Investigação sobre o Uso de Multiprogramação Leve como Alternativa para a Estimação de Movimento na Compressão de Vídeos de Alta Resolução

João Alberto Vortmann, Rafael Petry, Guilherme Corrêa, Fabiane Rediess, Luciano Agostini, Gerson Geraldo H. Cavalheiro

Universidade Federal de Pelotas
Departamento de Informática
Bacharelado em Ciência da Computação
{jvortmann_ifm, rpetry_ifm, gcorrea_ifm, frediess_ifm, agostini, gerson.cavalheiro}@ufpel.edu.br

Resumo

A popularização do uso de vídeo digital nas mais diferentes mídias tornou o processo de codificação de vídeo fundamental por permitir a redução da quantidade de dados a serem transmitidos e armazenados sem perda significativa de qualidade. Neste processo, a etapa de estimação de movimento é a que requer maior tempo de processamento, existindo diferentes algoritmos para implementa-la. Este trabalho apresenta o uso de técnicas e ferramentas de multiprogramação leve (OpenMP) em arquiteturas multi-core para a realização da estimação de movimento. Os resultados das implementações, explorando o paralelismo de diversas formas, são apresentados e discutidos. Os experimentos apontam que o algoritmo Full Search ofereceu os melhores ganhos de desempenho com tempo de processamento, 54.54% menor que a versão seqüencial.

1. Introdução

O grande interesse da indústria na codificação de vídeo digital é reflexo da popularização das diversas aplicações de vídeo em diferentes mídias, como telefones celulares, DVD *players*, câmeras e mesmo no Sistema Brasileiro de Televisão Digital (SBTVD). A codificação é essencial para que a transmissão em tempo real seja possível. Em conseqüência, diversos padrões de codificações foram propostos, entre os quais destaca-se o padrão H.264/AVC [1], que atinge a maior taxa de compressão, chegando ao dobro em relação a padrões anteriores.

Uma seqüência de vídeo consiste de uma série de quadros que são reproduzidos em uma taxa elevada (para tempo real são necessários 24 a 30 quadros por segundo). É natural que se note uma grande semelhança entre quadros vizinhos temporalmente. A estimação de movimento explora a redundância temporal, mapeando blocos (pequenas matrizes de *pixels*) de quadros vizinhos através de vetores de movimento. Após a estimação, o quadro alvo da codificação pode ser reconstruído utilizando apenas informações dos quadros anteriormente codificados e os vetores de movimento gerados. Esta é a etapa mais complexa do processo de compressão de vídeo, sendo responsável pela maior parte do processamento de digitalização de vídeos.

Para a realização da estimação de movimento, os quadros do vídeo são divididos em blocos e uma área de pesquisa é determinada no quadro de referência para cada bloco do quadro que está sendo analisado. A área de pesquisa geralmente compreende uma fração do quadro, visando diminuir a complexidade computacional, mas pode conter o quadro inteiro em determinados casos. Em seguida, um algoritmo de busca determina a maneira como esse bloco se desloca dentro da área de pesquisa para que a melhor relação de distorção seja encontrada.

Para obter, em *software*, uma implementação que ofereça tempo real, esta etapa deve explorar técnicas e recursos de programação eficientes. Este trabalho propõe o uso de arquiteturas *multi-core* e programação em OpenMP [2] para implementar o algoritmo *Full Search* e a técnica de *Diamond Search* [3] de estimação de movimento. São analisados desempenho das versões concorrentes e seqüenciais de ambas implementações.

Este trabalho está organizado como segue. A Seção 2 introduz o processo de estimação de movimento e apresenta o algoritmo *Full Search* e a técnica *Diamond Search*. O estudo de caso é apresentado na Seção 3, onde são discutidas as implementações realizadas para ambos algoritmos. A Seção 4 apresenta uma avaliação dos resultados obtidos e a Seção 5 conclui o trabalho.

2. Estimação de movimento

Um vídeo digital é formado por uma seqüência de quadros estáticos, que devem ser apresentados a uma taxa de 24 a 30 quadros por segundo para se ter a

impressão de tempo real. É natural que entre quadros vizinhos existam informações semelhantes. A estimação de movimento (EM) é a operação de um codificador que busca minimizar a redundância temporal entre quadros vizinhos. Para tanto, a EM utiliza um quadro previamente codificado como quadro de referência, e divide o quadro atual (alvo da codificação) em blocos. Para cada bloco, o algoritmo irá procurar o bloco mais semelhante no quadro de referência utilizando uma função de similaridade. Ao final da busca, é gerado um vetor de movimento que indica o bloco mais semelhante no quadro de referência. Essa informação é utilizada pelo decodificador para remontar a imagem a partir do quadro de referência. Com isso não é necessário enviar todo o novo quadro, sendo reaproveitados blocos já recebidos.

