;
}
}
}

void libmpeg2ToNormal(short * block) {
oneToAnother(block, LIBMPEG2_TO_NORMAL_MAP);
}

void normalToLibmpeg2(short * block) {
oneToAnother(block, NORMAL_TO_LIBMPEG2_MAP);
}
```

APÊNDICE C – CÓDIGO FONTE DO TESTE DE CONFORMIDADE ISO/IEC 13818-4 PARA IDCT

Código fonte do teste de conformidade com o padrão MPEG-2 para a IDCT

Arquivo idct_match_test.h

```
/*
 utils_idct - Some functions usefull to IDCT/DCT testing.

 Copyright (C) 2008 Mateus Krepsky Ludwich.

 This program is free software; you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation; either version 2 of the License, or
 (at your option) any later version.

 This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU Library General Public License for more details.

 You should have received a copy of the GNU General Public License
 along with this program; if not, write to the Free Software
 Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

 Contact: mkludwich@gmail.com
```

```
 A copy of GNU General Public License can be found in copying.txt file.

*/

/* IDCT match test as described in ISO/IEC 13818-4 (Conformance testing) */

#ifndef IDCT_MATCH_TEST
#define IDCT_MATCH_TEST

#include <stdint.h> //int8_t, int16_t

typedef int16_t * (IDCTFunction) (int16_t * outputFdctBlock);

void testMatch(IDCTFunction * func);

#endif // IDCT_MATCH_TEST
```

Arquivo idct_match_test.c

```
/*
 utils_idct - Some functions usefull to IDCT/DCT testing.

 Copyright (C) 2008 Mateus Krepsky Ludwich.

 This program is free software; you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation; either version 2 of the License, or
 (at your option) any later version.

 This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU Library General Public License for more details.

 You should have received a copy of the GNU General Public License
```

```
 along with this program; if not, write to the Free Software
 Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

 Contact: mkludwich@gmail.com

 A copy of GNU General Public License can be found in copying.txt file.

*/

/* IDCT match test as described in ISO/IEC 13818-4 (Conformance testing) */
#include <stdlib.h> // calloc
#include <stdio.h>

#include "idct_match_test.h"
#include "match_report.h"
#include "../../block.h"
#include "../../random_numbers_generator/random_numbers_generator.h"
#include "../../reference_idct_and_fdct/fdct.h"
#include "../../reference_idct_and_fdct/idct.h"

#define NUM_OF_BLOCKS 1000000
// #define NUM_OF_BLOCKS 10
#define BLOCK_SIZE 64

#define IDCT_MATCH_TEST_TRACE 0

/* trace and debug functions */
idct_match_test_trace(char * msg) {
#if IDCT_MATCH_TEST_TRACE
printf("%s\n", msg);
#endif
}

/* Static ("private") Functions prototypes */
static void testMatchDataSet(IDCTFunction * func, int numOfBlocks, long L, long H);
```

```
static void testBlock(IDCTFunction * func, long * block);
static int16_t * generateReferenceIDCTOutputs(int16_t * outputFdctBlock);
static void match(int16_t * outputFdctBlock, int16_t * ref, int16_t * obt);
static long * generateBlock(long L, long H);

void report(int numOfBlocks, long L, long H);

/* Functions implementations */
void testMatch(IDCTFunction * func) {
idct_match_test_trace("idct_match_test.c, testMatch");
match_report_init();
testMatchDataSet(func, NUM_OF_BLOCKS, 256, 255);
testMatchDataSet(func, NUM_OF_BLOCKS, 5, 5);
testMatchDataSet(func, NUM_OF_BLOCKS, 384, 383);
}

/* Generate a number of 2 * numOfBlocks blocks: numOfBlocks blocks and
numOfBlocks of "inverted blocks" - each element of a block with its
signal inverted. And test they. */
void testMatchDataSet(IDCTFunction * func, int numOfBlocks, long L, long H) {
idct_match_test_trace("idct_match_test.c, testMatchDataSet");
#if IDCT_MATCH_TEST_TRACE
printf("L: %i, H: %i\n", L, H);
#endif
int i;
for (i = 0; i < numOfBlocks; ++i) {
long * block = generateBlock(L, H);

testBlock(func, block);

long * invertedBlock = block_invert(block, BLOCK_SIZE, INT32_TYPE);
testBlock(func, invertedBlock);

free(block);
free(invertedBlock);
```

```
}
report(numOfBlocks * 2, L, H);
}

void testBlock(IDCTFunction * func, long * block) {
idct_match_test_trace("idct_match_test.c, testBlock");

int i;
int16_t outputFdctBlock[BLOCK_SIZE];
double inputFdctBlock[BLOCK_SIZE];

for (i = 0; i < BLOCK_SIZE; ++i) {
inputFdctBlock[i] = (double) block[i];
}
ieee_fdct(outputFdctBlock, inputFdctBlock);


int16_t * ref = generateReferenceIDCTOutputs(outputFdctBlock);
int16_t * obt = func(outputFdctBlock);

match(outputFdctBlock, ref, obt);

free(ref);
free(obt);

#if IDCT_MATCH_TEST_TRACE
printf("testBlock: The block that is input for IDCTs: \n");
block_print(outputFdctBlock, BLOCK_SIZE, INT32_TYPE);
#endif
}

int16_t * generateReferenceIDCTOutputs(int16_t * outputFdctBlock) {
idct_match_test_trace("idct_match_test.c, generateReferenceIDCTOutputs");

double inputIdctBlock[BLOCK_SIZE];
```

```
int16_t * outputIdctBlock = (int16_t *) calloc(BLOCK_SIZE, sizeof(int16_t));

int i;
for (i = 0; i < BLOCK_SIZE; ++i) {
inputIdctBlock[i] = (double) outputFdctBlock[i];
}
ieee_idct(outputIdctBlock, inputIdctBlock);

return outputIdctBlock;
}


void match(int16_t * outputFdctBlock, int16_t * ref, int16_t * obt) {
idct_match_test_trace("idct_match_test.c, match");

#if IDCT_MATCH_TEST_TRACE
printf("match: ref block: \n");
block_print(ref, BLOCK_SIZE, INT8_TYPE);
printf("match: obt block: \n");
block_print(obt, BLOCK_SIZE, INT8_TYPE);
#endif

match_report_eval(outputFdctBlock, ref, obt, BLOCK_SIZE);

}

/* Generate a 8x8 block of random longs */
long * generateBlock(long L, long H) {
idct_match_test_trace("idct_match_test.c, generateBlock");

long * block = (long *) calloc(BLOCK_SIZE, sizeof(long));

int i;
for (i = 0; i < BLOCK_SIZE; ++i) {
block[i] = random_numbers_generator_rand(L, H);
```

```
}

return block;
}

void report(int numOfBlocks, long L, long H) {
match_report_numOfBlocks = numOfBlocks;
match_report_L = L;
match_report_H = H;

match_report_computeReport();
match_report_printReport();

match_report_reset();
}
```

Arquivo common.h

```
/*
 utils_idct - Some functions usefull to IDCT/DCT testing.

 Copyright (C) 2008 Mateus Krepsky Ludwich.

 This program is free software; you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation; either version 2 of the License, or
 (at your option) any later version.

 This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU Library General Public License for more details.

 You should have received a copy of the GNU General Public License
 along with this program; if not, write to the Free Software
```

```
 Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

 Contact: mkludwich@gmail.com

 A copy of GNU General Public License can be found in copying.txt file.

*/
#ifndef COMMON
#define COMMON

#define ROW_COL_SIZE 8

double transformedCoeficient(int i);

/* rowNumber == 0..7, positionInRow == 0..7 */
double getRowElement(double * block, int rowNumber, int positionInRow);
void setRowElement(double * block, double element, int rowNumber,
   int positionInRow);

/* columnNumber == 0..7, positionInColumn == 0..7 */
double getColumnElement(double * block, int columnNumber, int positionInColumn);
double setColumnElement(double * block, double element, int columnNumber,
int positionInColumn);

#endif //COMMON
```

Arquivo common.c

```
/*
 utils_idct - Some functions usefull to IDCT/DCT testing.

 Copyright (C) 2008 Mateus Krepsky Ludwich.

 This program is free software; you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
```

```
 the Free Software Foundation; either version 2 of the License, or
 (at your option) any later version.

 This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU Library General Public License for more details.

 You should have received a copy of the GNU General Public License
 along with this program; if not, write to the Free Software
 Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

 Contact: mkludwich@gmail.com

 A copy of GNU General Public License can be found in copying.txt file.

*/
#include <math.h>
#include <stdio.h>
#include "common.h"

#define TRACE_COMMON 0
#define DEBUG_COMMON 1

static void trace_common(char * msg) {
#if TRACE_COMMON
printf("%s\n", msg);
#endif
}

static void verifyArrayIndex(int index) {
#if DEBUG_COMMON
if (index >= ROW_COL_SIZE * ROW_COL_SIZE)
printf("Index: %i out of the bounds!\n", index);
#endif
```

```
}

double transformedCoeficient(int i) {
double coef;

    if (i == 0)
        coef = 1.0 / sqrt(2.0);
    else
        coef = 1.0;

//~ #if TRACE_COMMON
//~ printf("transformedCoeficient: %f\n", coef);
//~ #endif

    return coef;
}

double getRowElement(double * block, int rowNumber, int positionInRow) {
trace_common("getRowElement");
verifyArrayIndex(rowNumber * ROW_COL_SIZE + positionInRow);

return block[rowNumber * ROW_COL_SIZE + positionInRow];
}

void setRowElement(double * block, double element, int rowNumber,
   int positionInRow) {
trace_common("setRowElement");
verifyArrayIndex(rowNumber * ROW_COL_SIZE + positionInRow);

block[rowNumber * ROW_COL_SIZE + positionInRow] = element;

trace_common("setRowElement - out");
}
```

```
double getColumnElement(double * block, int columnNumber,
int positionInColumn) {
trace_common("getColumnElement");
verifyArrayIndex(positionInColumn * ROW_COL_SIZE + columnNumber);

return block[positionInColumn * ROW_COL_SIZE + columnNumber];
}

double setColumnElement(double * block, double element, int columnNumber,
int positionInColumn) {
trace_common("setColumnElement");
verifyArrayIndex(positionInColumn * ROW_COL_SIZE + columnNumber);

block[positionInColumn * ROW_COL_SIZE + columnNumber] = element;

trace_common("setColumnElement - out");
}
```

Arquivo fdct.h

```
/*
 utils_idct - Some functions usefull to IDCT/DCT testing.

