Adaptação de um Decodificador MPEG-2 a uma API DSP Multiplataforma

Mateus Krepsky Ludwich, Danillo Moura Santos, Antônio Augusto Fröhlich
Laboratório de Integração de Software e Hardware – LISHA
Departamento de Informática e Estatística – INE
Universidade Federal de Santa Catarina – UFSC
Caixa Postal 476 – CEP 88.040-900 – Florianópolis – SC – Brasil
{mateus,danillo,guto}@lisha.ufsc.br

ABSTRACT
Video Codecs are commonly written with a specific platform in mind, resulting in a source code highly platform dependent, difficult to be applied in other architecture. The wide number of platforms to embedded video processing existing nowdays makes necessary the portability of codecs to different platforms without significant performance losses. Some stages of video decoders, as IDCT and Movement Compensation, require higher computational efforts, therefore, developers optimize these to the target platform Assembly language, resulting in the portability problem. This problem is addressed in this work with the use of a cross-platform API to support these decoding stages. The API proposed supports DSP features and some hardware DSP extensions (DSP-HW) as MACs and optimized IDCT IPs (hardware blocks). The LIBMPEG2 was ported to this API and some conformance and performance tests were done, comparing the new decoder performance to the existing one and the new decoder conformance to MPEG standard. An analysis of how to support Movement Compensation in the API is also showed, tracing the path to future works.

Categories and Subject Descriptors
I.4 [**Image Processing And Computer Vision**]: Compression (Coding)

General Terms
Algorithms, Design

Keywords
Video decoding, MPEG, DSP, API, LIBMPEG2

1. INTRODUÇÃO
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 de baixo nível de arquiteturas específicas. O lado ruim de tal abordagem é que a portabilidade dos codecs é prejudicada. Para solucionar este problema, utilizou-se uma API que concentra em si funcionalidades de processamento digital de sinal (DSP) e primitivas de processadores digitais de sinal (DSP-HW). Esta API tem como objetivo prover suporte para o desenvolvimento de aplicações portáveis, mantendo um desempenho satisfatório em diferentes arquiteturas computacionais. Como estudo de caso, aplicou-se esta API na biblioteca LIBMPEG2 [1] que implementa um decodificador MPEG-2.

A LIBMPEG2 possui um esquema nativo para prover portabilidade. Neste esquema os estágios que mais realizam processamento (i.e. IDCT e MC) fazem uso de diversos ponteiros de função. Estes ponteiros podem ser associados a funções genéricas, escritas em linguagem C, ou a funções otimizadas, escritas em linguagem de montagem de arquiteturas específicas.

A API DSP provê assinaturas de métodos/funções C++, entretanto a implementação de cada método pode ser realizada em C ou C++, em linguagem de montagem otimizada para arquiteturas específicas, ou ainda em hardware (através do uso de lógica programável por exemplo). Como a aplicação apenas "enxergará" métodos C++, é provida uma portabilidade de forma transparente.

Modificou-se a LIBMPEG2 para utilizar a IDCT provida pela API. Confirmou-se a corretude das alterações efetuadas, e também realizou-se uma análise de desempenho comparando o decodificador original com o modificado. A aplicabilidade da API também é mostrada para a MC, sendo apresentados os pontos deste estágio que podem fazer uso da API.

2. API DSP
O contínuo avanço da tecnologia de fabricação de circuitos integrados VLSI, como a maioria dos ASICs, proporciona o emprego de processamento digital de sinais em diversas áreas. Atualmente diversos sistemas embarcados fazem uso de algoritmos DSP, porém estes sistemas sofrem com a escassez de recursos, o que leva os projetistas ao *trade-off* entre poder de processamento e recursos computacionais existentes. O baixo consumo de energia é importante em sistemas alimentados por bateria, portanto, aliado ao fato das restrições de recursos, as implementações de algoritmos DSP devem ser adaptadas para sistemas embarcados, levando em consideração os requisitos da aplicação e também a eficiência no consumo de energia e os poucos recursos disponíveis. Esta adaptação na implementação dos algoritmos DSP para sistemas embarcados pode ser facilitada com o uso de uma metodologia de projeto que guie o desenvolvedor durante a implementação do sistema.