Para diminuir o tempo de processamento requerido na etapa de EM, a busca pelo bloco mais semelhante é geralmente feita em uma região limitada denominada área de pesquisa, como mostrado na Figura 1. Também podem ser empregados algoritmos que implementam heurísticas para diminuir o número de operações necessárias para se determinar o vetor de movimento.

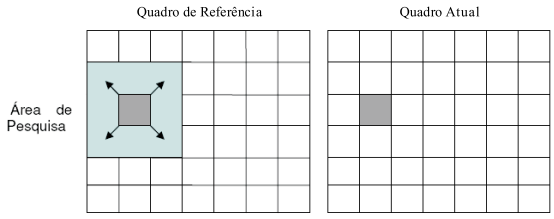

Figura 1. Determinação do vetor de movimento para um bloco.

2.1. O algoritmo Full Search

O *Full Search* [4] é um algoritmo popular de estimação de movimento que procura o melhor casamento de um determinado bloco, realizando a sua comparação com todos os blocos possíveis dentro da área de pesquisa do quadro de referência. Para definir a similaridade entre os dois blocos e, assim, definir o melhor casamento, é aplicado um critério de distorção. O algoritmo procura, então, com base neste critério de distorção, o bloco no quadro de referência que possui a maior similaridade com o bloco atual. A pesquisa é feita deslocando o bloco *pixel* a *pixel* a partir do canto esquerdo superior da imagem até o canto direito inferior. Depois de encontrado o melhor bloco, um vetor de movimento é gerado para referenciar o deslocamento do bloco atual com relação ao bloco de maior similaridade.

Um critério de distorção bastante difundido na literatura é o SAD (*Sum of Absolute Differences*) [3], que calcula a distorção entre as regiões comparadas como o somatório das diferenças absolutas para cada ponto do bloco atual e do bloco candidato da área de pesquisa. Outro critério de distorção, o MSE (*Mean Square Error*) [3], avalia a semelhança entre os blocos através do valor médio quadrático da diferença entre os valores dos pixels do bloco atual e do bloco candidato da área de pesquisa.

Neste trabalho, o critério de distorção aplicado foi o SAD. Portanto, para cada bloco candidato é feito o cálculo do seu SAD e o bloco selecionado será o que apresentar menor SAD. O critério MSE não é aplicado para a escolha do bloco, mas é utilizado no cálculo do PSNR (*Peak Signal-to-Noise Ratio*) [5] que é um critério também bastante difundido na literatura utilizado para avaliar a qualidade subjetiva da estimação.

Como o *Full Search* aplica a função de similaridade a todos os *pixels* de todos os blocos candidatos em relação ao bloco atual, o vetor de movimento gerado é sempre o melhor possível e, conseqüentemente, a melhor qualidade de compressão é obtida [6]. Logo, o *Full Search* é considerado o único algoritmo ótimo para estimação de movimento. Esta grande complexidade computacional demanda um grande tempo de execução para vídeos com grandes áreas de pesquisa.

2.2. A técnica Diamond Search

O algoritmo *Diamond Search* utiliza dois padrões (*patterns*) de busca em forma de diamante e é realizado em duas fases. O *Large Diamond Search Pattern* (LDSP) é utilizado nas iterações da etapa inicial e o *Small Diamond Search Pattern* (SDSP) na etapa final de refinamento. Esses padrões são ilustrados na Figura 2.

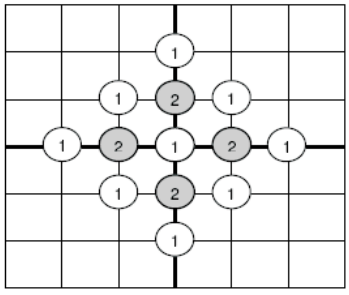

Figura 2. Padrões LDSP (1) e SDSP (2).

O algoritmo inicia aplicando o LDSP, blocos identificados por (1), no centro da área de pesquisa. Caso o bloco de menor erro (maior grau de similaridade) esteja localizado em uma posição que não seja a central deste diamante, esse bloco é definido como o novo centro, e o algoritmo repete a fase LDSP.

Existem duas possibilidades de iteração nessa etapa: por aresta e por vértice. Elas são ilustradas na Figura 3.

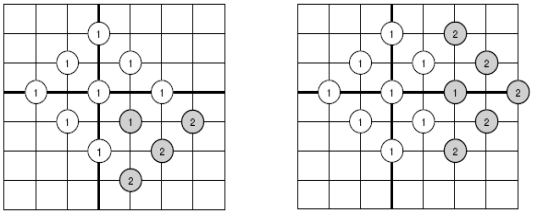

Figura 3. Busca por uma aresta (lado esquerdo) e busca por um vértice (lado direito).