 Copyright (C) 2008 Mateus Krepsky Ludwich.

 This program is free software; you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation; either version 2 of the License, or
 (at your option) any later version.

 This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU Library General Public License for more details.
```

```
 You should have received a copy of the GNU General Public License
 along with this program; if not, write to the Free Software
 Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

 Contact: mkludwich@gmail.com

 A copy of GNU General Public License can be found in copying.txt file.

*/
#ifndef FDCT
#define FDCT

/* Bidimensional FDCT in a 8x8 block */
void fdct(double * block);
void ieee_fdct(short * outputBlock, double * inputBlock);

/* Unidimensional FDCT in a 8 block */
void fdct1D(double * block, int rowColNumber, char rowOrCol);

#endif //FDCT
```

Arquivo fdct.c

```
/*
 utils_idct - Some functions usefull to IDCT/DCT testing.

 Copyright (C) 2008 Mateus Krepsky Ludwich.

 This program is free software; you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation; either version 2 of the License, or
 (at your option) any later version.

 This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU Library General Public License for more details.

 You should have received a copy of the GNU General Public License
 along with this program; if not, write to the Free Software
 Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

 Contact: mkludwich@gmail.com

 A copy of GNU General Public License can be found in copying.txt file.

*/
#include "fdct.h"
#include <string.h>
#include <math.h>
#include "common.h"

#define TRACE_FDCT 0

#if TRACE_FDCT
#include <stdio.h>
#endif


/* necessary this declaration to correct compilation */
long long int llround(double arg);

static void fdct_trace(char * msg, int i) {
#if TRACE_FDCT
printf("%s %i\n", msg, i);
#endif
}

void fdct1D(double * block, int rowColNumber, char rowOrCol) {
fdct_trace("fdct.c, fdct1D", 0);
```

```
double element, sum, expr;
int i, t;
double aux[ROW_COL_SIZE * ROW_COL_SIZE];

for (i = 0; i < ROW_COL_SIZE; ++ i) {
sum = 0.0;
for (t = 0; t < ROW_COL_SIZE; ++ t) {
expr = (2.0 * t + 1.0) * i * M_PI / (16.0);

if (rowOrCol == 'r') {
element = getRowElement(block, rowColNumber, t);
}
else {
element = getColumnElement(block, rowColNumber, t);
}

sum += element * cos(expr);
}

element = 0.5 * transformedCoeficient(i) * sum;
if (rowOrCol == 'r')
setRowElement(aux, element, rowColNumber, i);
else
setColumnElement(aux, element, rowColNumber, i);
}

/* coping modifications into block */
for (i = 0; i < ROW_COL_SIZE; ++ i) {
if (rowOrCol == 'r') {
element = getRowElement(aux, rowColNumber, i);
setRowElement(block, element, rowColNumber, i);
}
else {
element = getColumnElement(aux, rowColNumber, i);
```

```
setColumnElement(block, element, rowColNumber, i);
}
}

}

void fdctRow(double * block, int rowNumber) {
fdct_trace("fdct.c, fdctRow. rowNumber: ", rowNumber);
fdct1D(block, rowNumber, 'r');
}

void fdctColumn(double * block, int columnNumber) {
fdct_trace("fdct.c, fdctColumn. columnNumber: ", columnNumber);
fdct1D(block, columnNumber, 'c');
}


void fdct(double * block) {
fdct_trace("fdct.c, fdct", 0);

int i;

for (i = 0; i < ROW_COL_SIZE; ++ i) {
fdctRow(block, i);
}

for (i = 0; i < ROW_COL_SIZE; ++ i) {
fdctColumn(block, i);
}
}

void ieee_fdct(short * outputBlock, double * inputBlock) {
double aux[ROW_COL_SIZE * ROW_COL_SIZE]; /* 64 bits */
memcpy(aux, inputBlock, sizeof(double) * (ROW_COL_SIZE * ROW_COL_SIZE));
```

```
fdct(aux);

long long nearestInteger; /* 64 bits */
int i;
for (i = 0; i < ROW_COL_SIZE * ROW_COL_SIZE; ++ i) {
nearestInteger = llround(aux[i]);
/* cliping to range of (-2048, 2047) - 12 bits */
if (nearestInteger > 2047) {
nearestInteger = 2047;
}
else if (nearestInteger < -2048) {
nearestInteger = -2048;
}

outputBlock[i] = (short) nearestInteger;
}
}
```

Arquivo idct.h

```
/*
 utils_idct - Some functions usefull to IDCT/DCT testing.

 Copyright (C) 2008 Mateus Krepsky Ludwich.

 This program is free software; you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation; either version 2 of the License, or
 (at your option) any later version.

 This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU Library General Public License for more details.
```

```
 You should have received a copy of the GNU General Public License
 along with this program; if not, write to the Free Software
 Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

 Contact: mkludwich@gmail.com

 A copy of GNU General Public License can be found in copying.txt file.

*/
#ifndef IDCT
#define IDCT

#include <stdint.h> //int8_t, int16_t

/* Bidimensional IDCT in a 8x8 block */
void idct(double * block);
void ieee_idct(int16_t * outputBlock, double * inputBlock);

/* Unidimensional IDCT in a 8 block */
void idct1D(double * block, int rowColNumber, char rowOrCol);

#endif //IDCT
```

Arquivo idct.c

```
/*
 utils_idct - Some functions usefull to IDCT/DCT testing.

 Copyright (C) 2008 Mateus Krepsky Ludwich.

 This program is free software; you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation; either version 2 of the License, or
 (at your option) any later version.
```

```
 This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU Library General Public License for more details.

 You should have received a copy of the GNU General Public License
 along with this program; if not, write to the Free Software
 Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

 Contact: mkludwich@gmail.com

 A copy of GNU General Public License can be found in copying.txt file.

*/
#include "idct.h"
#include <string.h>
#include <math.h>
#include "common.h"

#define TRACE_IDCT 0

#if TRACE_IDCT
#include <stdio.h>
#endif


/* necessary this declaration to correct compilation */
long long int llround(double arg);

static void idct_trace(char * msg, int i) {
#if TRACE_IDCT
printf("%s %i\n", msg, i);
#endif
}
```

```
void idct1D(double * block, int rowColNumber, char rowOrCol) {
idct_trace("idct.c, idct1D", 0);

double element, sum, expr;
int i, t;
double aux[ROW_COL_SIZE * ROW_COL_SIZE];

for (i = 0; i < ROW_COL_SIZE; ++ i) {
sum = 0.0;
for (t = 0; t < ROW_COL_SIZE; ++ t) {
expr = (2.0 * i + 1.0) * t * M_PI / (16.0);

if (rowOrCol == 'r') {
element = getRowElement(block, rowColNumber, t);
}
else {
element = getColumnElement(block, rowColNumber, t);
}

sum += transformedCoeficient(t) * element * cos(expr);
}

element = 0.5 * sum;
if (rowOrCol == 'r')
setRowElement(aux, element, rowColNumber, i);
else
setColumnElement(aux, element, rowColNumber, i);
}

/* coping modifications into block */
for (i = 0; i < ROW_COL_SIZE; ++ i) {
if (rowOrCol == 'r') {
element = getRowElement(aux, rowColNumber, i);
setRowElement(block, element, rowColNumber, i);
}
```

```
else {
element = getColumnElement(aux, rowColNumber, i);
setColumnElement(block, element, rowColNumber, i);
}
}

}

void idctRow(double * block, int rowNumber) {
idct_trace("idct.c, idctRow. rowNumber: ", rowNumber);
idct1D(block, rowNumber, 'r');
}

void idctColumn(double * block, int columnNumber) {
idct_trace("idct.c, idctColumn. columnNumber: ", columnNumber);
idct1D(block, columnNumber, 'c');
}


void idct(double * block) {
idct_trace("idct.c, idct", 0);

int i;

for (i = 0; i < ROW_COL_SIZE; ++ i) {
idctRow(block, i);
}

for (i = 0; i < ROW_COL_SIZE; ++ i) {
idctColumn(block, i);
}
}

void ieee_idct(int16_t * outputBlock, double * inputBlock) {
double aux[ROW_COL_SIZE * ROW_COL_SIZE]; /* 64 bits */
```

```
memcpy(aux, inputBlock, sizeof(double) * (ROW_COL_SIZE * ROW_COL_SIZE));

idct(aux);

long long nearestInteger; /* 64 bits */
int i;
for (i = 0; i < ROW_COL_SIZE * ROW_COL_SIZE; ++ i) {
nearestInteger = llround(aux[i]);
/* cliping to range of (-256, 255) -> 9 bits */
if (nearestInteger > 255) {
nearestInteger = 255;
}
else if (nearestInteger < -256) {
nearestInteger = -256;
}

outputBlock[i] = (int16_t) nearestInteger;
}
}
```

APÊNDICE D – CÓDIGO FONTE DO TESTE DE CONFORMIDADE APLICADO NA IDCT DA LIBMPEG2

Arquivo match_test.c