Diversas características arquiteturais dos DSP-HWs, como instruções dedicadas e modelos de memória específicos,

impedem que o desenvolvimento de aplicações em linguagem de alto nível gere código eficiente para estas arquiteturas. Por isso, ainda hoje, muitas aplicações são desenvolvidas em linguagem de baixo nível, o que dificulta que estas sejam portadas para outras arquiteturas. Este é o ponto crucial deste trabalho, permitir o desenvolvimento de aplicações DSP em linguagem de alto nível de forma que através do uso de uma API o código gerado seja satisfatório (eficiente) e possa ser portado para várias arquiteturas. Dentre estas arquiteturas pode-se citar processadores digitais de sinais com instruções dedicadas, processadores de propósito geral e sistemas dedicados desenvolvidos em FPGA.

A proposta da API é tornar aplicações de DSP portáveis para diferentes arquiteturas. A portabilidade é interessante para desenvolvedores de produtos, pois estes podem implementar o seu conhecimento (software da aplicação) em diferentes processadores, analisando o melhor custo benefício. Além da redução de custo através da exploração de diferentes soluções, o uso de uma API homogênea para diferentes arquiteturas permite a criação de famílias de produtos com características diferentes, sem que seja necessário uma reengenharia de todo produto a cada nova versão (decorrente da troca de arquitetura). Um dos requisitos fundamentais deste tipo de aplicação é o cumprimento de requisitos de Tempo Real, os quais podem ser também suportados pela API, auxiliando o desenvolvedor da aplicação durante o projeto.

No desenvolvimento da API estão sendo empregados os conceitos da metodologia de desenvolvimento de sistemas orientados a aplicação, do inglês *Application-Oriented System Design* (AOSD). O EPOS, sistema operacional com o qual a API estará integrada, foi desenvolvido seguindo este metodologia. A AOSD[4] faz uso de algumas técnicas de engenharia de software para permitir o desenvolvimento de sistemas embarcados eficientes, algumas destas técnicas são: orientação a aspectos, desenvolvimento baseado em famílias e metaprogramação estática.

As funcionalidades da API são providas através de métodos C++. Dentre os métodos que a API pode prover estão transformadas conhecidas no domínio do DSP, como DCT, IDCT, DFT (FFT), filtros, e primitivas de DSP-HW, como multiplicadores acumuladores (MAC), deslocadores em barra (barrel shifter) e operações aritméticas diversas em ponto fixo.

A seguir, no estudo de caso realizado, é mostrado a função para realização de IDCT, provida pela API, assim como trechos de programa ilustrando a utilização da mesma.

3. ESTUDO DE CASO

O decodificador MPEG-2 que foi adaptado para trabalhar com a API foi o decodificador implementado pela biblioteca LIBMPEG2. A LIBMPEG2 trata-se de um software livre e é uma biblioteca amplamente utilizada e depurada. Em [1] 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 linguagem de montagem (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 linguagem de montagem são a transformada inversa discreta do cosseno, do inglês Inverse Discrete Cosine Transform (IDCT) e a compensação de movimento, do inglês Motion Compensation (MC). As plataformas para as quais atualmente existem otimizações em linguagem de montagem são x86 (instruções MMX) e Power PC (instruções AltiVec).

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.

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 imagens MPEG-2. Estes ponteiros podem ser associados a funções genéricas, escritas em linguagem C, ou a funções otimizadas, escritas em linguagem de montagem.

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 linguagem de montagem, pois as implementações da IDCT da própria API é que proverão versões em linguagem de montagem, otimizadas para plataformas específicas. Aqui, é mostrado como a função mpeg2_idct_copy_c foi alterada. Não há necessidade de se apresentar a alteração da função mpeg2_idct_add_c, pois ela aconteceu de forma muito semelhante a modificação de mpeg2_idct_copy_c.

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 construído. A implementação original pode ser vista na figura 1.
Nesta é aplicada uma IDCT unidimensional nas linhas do bloco e, em seguida, outra IDCT também unidimensional 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 [1].

A versão alterada de mpeg2_idct_copy_c, utilizando-se a API, pode ser observada na figura 2.

Como a função *idct2D*, provida pela API é escrita em C++, e a LIBMPEG2 em C, é necessário a utilização da função

```
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);
    // ...
}
```

Figure 1: mpeg2_idct_copy_c - Original

```
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);
    // ...
}
```

Figure 2: mpeg2_idct_copy_c - Adaptada

wrapper *c_idct2D*.

Verificou-se 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. Como a função de IDCT provida pela API necessita de uma entrada convencional utilizou-se a função *libmpeg2ToNormal* para desfazer a formatação de entrada realizada pela LIBMPEG2.