Quando a posição central for a que possuir o menor erro, é aplicado o SDSP para refinar o resultado. Nessa etapa são verificados quatro novos blocos que são comparados com o centro, e o melhor resultado é escolhido.

Uma análise de diversas técnicas de EM foi apresentada em [6], abrangendo tanto o *Full Search*, apresentado anteriormente, quanto o *Diamond Search*. Ademais, mostra que o *Diamond Search* permitiu atingir um resultado próximo ao *Full Search* requerendo menos operações, diminuindo a quantidade de recursos e de tempo para a etapa de EM, sem perda significativa de qualidade.

3. Estudo de caso

Esta seção apresenta o modelo das implementações paralelas realizadas para o algoritmo *Full Search* e *Diamont Search*. Dado a inerente natureza de paralelismo de dados manifestada pela aplicação, a opção por utilizar OpenMP mostra-se adequada. Diferentes estratégias de exploração do paralelismo foram propostas, particularmente para o algoritmo *Full Search* que apresenta um número maior de variações. As implementações foram realizadas em C.

A fim de diminuir o tempo de execução de ambos algoritmos, viabilizando sua execução em tempo real em software, foi realizado um amplo estudo sobre técnicas e ferramentas de multiprogramação leve como OpenMP. Em seguida foram adicionadas diretivas de compilação OpenMP ao código original escrito em linguagem C e procedeu-se as análises.

Seis versões concorrentes foram elaboradas para o *Full Search*. A primeira (FS1) realiza o processamento dos quadros de vídeo de forma paralela. Todos os quadros são processados em uma ordem aleatória, de acordo com o sistema de escalonamento de *threads*, e os resultados são escritos em arquivo ao final de todo o processamento do vídeo. Nesta versão, a diretiva #pragma omp parallel for foi utilizada no laço que itera sobre os quadros do vídeo.

Na segunda versão (FS2), o processamento de quadros também é paralelo. Contudo, a escrita dos resultados em arquivo é realizada de forma ordenada, embora o processamento ocorra em ordem aleatória. Para tanto, além da diretiva utilizada na FS1, a diretiva #pragma omp ordered foi utilizada dentro do laço de repetição para que a escrita se dê na ordem em que ocorreria caso fosse executada em um laço seqüencial.

A terceira versão (FS3) realiza o processamento de quadros em ordem seqüencial. Contudo, a seleção do bloco que é usado como referência para o cálculo de SAD é realizada em paralelo. Para isso, duas diretivas #pragma omp parallel for foram inseridas no código: uma no laço mais externo, que itera sobre as colunas do quadro, e outra no laço mais interno, que itera sobre as linhas.

Na quarta versão (FS4), a seleção do bloco atual é feita seqüencialmente, mas a busca pelo melhor quadro (menor SAD) é feita em paralelo. Ou seja, com base em um bloco atual, os blocos restantes são analisados concorrentemente, em ordem aleatória. No entanto, como o a escolha do menor SAD requer uma comparação entre os SADs já calculados, o teste para escolhê-lo é realizado em regime de exclusão mútua. Ou seja, todas as *threads* executarão o teste, mas nunca ao mesmo tempo. Para que isso aconteça, a diretiva #pragma omp critical (teste) foi inserida antes do teste.

Os cálculos realizados para obtenção do SAD e do MSE são as operações realizadas em paralelo na quinta versão (FS5). Como mencionado, o SAD é o somatório das diferenças absolutas para cada ponto do bloco atual e do bloco candidato da área de pesquisa. Assim, como a soma é comutativa, a diferença absoluta entre cada par de pontos do bloco pode ser feita em ordem aleatória. Apesar disso, como cada *thread* possui uma cópia distinta do SAD calculado, é necessário o uso da cláusula reduction (+ : sad) após o uso da diretiva #pragma omp parallel for para que o valor do SAD seja atualizado corretamente no *thread master*. O cálculo de MSE é realizado de forma análoga ao cálculo de SAD. Todas as concorrências exploradas nas versões anteriores foram reunidas na sexta versão do *Full Search* (FS6).

A técnica *Diamond Search* foi implementada na linguagem C em duas versões, seqüencial e concorrente com OpenMP. A versão OpenMP utiliza paralelismo por quadro, ou seja, cada *thread* é responsável pelo processamento de um quadro. Para tanto, a diretiva #pragma omp parallel for foi utilizada no laço que itera sobre os quadros do vídeo.