```
/*
 Author: Mateus Krepsky Ludwich.
 Year: 2008
 Contact: mkludwich@gmail.com

*/

/* IDCT match test as described in ISO/IEC 13818-4 (Conformance testing) */

#include <stdint.h> //int8_t
#include <stdlib.h> // calloc

#include "../../utils_idct/tools/idct_match_test/idct_match_test.h"
#include "../../libmpeg2/idct_libmpeg2/idct.h"
#include "../../libmpeg2_to_normal/src/libmpeg2_to_normal.h"

int16_t * idctForTest(int16_t * block) {
int16_t inputBlock[64];

int i;

for (i = 0; i < 64; ++ i) {
inputBlock[i] = block[i];
}
```

```
normalToLibmpeg2(inputBlock);

int16_t * outputBlock = idct2DWithClip(inputBlock);

return outputBlock;
}

int main() {
testMatch(idctForTest);
return 0;
}
```

Arquivo idct.h

```
/*
Header for idct.c
Author: Mateus Krepsky Ludwich - mateus@lisha.ufsc.br
*/
#include <stdint.h> //int8_t, int16_t

int16_t * idct2DWithClip(int16_t * inputBlock);
```

Arquivo idct.c

```
/* idct.c - Adapted from
mpeg2dec, a free MPEG-2 video stream decoder
http://libmpeg2.sourceforge.net/
by

Mateus Krepsky Ludwich - mateus@lisha.ufsc.br

See original copyright bellow.
*/
```

```
/*
 * idct.c
 * Copyright (C) 2000-2003 Michel Lespinasse <walken@zoy.org>
 * Copyright (C) 1999-2000 Aaron Holtzman <aholtzma@ess.engr.uvic.ca>
 *
 * This file is part of mpeg2dec, a free MPEG-2 video stream decoder.
 * See http://libmpeg2.sourceforge.net/ for updates.
 *
 * mpeg2dec is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * mpeg2dec is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 */


#include <stdlib.h>
#include <inttypes.h>

#include "attributes.h"

#include <stdio.h>
#include <stdlib.h> // calloc

#define TRACE 0

#define W1 2841 /* 2048 * sqrt (2) * cos (1 * pi / 16) */
```

```
#define W2 2676 /* 2048 * sqrt (2) * cos (2 * pi / 16) */
#define W3 2408 /* 2048 * sqrt (2) * cos (3 * pi / 16) */
#define W5 1609 /* 2048 * sqrt (2) * cos (5 * pi / 16) */
#define W6 1108 /* 2048 * sqrt (2) * cos (6 * pi / 16) */
#define W7 565  /* 2048 * sqrt (2) * cos (7 * pi / 16) */

/*
#if 0
#define BUTTERFLY(t0,t1,W0,W1,d0,d1) \
do { \
    t0 = W0 * d0 + W1 * d1; \
    t1 = W0 * d1 - W1 * d0; \
} while (0)
#else
*/
#define BUTTERFLY(t0,t1,W0,W1,d0,d1) \
do { \
    int tmp = W0 * (d0 + d1); \
    t0 = tmp + (W1 - W0) * d1; \
    t1 = tmp - (W1 + W0) * d0; \
} while (0)
//#endif

#if TRACE
static void printBlock(int16_t * block, int n) {
    int i;
int line = 0;

    for (i = 0; i < n; i++) {
        if (line == 8) {
line = 0;
printf("\n");
}
printf("%d, ", block[i]);
line ++;
```

```
    }
printf("\n");
}
#endif

static inline void idct_row (int16_t * const block)
{
//~ #if TRACE
//~ printf("idct_row - begin \n");
//~ printBlock(block, 8);
//~ #endif

    int d0, d1, d2, d3;
    int a0, a1, a2, a3, b0, b1, b2, b3;
    int t0, t1, t2, t3;

    /* shortcut */
    if (likely (!(block[1] | ((int32_t *)block)[1] | ((int32_t *)block)[2] |
  ((int32_t *)block)[3]))) {
uint32_t tmp = (uint16_t) (block[0] >> 1);
tmp |= tmp << 16;
((int32_t *)block)[0] = tmp;
((int32_t *)block)[1] = tmp;
((int32_t *)block)[2] = tmp;
((int32_t *)block)[3] = tmp;
//~ #if TRACE
//~ printf("idct_row - end by shortcut \n");
//~ printBlock(block, 8);
//~ #endif
return;
    }

    d0 = (block[0] << 11) + 2048;
    d1 = block[1];
    d2 = block[2] << 11;
```

```
    d3 = block[3];
    t0 = d0 + d2;
    t1 = d0 - d2;
    BUTTERFLY (t2, t3, W6, W2, d3, d1);
    a0 = t0 + t2;
    a1 = t1 + t3;
    a2 = t1 - t3;
    a3 = t0 - t2;

    d0 = block[4];
    d1 = block[5];
    d2 = block[6];
    d3 = block[7];
    BUTTERFLY (t0, t1, W7, W1, d3, d0);
    BUTTERFLY (t2, t3, W3, W5, d1, d2);
    b0 = t0 + t2;
    b3 = t1 + t3;
    t0 -= t2;
    t1 -= t3;
    b1 = ((t0 + t1) >> 8) * 181;
    b2 = ((t0 - t1) >> 8) * 181;

    block[0] = (a0 + b0) >> 12;
    block[1] = (a1 + b1) >> 12;
    block[2] = (a2 + b2) >> 12;
    block[3] = (a3 + b3) >> 12;
    block[4] = (a3 - b3) >> 12;
    block[5] = (a2 - b2) >> 12;
    block[6] = (a1 - b1) >> 12;
    block[7] = (a0 - b0) >> 12;
//~ #if TRACE
//~ printf("idct_row - end \n");
//~ printBlock(block, 8);
//~ #endif
}
```

```
static inline void idct_col (int16_t * const block)
{
    int d0, d1, d2, d3;
    int a0, a1, a2, a3, b0, b1, b2, b3;
    int t0, t1, t2, t3;

    d0 = (block[8*0] << 11) + 65536;
    d1 = block[8*1];
    d2 = block[8*2] << 11;
    d3 = block[8*3];
    t0 = d0 + d2;
    t1 = d0 - d2;
    BUTTERFLY (t2, t3, W6, W2, d3, d1);
    a0 = t0 + t2;
    a1 = t1 + t3;
    a2 = t1 - t3;
    a3 = t0 - t2;

    d0 = block[8*4];
    d1 = block[8*5];
    d2 = block[8*6];
    d3 = block[8*7];
    BUTTERFLY (t0, t1, W7, W1, d3, d0);
    BUTTERFLY (t2, t3, W3, W5, d1, d2);
    b0 = t0 + t2;
    b3 = t1 + t3;
    t0 -= t2;
    t1 -= t3;
    b1 = ((t0 + t1) >> 8) * 181;
    b2 = ((t0 - t1) >> 8) * 181;

    block[8*0] = (a0 + b0) >> 17;
    block[8*1] = (a1 + b1) >> 17;
    block[8*2] = (a2 + b2) >> 17;
```

```
    block[8*3] = (a3 + b3) >> 17;
    block[8*4] = (a3 - b3) >> 17;
    block[8*5] = (a2 - b2) >> 17;
    block[8*6] = (a1 - b1) >> 17;
    block[8*7] = (a0 - b0) >> 17;
}

static inline int16_t clip(int16_t v) {
/* clip to the range of (-256, 255) i.e. 9 bits */
return (v < -256) ? -256 : ((v > 255) ? 255 : v);
}

static inline void idct2D(int16_t * block) {
    int i;

    for (i = 0; i < 8; i++) {
idct_row (block + 8 * i);
}

    for (i = 0; i < 8; i++) {
idct_col (block + i);
}
}

int16_t * idct2DWithClip(int16_t * inputBlock) {
int16_t block[64];
int16_t * outputBlock = (int16_t *) calloc(64, sizeof(int16_t));

int i;

for (i = 0; i < 64; ++i) {
block[i] = inputBlock[i];
}

idct2D(block);
```

```
for (i = 0; i < 64; ++i) {
outputBlock[i] = clip(block[i]);
}

return outputBlock;
}
```

APÊNDICE E – CÓDIGO FONTE DO TESTE DE CONFORMIDADE APLICADO NA IDCT DA API

Arquivo match_test.c

```
/*
 Author: Mateus Krepsky Ludwich.
 Year: 2008
 Contact: mkludwich@gmail.com

*/

/* IDCT match test as described in ISO/IEC 13818-4 (Conformance testing) */

#include <stdint.h> //int<X>_t
#include <stdlib.h> // calloc

#include "../../utils_idct/tools/idct_match_test/idct_match_test.h"

extern void c_idct2D(short * block);

static inline int16_t clip(int16_t v) {
/* clip to the range of (-256, 255) i.e. 9 bits */
return (v < -256) ? -256 : ((v > 255) ? 255 : v);
}

int16_t * idctForTest(int16_t * block) {
int16_t inputBlock[64];
int16_t * outputBlock = (int16_t *) calloc(64, sizeof(int16_t));
```

```
int i;

for (i = 0; i < 64; ++ i) {
inputBlock[i] = block[i];
}

c_idct2D(inputBlock);

for (i = 0; i < 64; ++ i) {
outputBlock[i] = clip(inputBlock[i]);
}

return outputBlock;
}

int main() {
testMatch(idctForTest);
return 0;
}
```

APÊNDICE F – CÓDIGO FONTE DO PROCEDIMENTO EXECUTADO NA ANÁLISE DE DESEMPENHO DA IDCT DA API EM RELAÇÃO A IDCT ORIGINAL DA LIBMPEG2

Arquivo main.cc

```
/*
 performance_analysis_test - compute the execution time of
 the libmpeg2 IDCT and API IDCT

 Copyright (C) 2008 Mateus Krepsky Ludwich.