Como mencionado, a API DSP é quem proverá implementações em linguagem de montagem, otimizadas para plataformas específicas para a função *idct2D*. Devido a isto as otimizações de cada método/função e a portabilidade dos mesmos, 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.

3.1 Teste de conformidade

O padrão MPEG-2 parte 4 (testes de conformidade)[5] 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 comparando 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.

3.2 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 [3] o qual utiliza a LIBMPEG2. Mediu-se a carga da CPU durante a execução do VLC utilizando-se o programa TOP. 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.

3.3 Aplicabilidade da API

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. Uma etapa fundamental na MC, consiste na formação de predições e preditores, que posteriormente serão adicionados à imagens chaves, reconstruindo imagens que foram codificadas em função de outras.

As funções da LIBMPEG2 que lidam com a formação das predições fazem uso de laços e operações de deslocamento em sua implementações. As primitivas da API que podem ser utilizadas, neste caso, são 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 utilizá-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 em linguagem de montagem das funções, uma vez que as otimizações ficarão a cargo das implementações providas pela API.

4. TRABALHOS RELACIONADOS

VSIPL é uma especificação de domínio público para processamento de imagens e sinais [6]. Define uma API em C que é razoavelmente difundida, e possui implementações de diferentes *Vendors* para diferentes arquiteturas. Existe também uma versão deste padrão baseada na linguagem C++, cujo intuito é prover um maior nível de abstração na modelagem de aplicações DSP, chamado VSIPL++. A API usada no presente trabalho suportará uma versão da VSIPL++ criada para sistemas embarcados chamada VSIPL++ Core Lite.

Multimedia Macros (MMM) [7], é uma biblioteca de macros do pré-processador da linguagem C voltada para a construção de aplicações DSP e multimídia. Uma aplicação escrita utilizando a MMM utiliza suas macros, que são expandidas para instruções em linguagem de montagem de diversas arquiteturas. Possui implementações para diversos DSP-HW e para processadores de uso geral (General Purpose Processor - GPP) com extensões multimídia (e.g. Pentium/SSE, PowerPC/AltiVec).

OpenMax é uma especificação que define é um conjunto de PIs (Programming Interfaces) [2] Ela é dividida em três camadas: Application, Integration e Development. Esta terceira camada, Development, estabelece primitivas como IDCT, VLC, Zig-Zag, FIR, entre outras relacionadas com DSP e processamento de áudio e vídeo.

5. CONCLUSÕES

Através deste trabalho foi demonstrada a utilização de uma API DSP como meio de solucionar o problema de portabilidade enfrentado pelos codecs de vídeo, sem maiores perdas de desempenho. Esta demonstração foi realizada adaptando-se a biblioteca LIBMPEG2, que implementa um decodificador MPEG-2, para trabalhar com a API. Os estágios do decodificador trabalhados foram a transformada inversa discreta de coseno (IDCT) e a compensação de movimento (MC).

Modificou-se a LIBMPEG2 para utilizar a IDCT provida pela API. A corretude das alterações efetuadas foi confirmada com a aplicação do teste de conformidade descrito no padrão MPEG-2 parte 4 [5]. Fez-se também uma análise de desempenho comparando a IDCT original com a provida pela API. Neste quesito a IDCT da API mostrou-se aproximadamente 2.5 vezes mais lenta que a IDCT original.

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 utilizá-los a partir da LIBMPEG2 será a mesma que foi usada para substituir a IDCT.

6. REFERENCES

[1] libmpeg2 - a free mpeg-2 video stream decoder, 2008.

[2] The standard for media library portability, 2008.

[3] Vlc media player - overview, 2008.

[4] A. A. Fröhlich. *Application-Oriented Operating Systems*. PhD thesis, Sankt Augustin: GMD - Forschungszentrum Informationstechnik GmbH, Sankt Augustin, Germany, 2001.

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

[6] R. S. Janka, R. Judd, J. Lebak, M. Richards, and D. Campbell. VSIPL: an object-based open standard API for vector, signal, and image processing. In *ICASSP 01: Proceedings of the Acoustics, Speech, and Signal Processing, 200. on IEEE International Conference*, pages 949–952, Washington, DC, USA, 2001. IEEE Computer Society.

[7] J. C. Rojas and M. Leeser. Programming portable optimized multimedia applications. In *MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia*, pages 291–294, New York, NY, USA, 2003. ACM.