 This program is free software; you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation; either version 2 of the License, or
 (at your option) any later version.

 This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU Library General Public License for more details.

 You should have received a copy of the GNU General Public License
 along with this program; if not, write to the Free Software
 Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

 Contact: mkludwich@gmail.com
```

```
 A copy of GNU General Public License can be found in copying.txt file.

*/
#include <iostream> // print in screen
#include <string>

#include "performance_analysis_test.hh"

using namespace std;

int main(int argc, char *argv[]) {
/* With this set to 1000, we have a precision of nanoseconds.
Maybe move this into inside PerformanceAnalysisTest. */
int numOfExecutionsToComputeOneMean = 1000;

//int numBlocksOfEachClass = 1000000;
int numBlocksOfEachClass = 100000;
PerformanceAnalysisTest * test =
new PerformanceAnalysisTest(numOfExecutionsToComputeOneMean,
numBlocksOfEachClass);

string report = test->execute();
cout << report << endl;

delete test;
return 0;
}
```

Arquivo performance_analysis_test.hh

```
/*
 performance_analysis_test - compute the execution time of
 the libmpeg2 IDCT and API IDCT

 Copyright (C) 2008 Mateus Krepsky Ludwich.
```

```
 This program is free software; you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation; either version 2 of the License, or
 (at your option) any later version.

 This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU Library General Public License for more details.

 You should have received a copy of the GNU General Public License
 along with this program; if not, write to the Free Software
 Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

 Contact: mkludwich@gmail.com

 A copy of GNU General Public License can be found in copying.txt file.

*/

#ifndef PERFORMANCE_ANALYSYS_TEST_HEADER
#define PERFORMANCE_ANALYSYS_TEST_HEADER

#include <string>

extern "C" {
#include "../utils_idct/block_generator/block_generator.h"
}

using namespace std;

class PerformanceAnalysisTest {
public:
PerformanceAnalysisTest(int numOfExecutionsToComputeOneMean,
```

```
int numBlocksOfEachClass);

string execute();

private:
int numOfExecutionsToComputeOneMean;
int numBlocksOfEachClass;

long int giveMeanTimeLibmpeg2(int16_t * _block);
long int giveMeanTimeAPI(int16_t * _block);
long int giveMeanTimeNormalToLibmpeg2Mapping(int16_t * _block);

string generateReport(long int meanOfMeansLibmpeg2,
  long int meanOfMeansAPI,
  BlockGeneratorData & data,
  long int meanOfMeansNormalToLibmpeg2);
};

#endif
```

Arquivo performance_analysis_test.cc

```
/*
 performance_analysis_test - compute the execution time of
 the libmpeg2 IDCT and API IDCT

 Copyright (C) 2008 Mateus Krepsky Ludwich.

 This program is free software; you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation; either version 2 of the License, or
 (at your option) any later version.

 This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU Library General Public License for more details.

 You should have received a copy of the GNU General Public License
 along with this program; if not, write to the Free Software
 Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

 Contact: mkludwich@gmail.com

 A copy of GNU General Public License can be found in copying.txt file.

*/
#include <stdint.h> //int8_t, int16_t
#include <vector>

#include "performance_analysis_test.hh"
#include "../utils_idct/string_utils.hh"
#include "../utils_idct/algorithms/algorithms.hh"


// To perform IDCTs
#include "../impl_api_dsp/src/idct_api_dsp.h"
// ---

// Used in Trace
#define PERFORMANCE_ANALYSYS_TEST_TRACE 0
#include <iostream>

// To call C functions
extern "C" {
#include "../utils_idct/chronometer/chronometer.h"

#include "../libmpeg2_to_normal/src/libmpeg2_to_normal.h"
#include "../libmpeg2/idct_libmpeg2/idct.h"
#include "../utils_idct/block.h"
```

```
#include "../utils_idct/reference_idct_and_fdct/fdct.h"
#include "../utils_idct/reference_idct_and_fdct/idct.h"
#include "../utils_idct/reference_idct_and_fdct/common.h"
#include "../utils_idct/random_numbers_generator/random_numbers_generator.h"
}
// ---

PerformanceAnalysisTest::PerformanceAnalysisTest(
int numOfExecutionsToComputeOneMean, int numBlocksOfEachClass) {
this->numOfExecutionsToComputeOneMean = numOfExecutionsToComputeOneMean;
this->numBlocksOfEachClass = numBlocksOfEachClass;
}

string PerformanceAnalysisTest::execute() {
vector<long int> meansTimeLibmpeg2;
vector<long int> meansTimeAPI;
vector<long int> meansTimeNormalToLibmpeg2;

long int meanOfMeansLibmpeg2;
long int meanOfMeansAPI;
long int meanOfMeansNormalToLibmpeg2;

BlockGeneratorData data;
block_generator_init(&data, MPEG2_TEST_IDCT_BLOCKS, numBlocksOfEachClass);
while (block_generator_hasBlockToGenerate(&data)) {
int16_t * block =  block_generator_generateBlockForIDCT(&data);

meansTimeLibmpeg2.push_back(giveMeanTimeLibmpeg2(block));
meansTimeAPI.push_back(giveMeanTimeAPI(block));
meansTimeNormalToLibmpeg2.push_back
(giveMeanTimeNormalToLibmpeg2Mapping(block));

meanOfMeansLibmpeg2 = Algorithms::mean(meansTimeLibmpeg2);
meanOfMeansAPI = Algorithms::mean(meansTimeAPI);
meanOfMeansNormalToLibmpeg2 =
```

```
Algorithms::mean(meansTimeNormalToLibmpeg2);
}

return generateReport(meanOfMeansLibmpeg2, meanOfMeansAPI, data,
  meanOfMeansNormalToLibmpeg2);
}

string PerformanceAnalysisTest::generateReport(long int meanOfMeansLibmpeg2,
long int meanOfMeansAPI, BlockGeneratorData & data,
long int meanOfMeansNormalToLibmpeg2) {
string report = "Report: \n";
report += "Mean Time of libmpeg2 IDCT execution: " +
StringUtils::longToString(meanOfMeansLibmpeg2) + "\n";
report += "Mean Time of API IDCT execution: " +
StringUtils::longToString(meanOfMeansAPI) + "\n";
report += "Difference (libmpeg2 - API): " +
StringUtils::longToString(meanOfMeansLibmpeg2 - meanOfMeansAPI) + "\n";
report += "Number of computed blocks: " +
StringUtils::intToString(data.numGeneratedBlocks) + "\n";
if (meanOfMeansLibmpeg2 > meanOfMeansAPI)
report += "The IDCT of API is better than IDCT of libmpeg2\n";
else if (meanOfMeansLibmpeg2 < meanOfMeansAPI)
report += "The IDCT of API is worse than IDCT of libmpeg2\n";
else
report += "The IDCT of API and IDCT of libmpeg2 are equivalent\n";
report += "Mean Time of normalToLibmpeg2 (not used to compute\
the time of libmpeg2 IDCT): " +
StringUtils::longToString(meanOfMeansNormalToLibmpeg2) + "\n";

return report;
}

long int PerformanceAnalysisTest::giveMeanTimeLibmpeg2(int16_t * _block) {
#if PERFORMANCE_ANALYSYS_TEST_TRACE
cout << "PerformanceAnalysisTest::giveMeanTimeLibmpeg2: " << endl;
```

```
#endif
int16_t * block = (int16_t *) block_copy(_block, BLOCK_SIZE, INT16_TYPE);
normalToLibmpeg2(block);

ChronometerData data;
chronometer_clear(&data);
chronometer_start(&data);

for(int i = 0; i < numOfExecutionsToComputeOneMean; ++i) {
libmpeg2_idct2D(block);
}

chronometer_stop(&data);

#if PERFORMANCE_ANALYSYS_TEST_TRACE
cout << "giveMeanTimeLibmpeg2: "
<< chronometer_getDiffTimeMicro(&data) << endl;
#endif
free(block);

return chronometer_getDiffTimeMicro(&data);
}

long int PerformanceAnalysisTest::giveMeanTimeAPI(int16_t * _block) {
#if PERFORMANCE_ANALYSYS_TEST_TRACE
cout << "PerformanceAnalysisTest::giveMeanTimeAPI: " << endl;
#endif
int16_t * block = (int16_t *) block_copy(_block, BLOCK_SIZE, INT16_TYPE);

ChronometerData data;
chronometer_clear(&data);
chronometer_start(&data);

for(int i = 0; i < numOfExecutionsToComputeOneMean; ++i) {
c_idct2D(block); // (with wrapper - the real case)
```

```
//idct2D(block); // (without wrapper)
}

chronometer_stop(&data);

#if PERFORMANCE_ANALYSYS_TEST_TRACE
cout << "giveMeanTimeAPI: " << chronometer_getDiffTimeMicro(&data) << endl;
#endif
free(block);

return chronometer_getDiffTimeMicro(&data);
}

long int PerformanceAnalysisTest::
giveMeanTimeNormalToLibmpeg2Mapping(int16_t * _block) {
#if PERFORMANCE_ANALYSYS_TEST_TRACE
cout << "PerformanceAnalysisTest::giveMeanTimeNormalToLibmpeg2Mapping: "<<endl;
#endif
int16_t * block = (int16_t *) block_copy(_block, BLOCK_SIZE, INT16_TYPE);

ChronometerData data;
chronometer_clear(&data);
chronometer_start(&data);

for(int i = 0; i < numOfExecutionsToComputeOneMean; ++i) {
normalToLibmpeg2(block);
}

chronometer_stop(&data);

#if PERFORMANCE_ANALYSYS_TEST_TRACE
cout << "giveMeanTimeNormalToLibmpeg2Mapping: " <<
chronometer_getDiffTimeMicro(&data) << endl;
#endif
free(block);
```

```
return chronometer_getDiffTimeMicro(&data);
}
```

ANEXO A – CÓDIGO FONTE DA LIBMPEG2 - ARQUIVOS MODIFICADOS

Este anexo contém o código fonte dos arquivos da libmpeg2 que foram alterados neste trabalho, já com as alterações aplicadas.

Código fonte do programa sample1

```
/*
 * sample1.c
 * Copyright (C) 2003      Regis Duchesne <hpreg@zoy.org>
 * Copyright (C) 2000-2003 Michel Lespinasse <walken@zoy.org>
 * Copyright (C) 1999-2000 Aaron Holtzman <aholtzma@ess.engr.uvic.ca>
 *
 * This file is part of mpeg2dec, a free MPEG-2 video stream decoder.
 * See http://libmpeg2.sourceforge.net/ for updates.
 *
 * mpeg2dec is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * mpeg2dec is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
```

```
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * This program reads a MPEG-2 stream, and saves each of its frames as
 * an image file using the PGM format (black and white).
 *
 * It demonstrates how to use the following features of libmpeg2:
 * - Output buffers use the YUV 4:2:0 planar format.
 * - Output buffers are allocated and managed by the library.
 */

#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>

#include "mpeg2.h"

#define TRACE_PARSER 1

static void save_pgm (int width, int height,
      int chroma_width, int chroma_height,
      uint8_t * const * buf, int num)
{
    char filename[100];
    FILE * pgmfile;
    int i;
    static uint8_t black[16384] = { 0 };

    sprintf (filename, "%d.pgm", num);
    pgmfile = fopen (filename, "wb");
    if (!pgmfile) {
fprintf (stderr, "Could not open file \"%s\".\n", filename);
exit (1);
    }
    fprintf (pgmfile, "P5\n%d %d\n255\n",
```

```
     2 * chroma_width, height + chroma_height);
    for (i = 0; i < height; i++) {
fwrite (buf[0] + i * width, width, 1, pgmfile);
fwrite (black, 2 * chroma_width - width, 1, pgmfile);
    }
    for (i = 0; i < chroma_height; i++) {
fwrite (buf[1] + i * chroma_width, chroma_width, 1, pgmfile);
fwrite (buf[2] + i * chroma_width, chroma_width, 1, pgmfile);
    }
    fclose (pgmfile);
}

static void sample1 (FILE * mpgfile)
{
#define BUFFER_SIZE 4096
    uint8_t buffer[BUFFER_SIZE];
    mpeg2dec_t * decoder;
    const mpeg2_info_t * info;
    const mpeg2_sequence_t * sequence;
    mpeg2_state_t state;
    size_t size;
    int framenum = 0;

mpeg2_accel (0);

    decoder = mpeg2_init ();
    if (decoder == NULL) {
fprintf (stderr, "Could not allocate a decoder object.\n");
exit (1);
    }
    info = mpeg2_info (decoder);

    size = (size_t)-1;
    do {
state = mpeg2_parse (decoder);
```

```
sequence = info->sequence;
switch (state) {
case STATE_BUFFER:
    size = fread (buffer, 1, BUFFER_SIZE, mpgfile);
    mpeg2_buffer (decoder, buffer, buffer + size);
    break;
case STATE_SLICE:
case STATE_END:
case STATE_INVALID_END:
    if (info->display_fbuf)
save_pgm (sequence->width, sequence->height,
  sequence->chroma_width, sequence->chroma_height,
  info->display_fbuf->buf, framenum++);
    break;
default:
    break;
}
    } while (size);

    mpeg2_close (decoder);
}

int main (int argc, char ** argv)
{
    FILE * mpgfile;

    if (argc > 1) {
mpgfile = fopen (argv[1], "rb");
if (!mpgfile) {
    fprintf (stderr, "Could not open file \"%s\".\n", argv[1]);
    exit (1);
}
    } else
mpgfile = stdin;
```

```
    sample1 (mpgfile);

#if TRACE_PARSER
printf("$\n");
#endif

    return 0;
}
```

Arquivo idct.c

```
/*
 * idct.c
 * Copyright (C) 2000-2003 Michel Lespinasse <walken@zoy.org>
 * Copyright (C) 1999-2000 Aaron Holtzman <aholtzma@ess.engr.uvic.ca>
 *
 * This file is part of mpeg2dec, a free MPEG-2 video stream decoder.
 * See http://libmpeg2.sourceforge.net/ for updates.
 *
 * mpeg2dec is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * mpeg2dec is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 */

#include "config.h"
```

```
#include <stdlib.h>
#include <inttypes.h>

#include <stdio.h>
#include <assert.h>

#include "mpeg2.h"
#include "attributes.h"
#include "mpeg2_internal.h"

#include "../../../../libmpeg2_to_normal/src/libmpeg2_to_normal.h"
#include "../../../../utils_idct/utils_idct_c.h"
#include "../../../../utils_idct/block.h"

#define TRACE_PARSER 0 // "print blocks before CLIP"
#define TRACE_PARSER2 0 // "print blocks after CLIP"
#define TRACE_PARSER3 1 // "print blocks before CLIP and after idctClip"
#define TRACE_PARSER4 1 // "print blocks before IDCT and after libmpeg2ToNormal"
#define IDCT_DEBUG 0

#define W1 2841 /* 2048 * sqrt (2) * cos (1 * pi / 16) */
#define W2 2676 /* 2048 * sqrt (2) * cos (2 * pi / 16) */
#define W3 2408 /* 2048 * sqrt (2) * cos (3 * pi / 16) */
#define W5 1609 /* 2048 * sqrt (2) * cos (5 * pi / 16) */
#define W6 1108 /* 2048 * sqrt (2) * cos (6 * pi / 16) */
#define W7 565  /* 2048 * sqrt (2) * cos (7 * pi / 16) */

/* idct main entry point  */
void (* mpeg2_idct_copy) (int16_t * block, uint8_t * dest, int stride);
void (* mpeg2_idct_add) (int last, int16_t * block,
 uint8_t * dest, int stride);

/*
 * In legal streams, the IDCT output should be between -384 and +384.
```

```
 * In corrupted streams, it is possible to force the IDCT output to go
 * to +-3826 - this is the worst case for a column IDCT where the
 * column inputs are 16-bit values.
 */
uint8_t mpeg2_clip[3840 * 2 + 256];
#define CLIP(i) ((mpeg2_clip + 3840)[i])

#if 0
#define BUTTERFLY(t0,t1,W0,W1,d0,d1) \
do { \
    t0 = W0 * d0 + W1 * d1; \
    t1 = W0 * d1 - W1 * d0; \
} while (0)
#else
#define BUTTERFLY(t0,t1,W0,W1,d0,d1) \
do { \
    int tmp = W0 * (d0 + d1); \
    t0 = tmp + (W1 - W0) * d1; \
    t1 = tmp - (W1 + W0) * d0; \
} while (0)
#endif

extern void c_idct2D(short * block);

/* Trace/Debug functions - begin */
static void traceParser(int16_t * block, char * extraInfo) {
#if TRACE_PARSER
printf("begin_block #%s\n", extraInfo);
printBlock(block, 64);
printf("end_block\n");
#endif
}

static void traceParser2Block(unsigned char * block) {
#if TRACE_PARSER2
```

```
printBlock(block, 8, UINT8_TYPE);
#endif
}

static void traceParser2BeginBlock(char * extraInfo) {
#if TRACE_PARSER2
printf("begin_block #%s\n", extraInfo);
#endif
}

static void traceParser2EndBlock(char * extraInfo) {
#if TRACE_PARSER2
printf("end_block #%s\n", extraInfo);
#endif
}

static void traceParser3(int16_t * block, char * extraInfo) {
#if TRACE_PARSER3
int16_t clipedBlock[64];

int i;
for (i = 0; i < 64; ++ i) {
clipedBlock[i] = idctClip(block[i]);
}

printf("begin_block #%s\n", extraInfo);
printBlock(clipedBlock, 64, INT16_TYPE);
printf("end_block\n");


#endif
}

static void traceParser4(int16_t * block, char * extraInfo) {
#if TRACE_PARSER4
```

```
printf("begin_input_block #%s\n", extraInfo);
printBlock(block, 64, INT16_TYPE);
printf("end_input_block\n");
#endif
}

/* Trace/Debug functions - end */


static void mpeg2_idct_copy_c (int16_t * block, uint8_t * dest,
       const int stride)
{
traceParser(block, "copy, before idct");
#if IDCT_DEBUG
short * originalBlock = copyBlock(block, 64);
#endif

int i = 8;
libmpeg2ToNormal(block);

traceParser4(block, "copy");

#if IDCT_DEBUG
short * restauredBlock = copyBlock(block, 64);
normalToLibmpeg2(restauredBlock);
assert(equals(originalBlock, restauredBlock, 64));
#endif

c_idct2D(block);

traceParser(block, "copy, after idct");

traceParser3(block, "copy");

traceParser2BeginBlock("copy");
```

```
    do {
dest[0] = CLIP (block[0]);
dest[1] = CLIP (block[1]);
dest[2] = CLIP (block[2]);
dest[3] = CLIP (block[3]);
dest[4] = CLIP (block[4]);
dest[5] = CLIP (block[5]);
dest[6] = CLIP (block[6]);
dest[7] = CLIP (block[7]);

traceParser2Block(dest);

((int32_t *)block)[0] = 0; ((int32_t *)block)[1] = 0;
((int32_t *)block)[2] = 0; ((int32_t *)block)[3] = 0;

dest += stride;
block += 8;
    } while (--i);

traceParser2EndBlock("copy");

}

static void mpeg2_idct_add_c (const int last, int16_t * block,
      uint8_t * dest, const int stride)
{
int i = 8;

    if (last != 129 || (block[0] & (7 << 4)) == (4 << 4)) {
traceParser(block, "add, before idct");

#if IDCT_DEBUG
short * originalBlock = copyBlock(block, 64);
#endif
```

```
libmpeg2ToNormal(block);

traceParser4(block, "add");

#if IDCT_DEBUG
short * restauredBlock = copyBlock(block, 64);
normalToLibmpeg2(restauredBlock);
assert(equals(originalBlock, restauredBlock, 64));
#endif

c_idct2D(block);

traceParser(block, "add, after idct");

traceParser3(block, "add");

traceParser2BeginBlock("add, p1");

do {
    dest[0] = CLIP (block[0] + dest[0]);
    dest[1] = CLIP (block[1] + dest[1]);
    dest[2] = CLIP (block[2] + dest[2]);
    dest[3] = CLIP (block[3] + dest[3]);
    dest[4] = CLIP (block[4] + dest[4]);
    dest[5] = CLIP (block[5] + dest[5]);
    dest[6] = CLIP (block[6] + dest[6]);
    dest[7] = CLIP (block[7] + dest[7]);

traceParser2Block(dest);

    ((int32_t *)block)[0] = 0; ((int32_t *)block)[1] = 0;
    ((int32_t *)block)[2] = 0; ((int32_t *)block)[3] = 0;

    dest += stride;
```

```
    block += 8;
} while (--i);

traceParser2EndBlock("add, p1");

    } else {
int DC;

//~ traceParser2BeginBlock("add, p2");

DC = (block[0] + 64) >> 7;
block[0] = block[63] = 0;
i = 8;
do {
    dest[0] = CLIP (DC + dest[0]);
    dest[1] = CLIP (DC + dest[1]);
    dest[2] = CLIP (DC + dest[2]);
    dest[3] = CLIP (DC + dest[3]);
    dest[4] = CLIP (DC + dest[4]);
    dest[5] = CLIP (DC + dest[5]);
    dest[6] = CLIP (DC + dest[6]);
    dest[7] = CLIP (DC + dest[7]);

//~ traceParser2Block(dest);

    dest += stride;
} while (--i);
//~ traceParser2EndBlock("add, p2");
    }
}

void mpeg2_idct_init (uint32_t accel)
{
#ifdef ARCH_X86
    if (accel & MPEG2_ACCEL_X86_MMXEXT) {
```

```
mpeg2_idct_copy = mpeg2_idct_copy_mmxext;
mpeg2_idct_add = mpeg2_idct_add_mmxext;
mpeg2_idct_mmx_init ();
    } else if (accel & MPEG2_ACCEL_X86_MMX) {
mpeg2_idct_copy = mpeg2_idct_copy_mmx;
mpeg2_idct_add = mpeg2_idct_add_mmx;
mpeg2_idct_mmx_init ();
    } else
#endif
#if defined(ARCH_PPC) && defined(HAVE_ALTIVEC)
    if (accel & MPEG2_ACCEL_PPC_ALTIVEC) {
mpeg2_idct_copy = mpeg2_idct_copy_altivec;
mpeg2_idct_add = mpeg2_idct_add_altivec;
mpeg2_idct_altivec_init ();
    } else
#endif
#ifdef ARCH_ALPHA
    if (accel & MPEG2_ACCEL_ALPHA_MVI) {
mpeg2_idct_copy = mpeg2_idct_copy_mvi;
mpeg2_idct_add = mpeg2_idct_add_mvi;
mpeg2_idct_alpha_init ();
    } else if (accel & MPEG2_ACCEL_ALPHA) {
int i;

mpeg2_idct_copy = mpeg2_idct_copy_alpha;
mpeg2_idct_add = mpeg2_idct_add_alpha;
mpeg2_idct_alpha_init ();
for (i = -3840; i < 3840 + 256; i++)
    CLIP(i) = (i < 0) ? 0 : ((i > 255) ? 255 : i);
    } else
#endif
    {
extern uint8_t mpeg2_scan_norm[64];
extern uint8_t mpeg2_scan_alt[64];
int i, j;
```

```
mpeg2_idct_copy = mpeg2_idct_copy_c;
mpeg2_idct_add = mpeg2_idct_add_c;
for (i = -3840; i < 3840 + 256; i++)
    CLIP(i) = (i < 0) ? 0 : ((i > 255) ? 255 : i);
for (i = 0; i < 64; i++) {
    j = mpeg2_scan_norm[i];
    mpeg2_scan_norm[i] = ((j & 0x36) >> 1) | ((j & 0x09) << 2);
    j = mpeg2_scan_alt[i];
    mpeg2_scan_alt[i] = ((j & 0x36) >> 1) | ((j & 0x09) << 2);
}
    }
}
```

ANEXO B – ARTIGO

Adaptação de um Decodificador de Vídeo Digital a uma API DSP Multiplataforma

Mateus Krepsky Ludwich[1]

[1]Laboratório de Integração Software Hardware – LISHA
Departamento de Informática e Estatística – INE
Universidade Federal de Santa Catarina – UFSC
Florianópolis – SC – Brasil

{mateus}@lisha.ufsc.br

Abstract. *In video codecs the stages of digital signal processing are the more complex in the computational sense. Commonly these stages are written total or partially in assembly language of specific architectures. At the same time that this contributes for a performance increase, this reduces the portability of the codecs. The utilization of a API that provides features of digital signal processing (DSP) and primitive of digital signal processors, can contribute in increasing the portability of codecs.*
This work applied a DSP API (DSP/DSP-HW) in a MPEG-2 video decoder. The stage of Inverse Discrete Cosine Transform (IDCT) was modified to use the API. The MPEG-2 part 4 conformance test for IDCT was applied on this modification. The modified decoder passed in this test. A performance analysis also was made, where the modified decoder was compared with the original one. The performance of modified decoder was satisfactory. Another stage of MPEG-2 that was worked was the motion compensation (MC). Then was generated notes where the utilization of the API is possible in this stage.

Resumo. *Em codecs de vídeo os estágios de processamento digital de sinais são os mais complexos do ponto de vista computacional. Comumente tais estágios são escritos total ou parcialmente em linguagem assembly de arquiteturas específicas. Isto ao mesmo passo que contribui para um aumento no desempenho, reduz a portabilidade dos codecs. A utilização de uma API que concentre em si funcionalidades de processamento digital de sinais (DSP) e primitivas de processadores digitais de sinais (DSP-HW), pode contribuir no aumento da portabilidade dos codecs.*
Este trabalho aplicou uma API DSP (DSP/DSP-HW) em um decodificador de vídeo MPEG-2. Modificou-se o estágio da realização da Transformada Inversa Discreta de Cosseno (IDCT), para usar a API. Executou-se sobre esta modificação o teste de conformidade descrito no padrão MPEG-2 parte 4. O decodificador modificado demonstrou-se correto de acordo com o teste. Realizou-se também uma análise de desempenho comparando o decodificador modificado com o original. O desempenho do decodificador modificado mostrou-se satisfatório. Outro estágio do MPEG-2 trabalhado foi a compensação de movimento (MC), sobre o qual foram gerados apontamentos de onde é possível a utilização da API.

1. Introdução

Processos de codificação e decodificação de vídeo lidam com processamento de sinais, afinal o vídeo é um tipo de sinal. São chamados de codecs os componentes que implementam tais processos. É desejável que codecs implementados em software sejam portáveis com desempenho satisfatório para diversas plataformas de hardware, como processadores de uso geral, sistemas dedicados desenvolvidos em lógica programável e processadores digitais de sinais com instruções dedicadas. Em codecs de vídeo os estágios de processamento digital de sinais são os mais complexos do ponto de vista computacional. Em virtude disto é freqüente a busca de uma otimização de tais estágios, o que é em geral obtido implementando-os total ou parcialmente em linguagem assembly de arquiteturas específicas. O lado ruim de tal abordagem é que a portabilidade dos codecs é prejudicada. A utilização de uma API que concentre em si funcionalidades de processamento digital de sinais e correlatas, pode contribuir na solução do problema de portabilidade dos codecs, uma vez que ela proverá aos mesmos uma forma transparente de acessar tais funcionalidades. No presente trabalho buscou-se adaptar um decodificador de vídeo padrão MPEG-2 previamente existente a uma API que contempla tais requisitos de portabilidade e desempenho.

As seções que seguem estão organizados da seguinte maneira. As seções 2 e 3 apresentam, respectivamente, a API DSP e o decodificador utilizados neste trabalho. A seção 4 mostra como o decodificador foi adaptado para usar a API, assim como os testes realizados e os apontamentos gerados. As considerações finais são apresentadas na seção 5.

2. API DSP

Neste projeto foi utilizada uma API que oferece funcionalidades de processamento digital de sinais e primitivas de processadores digitais de sinais. Esta é o trabalho de Danillo Moura Santos, co-orientador deste projeto. A principal idéia desta API é de prover funcionalidades de DSP e correlatas de forma transparente para seus clientes (e.g. codificadores e decodificadores), possuindo implementações otimizadas para para diversas arquiteturas heterogêneas, como por exemplo processadores digitais de sinais com instruções dedicadas, processadores de propósito geral, sistemas dedicados desenvolvidos em FPGA. No presente momento a API se encontra em desenvolvimento, portanto as funções providas ainda não se encontram nas suas versões definitivas, e novas funções serão elaboradas.

3. Libmpeg2

Neste projeto foi utilizada a biblioteca libmpeg2. Esta implementa um decodificador que possui conformidade com o padrão MPEG-2 (“main profile”) e trata-se de um software livre, ou seja, com código fonte disponível e passível de ser alterado. A libmpeg2 é amplamente utilizada e depurada. Em [lib 2008a] existe uma lista de projetos que usam a libmpeg2, dentre eles podemos citar os seguintes players multimídia: VLC (VideoLAN), MPlayer e Xine.

A libmpeg2 é escrita na linguagem C. Entretanto, alguns trechos da biblioteca possuem versões em assembly (além da versão em C), trechos estes associados a estágios do processo de decodificação que podem aproveitar-se de otimizações específicas de determinadas plataformas de hardware. Os estágios da libmpeg2 que possuem otimizações em assembly são a IDCT e a compensação de movimento (Motion Compensation - MC).

As plataformas para as quais atualmente existem otimizações em assembly são: x86 (instruções MMX) e Power PC - ppc (instruções AltiVec).

4. Adaptação do decodificador

Esta seção descreve quais partes do decodificador foram trabalhadas visando sua integração com a API DSP. Primeiramente são explicadas as modificações que foram realizadas na IDCT do decodificador, assim como os testes relativos a estas modificações. Em seguida é apresentado como a implementação da compensação de movimento está estruturada no decodificador escolhido e são apontados os pontos do mesmo que podem fazer uso da API.

4.1. Modificações na IDCT

4.1.1. Função da API utilizada

Na modificação do decodificador no estágio de IDCT, onde a implementação de IDCT original da libmpeg2 foi trocada pela implementação fornecida pela API DSP, utilizou-se a seguinte função da API:
*void idct2D(DCTELEM *dctWeights);*
onde *DCTELEM* é um *short* (inteiro de 16 bits sinalizados) e *dctWeights* é o bloco de entrada (e saída) da IDCT. Esta função, opera uma IDCT bidimensional sobre um bloco de 8x8 elementos.

4.1.2. Alterações realizadas

Na libmpeg2, existem dois ponteiros de função que são associados a funções relacionadas com a IDCT. São eles: mpeg2_idct_copy e mpeg2_idct_add, utilizados, respectivamente, na decodificação intra e inter de quadros/campos MPEG-2. Em tempo de compilação são disponibilizadas as funções com implementação em assembly para plataformas específicas (caso existam para a plataforma em questão) e as funções escritas em C (sempre disponíveis). As funções C que podem ser associadas aos ponteiros mpeg2_idct_copy e mpeg2_idct_add são, mpeg2_idct_copy_c e mpeg2_idct_add_c (nesta ordem). Os nomes das funções que são implementadas em assembly dependem da plataforma: mpeg2_idct_copy_altivec, por exemplo, é a função para a plataforma Power PC (com instruções AltiVec) que pode ser associada ao ponteiro mpeg2_idct_copy. Em tempo de execução, se houverem funções otimizadas, é possível escolher entre elas e as versões em C.

No presente trabalho as funções mpeg2_idct_copy_c e mpeg2_idct_add_c foram modificadas para utilizarem a função de IDCT provida pela API. Não houve preocupação em alterar as funções com implementação em assembly, pois as implementações da IDCT da própria API é que poderão prover versões em assembly, otimizadas para plataformas espefíficas. Aqui, é demonstrado como a função mpeg2_idct_copy_c foi alterada. A função mpeg2_idct_add_c foi alterada de forma muito semelhante, por isto não é mostrada aqui.

A função mpeg2_idct_copy_c possui a seguinte assinatura:
*void mpeg2_idct_copy_c (int16_t * block, uint8_t * dest, const int stride);* onde: *block* é o bloco de entrada da IDCT, *dest* é o bloco de saída *stride* indica o quanto se avança no

bloco de saída, a medida em que ele está sendo reconstruído. A implementação original é basicamente a seguinte:

```
static void mpeg2_idct_copy_c (int16_t * block, uint8_t * dest,
const int stride)
{
int i;

/* Realização da IDCT */
for (i = 0; i < 8; i++)
idct_row (block + 8 * i);
for (i = 0; i < 8; i++)
idct_col (block + i);

// ...

}
```

Onde é aplicado uma IDCT unidimensional nas linhas do bloco e, em seguida, outra IDCT de uma dimensão em cada coluna do bloco alterado. Esta função pode ser observada na íntegra nos fontes da libmpeg2 (versão mpeg2dec-0.4.1), que podem ser obtidos em [lib 2008a].

A versão alterada de mpeg2_idct_copy_c, utilizando-se a API, fica da seguinte forma:

```
static void mpeg2_idct_copy_c (int16_t * block, uint8_t * dest,
const int stride)
{
int i = 8;
libmpeg2ToNormal(block);

/* Realização da IDCT */
c_idct2D(block);

// ...

}
```

Como a função *idct2D* , provida pela API é escrita em C++, e a libmpeg2 em C, é necessário a utilização da função wrapper *c_idct2D* .

Utilizando-se a IDCT da libmpeg2 de forma isolada percebeu-se que, para um mesmo conjunto de entradas, a saída da IDCT da libmpeg2 era diferente de uma IDCT convencional. Verificou-se então que a IDCT da libmpeg2 exige que sua entrada esteja em um formato específico para operar corretamente e gerar uma saída igual a de uma IDCT convencional. Esta formatação acontece quando os blocos que entrarão na IDCT ainda estão sendo reconstruídos (estágios de Decodificação de comprimento variável, Esquadrinhamento inverso e Quantização inversa). A formatação consiste em uma reordenação dos elementos do bloco e na **multiplicação** de cada elemento por 16.

Considerando que a IDCT provida pela API é uma IDCT convencional, exigindo blocos "não formatados" como entrada, duas opções existiam: a primeira consistiria em alterar a reconstrução dos blocos na libmpeg2 nos estágios anteriores à IDCT para que não efetuassem nenhuma formatação, a segunda seria a de reverter tal formatação. Optou-se pela segunda opção, pois apesar de ela gerar um pequeno overhead (já que todos os blocos na antes de entrarem na IDCT são "reformatados"), ela não é uma modificação tão intrusiva já que não altera as diversas funções da libmpeg2 responsáveis pela montagem do bloco, e ela é mais localizada, ocorrendo apenas na entrada da IDCT e em mais nenhum outro lugar. A reformatação consiste basicamente na "reordenação inversa" dos elementos do bloco e na **divisão** de cada elemento por 16.

Como mencionado anteriormente, as implementações de *void idct2D(DCTELEM *dctWeights);* é que poderão prover versões em assembly, otimizadas para plataformas específicas. Portanto tais otimizações específicas a cada plataforma, e a própria portabilidade (neste sentido), passam a residir não mais nas várias versões das funções associadas a mpeg2_idct_copy e mpeg2_idct_add e sim na função provida pela API. Desta forma a granularidade das otimizações para plataformas se torna mais fina e focada na API, assim o desenvolvedor do decodificador passa a se preocupar apenas na interface das funções DSP que utilizará e não nas implementações das mesmas.

4.2. Teste de conformidade e Análise de desempenho

4.2.1. Teste de conformidade

O padrão MPEG-2 parte 4 (testes de conformidade)[ISO/IEC 1998] descreve como testar uma IDCT e verificar se ela pode ser considerada de acordo com o mesmo. Tal teste de conformidade foi implementado no decorrer deste trabalho. Ele consiste basicamente na geração de 6 milhões de blocos de entrada de IDCT, aplicação destes em uma IDCT de referência e na IDCT a ser validada e comparação dos resultados. A construção desta IDCT de referência também é especificada pelo padrão.

Aplicou-se este teste na IDCT original da libmpeg2 para confirmar se esta era realmente compatível com o padrão e como uma forma de validar o teste desenvolvido. Como era de se esperar a IDCT original da libmpeg2 passou no teste. Testou-se então a implementação padrão da IDCT provida pela API, e esta também demonstrou-se de acordo com os requisitos do padrão.

Como teste de integração da IDCT na libmpeg2 executou-se a libmpeg2 adaptada e a original, sobre os mesmos vídeos, imprimindo as saídas das IDCTs e compararando os resultados. Neste caso considerou-se a saída da libmpeg2 original como a referência. Também neste caso, como era de se esperar, as diferenças entre os resultados se encaixam nos requisitos do padrão MPEG-2. Para executar a libmpeg2, utilizou-se de um programa exemplo que vem com a mesma. Este programa tem como entrada um arquivo de vídeo em MPEG-2, e sua saída são os arquivos de imagem não compactada, que formam os vários quadros do vídeo decodificado.

4.2.2. Análise de desempenho

A implementação de IDCT provida pela API foi comparada com a IDCT original da libmpeg2 com relação a tempo médio de computação de um determinado número de blocos.

Estes blocos foram gerados pelo mesmo mecanismo empregado no teste de conformidade da IDCT (descrito acima).

Em uma execução com 60000 (sessenta mil) blocos distintos, o tempo médio de execução das IDCTs foi o seguinte:

- IDCT libmpeg2: 869 nanosegundos (ns)
- IDCT API: 1326 nanosegundos (ns)

A diferença entre os tempos de execução é de 457 nanosegundos. A relação tempo médio execução IDCT API / tempo médio execução IDCT libmpeg2, ou seja, 1326 ns / 869 ns, é de aproximadamente 1.5. Isto significa que a IDCT da libmpeg2 se apresentou cerca de 1.5 vezes mais rápida do que a implementação de IDCT provida pela API. No procedimento descrito acima, desejou-se verificar o desempenho das IDCTs (libmpeg2 e API) de forma isolada. Portanto o tempo de mapeamento de blocos de entrada de IDCT formato libmpeg2 para blocos IDCT convencionais não foi considerado. Repetindo-se este procedimento, considerando o tempo de mapeamento, a diferença entre os tempos de execução sobe para cerca de 2.5 vezes.

Atualmente nem o decodificador original nem o modificado se encontram integrados a um player de vídeo. Esta integração seria interessante para se avaliar de forma visual qual a representatividade desta diferença de 2.5 vezes entre o tempo de computação das duas IDCTs. Entretanto realizou-se um pequeno experimento que forneceu uma indicação aproximada da representatividade desta diferença. O experimento foi o seguinte: executou-se alguns vídeos MPEG-2 no player VLC [vlc 2008] o qual utiliza a libmpeg2. Mediu-se a carga da CPU durante a execução do VLC utilizando-se o programa TOP [top 2008]. Em média a carga da CPU fica abaixo de 5%, o que representa de forma aproximada a carga utilizando-se o decodificador original. Considerando uma relação diretamente proporcional e linear entre o tempo de execução da IDCT e a carga média da CPU, a utilização do decodificador modificado no VLC resultaria em uma carga de 12.5% (2.5 x 5%). Como 12.5% é uma carga relativamente baixa de utilização de CPU, pode-se inferir que a diferença de 2.5 vezes entre os tempos médios de execução das IDCTs é aceitável.

4.3. Compensação de Movimento

A compensação de movimento (Motion Compensation - MC), é um importante estágio do processo de decodificação MPEG-2, pois é neste momento em que a redundância temporal é tratada, e as imagens codificadas em função de outras, reconstruídas. Esta importância se reflete na libmpeg2, pois além de uma implementação genérica escrita em C, existem também implementações escritas em assembly, otimizadas para plataformas específicas.

4.3.1. Compensação de Movimento na libmpeg2

A compensação de movimento, na libmpeg2, está organizada da seguinte maneira:

No nível mais alto existe a macro MOTION_CALL, utilizada no processamento dos macroblocos que fazem parte de um determinado slice. Está macro executará a rotina de MC desejada de acordo com a estrutura da imagem e os modos de macrobloco.

As funções que MOTION_CALL poderá chamar são específicas para frames ou fields, para o modo de predição a ser utilizado, e para cada tipo de macrobloco. Seguem alguns exemplos:

- *motion_fr_field_420* utilizada em frames que utilizam predição do modo field, em macroblocos do tipo 4:2:0
- *motion_fr_frame_422* utilizada em frames que utilizam predição do modo frame, em macroblocos do tipo 4:2:2
- *motion_fi_16x8_444* utilizada em frames que utilizam predição do modo 16x8 MC, em macroblocos do tipo 4:4:4

As funções do tipo *motion_fr_** e *motion_fi_** , em suas implementações utilizam macros que são expandidas em operações relacionadas ao processamento dos vetores de movimento. Estas macros possuem nomes como MOTION_420, MOTION_FIELD_444, MOTION_DMV_422, ou seja, estão relacionadas ao modo de predição utilizado (field, frame, dual-prime, 16x8 MC) e ao tipo do macrobloco (4:2:0, 4:2:2, 4:4:4).

Existem funções que atuam na etapa de formação das predições da MC, operando sobre blocos de referência e vetores de movimento já decodificados. Estas funções são implementadas basicamente com o uso de laços e operações de deslocamento (shifts). Tais funções possuem implementações genéricas escritas em C e versões escritas em assembly, otimizadas para plataformas específicas. Alguns exemplos destas funções são:

- *MC_put_x_16_c*
  Realiza operação de atribuição (put), atuando na componente horizontal do macrobloco (x) e sobre toda sua extensão (16 pixels). A terminação "_c"indica que trata-se da versão da função escrita em C.
- *MC_put_x_16_mmx*
  Versão assembly utilizando instruções MMX da função acima.
- *MC_avg_x_16_c*
  Semelhante as anteriores, porém este realiza uma média (average - avg), utilizada no calculo das meias-amostras.

4.3.2. Pontos em que a API pode ser utilizada

A API DSP deverá prover além de funcionalidades de processamento digital de sinais (DSP) como IDCT, FFT, DWT, entre outras; primitivas de processadores digitais de sinais (DSP-HW) como acumuladores de multiplicação (MAC), multiplicações de matrizes, laços em hardware, e deslocadores em barra (barrel shifters).

Como já foi afirmado em 4.3.1, as funções da libmpeg2 que lidam com a formação das predições fazem uso de laços e operações de deslocamento em sua implementações. A implementação da função *MC_put_x_16_c* ilustra esta afirmação:

```
// From motion_comp.c with macros extended
static void MC_avg_xy_16_c (uint8_t * dest, const uint8_t * ref,
                                const int stride, int height)
{
    do
    {
        dest[0] = (((((ref[0]+ref[0 +1]+(ref+stride)[0]+
```

```
                        (ref+stride)[0 +1]+2)>>2))+dest[0]+1)>>1);
        dest[1] = (((((ref[1]+ref[1 +1]+(ref+stride)[1]+
                        (ref+stride)[1 +1]+2)>>2))+dest[1]+1)>>1);

// dest[2] a dest[14] de forma semelhante, ...
        dest[15] = (((((ref[15]+ref[15 +1]+(ref+stride)[15]+
                         (ref+stride)[15 +1]+2)>>2))+dest[15]+1)>>1);
        ref += stride;
        dest += stride;
    } while (--height);
}
```

Estas funções que lidam com a formação das predições representam pontos onde a utilização da API é possível. As primitivas da API a serem utilizadas, neste caso, seriam os deslocadores em barra e os laços em hardware. Os deslocadores em barra permitem a realização das operações de deslocamento em um único ciclo de relógio. E laços em hardware, poderiam substituir os laços em software presentes nestas funções.

No presente momento deslocadores em barra e os laços em hardware ainda não são providos pela API. Porém, quando eles estiverem disponíveis, a estratégia para utiliza-los a partir da libmpeg2 será a mesma que foi usada para substituir a IDCT, isto é, a alteração da versão em C das devidas funções. Assim como no caso da IDCT não haverá necessidade de alterar as versões assembly das funções, uma vez que as otimizações ficarão a cargo das implementações providas pela API.

5. Conclusões

É desejável que implementações em software de codecs sejam portáveis para as diversas plataformas de hardware existentes, mantendo um desempenho satisfatório nas diversas plataformas. O processamento de sinais de áudio e vídeo costuma consumir uma parcela significativa no tempo de processamento de tais codecs. Em virtude disto diversas implementações de codecs possuem otimizações para processamento de sinais escritas em linguagem assembly, focadas em plataformas específicas. Com a disponibilidade de uma API que concentre em si funcionalidades de DSP e DSP-HW, provendo implementações para diversas arquiteturas distintas, a construção ou adaptação de um codec para atender o requisito de ser multiplataforma se torna facilitada. Este projeto escolheu um decodificador de vídeo padrão MPEG-2 para ser adaptado para tal API. Esta API é o trabalho do mestrando Danillo Moura Santos, co-orientador deste projeto. Os estágios do decodificador trabalhados foram a transformada inversa discreta de coseno (IDCT) e a compensação de movimento (MC). O decodificador escolhido para ser adaptado foi a libmpeg2 , que possui código fonte disponível e passível de ser alterado e apresenta conformidade com o padrão MPEG-2.

Pesquisou-se na libmpeg2 onde a IDCT era utilizada e como era utilizada. A partir daí modificou-se a libmpeg2 para utilizar a IDCT provida pela API DSP. A compensação de movimento também foi analisada e gerou-se apontamentos de onde a utilização da API é possível.

Como a IDCT da libmpeg2 exige que sua entrada esteja em um formato específico, diferente do esperado para uma IDCT convencional e como a API provê uma IDCT que funciona com entradas convencionais, foi necessário realizar um mapeamento de “en-

tradas formato libmpeg2" para "entradas convencionais", para poder fazer uso da IDCT fornecida pela API dentro da libmpeg2. Aplicou-se o teste de conformidade descrito no padrão MPEG-2 parte 4 [ISO/IEC 1998] na implementação original da IDCT da libmpeg2 e na implementação provida pela API. Ambas as IDCTs passaram no testes. Comparou-se as saídas de IDCT geradas na decodificação de vídeos do decodificador original e do decodificador modificado, analisando-as sob os requisitos do mesmo teste de conformidade aplicado nas IDCTs de forma isolada. Constatou-se que os resultados também estão de acordo com os requisitos do teste. Fez-se também uma análise de desempenho comparando a IDCT original com a provida pela API. Também neste quesito os resultados foram satisfatórios.

Foram identificados os pontos da libmpeg2 relativos a compensação de movimento que podem usar a API DSP. Atualmente as primitivas da API a serem utilizadas pela MC ainda não estão disponíveis já que a mesma se encontra em desenvolvimento. Porém, quando elas estiverem disponíveis, a estratégia para utiliza-los a partir da libmpeg2 será a mesma que foi usada para substituir a IDCT.

Através deste trabalho pode ser demonstrada a aplicabilidade de uma API DSP multiplataforma em um problema real, que é a decodificação de vídeo digital. Para tanto um decodificador padrão MPEG-2 foi adaptado à API, se beneficiando de todas as características providas pela mesma. Um outro fruto deste trabalho foi a implementação do teste de conformidade MPEG-2 parte 4 no que diz respeito a teste de IDCT. Como parte deste teste foram implementadas também uma FDCT e uma IDCT de referência. Este teste pode ser aplicado a cada nova implementação de IDCT que se desejar verificar conformidade com o padrão MPEG-2.

Referências

(2008a). libmpeg2 - a free mpeg-2 video stream decoder.

(2008b). libmpeg2-devel - mailing list archive.

(2008). Mpeg home page.

(2008). top (unix).

(2008). Vlc media player - overview.

IEEE (1990-1991). *IEEE Standard Specifications for the Implementations of 8x8 Inverse Discrete Cosine Transform*.

ISO/IEC (1995). *International Standard ISO/IEC 13818-2 (Video), Recommendation ITU-T H.262*.

ISO/IEC (1998). *International Standard ISO/IEC 13818-4 (Conformance testing)*.