O objetivo de comparar estas duas estratégias é comparar o ganho de desempenho de suas respectivas versões paralelas em software. Esta avaliação compara a estratégia que oferece a melhor qualidade de estimação de movimento, *Full Search* [7], com a que oferece a melhor relação entre custo computacional seqüencial e qualidade de resultado, *Diamond Search* [8]. Como resultado deve-se apontar a melhor

estratégia para ser implementada em software explorando o potencial de processamento paralelo disponibilizado nas novas configurações de hardware dotadas de processadores multi-core.

4. Resultados obtidos

O desempenho das implementações descritas acima foi avaliado tendo como entrada os primeiros cem quadros de dez vídeos digitais não comprimidos em resolução SDTV (720x480 pixels). Estes vídeos foram obtidos em [9] e os respectivos primeiros quadros são apresentados na Figura 4. O SAD – soma das diferenças absolutas – foi definido como critério de similaridade e a área de busca máxima foi de 208x208 *pixels*. Os vídeos são variados, apresentando diferentes graus de movimento.

Figura 4. Primeiro quadro dos vídeos utilizados nas simulações.

Os resultados de desempenho apresentados nesta seção correspondem à média de, pelo menos, 300 execuções em um computador dedicado, não sendo verificado desvio padrão acima de 5%. O computador utilizado é um Apple Mac mini, com processador Intel Core Duo (1.6 GHz) com 4 MB de memória cache compartilhada e 512 MB de RAM sob Mac OS X, e os desempenhos foram contabilizados em termos de tempo (segundos). Os tempos relacionados as operações de entrada e saída foram contabilizados.

Cada versão do *Full Search* foi executada utilizando quadros divididos em blocos de 16 x 16 *pixels* e área de pesquisa de 46 x 46 blocos.

A versão seqüencial do *Full Search*, quando compilada com o Intel Compiler 10.1, apresentou tempo de execução de 585,89 segundos. O desvio padrão das médias dos tempos de execução ficou abaixo de 0,01%.

A Tabela 1 mostra os tempos de execução das versões concorrentes do *Full Search* utilizando 2, 4 e 8 *threads* de serviço no *pool* de execução, para o vídeo citado na seção anterior. Todas as versões concorrentes foram compiladas utilizando o Intel Compiler com a opção de compilação `openmp`. Destaca-se no resultado o aumento do tempo de execução de acordo com o aumento no número de *threads* utilizadas, isto pode ocorrer pelo fato de as *threads* estarem disputando a memória compartilhada pelos núcleos do processador.

Tabela 1. Tempos de execução (em segundos) das versões concorrentes utilizando-se 2, 4 e 8 threads no pool de execução (compiladas com o Intel Compiler).

Versão	Regiões Paralelas	Tempo de Processamento		
		2 threads	4 threads	8 threads
FS1	100	272,76	265,78	265,68
FS2	100	504, 71	516,81	523,35
FS3	135.000	266, 30	269,82	266,51
FS4	129.735.000	291, 49	325,55	345,19
FS5	32.212.160.000	1.407,92	5.447,45	6.342,9
FS6	33.341.895.100	1.123,83	12.227,82	22.738,39
FS1+FS3	135.100	264,78	264,46	263,65

Conforme mostra a tabela, as implementações com menor desempenho foram aquelas que inseriram operações de sincronização entre as regiões paralelas, FS2 e FS4, ou que reduziram muito a granulosidade da aplicação com a criação de um número muito grande de regiões paralelas, FS5 e FS6. Os melhores índices de desempenho foram obtidos com as versões FS1 e FS3, cujas estratégias de implementação são de baixa complexidade, permitindo obter uma granulosidade de paralelismo suficiente para o hardware disponível. A versão paralela do algoritmo *Full Search* foi 2,22 vezes mais eficiente que a versão seqüencial do mesmo algoritmo, tendo como base o código gerado com o compilador da Intel. Observa-se que o ganho acima de 2, número de processadores disponíveis, foi possível pela sobreposição das operações de entrada e saída com cálculo efetivo [10].

A partir dos resultados obtidos na execução de FS1 até FS6, uma nova versão do *Full Search* concorrente foi criada combinando-se as duas versões com melhor desempenho (FS1 e FS3). Os tempos de execução da combinação estão apresentados na última linha da Tabela 1. Como se pode perceber, houve um pequeno ganho quando os dois paralelismos foram combinados. Percebe-se também que, nesta versão combinada, os tempos de execução tendem a diminuir com o aumento do número de *threads* no *pool* de execução.

A execução seqüencial do *Diamond Search* levou, em média, 8.42 segundos. Tabela 2 mostra os resultados de tempo de processamento médio em segundo obtidas para as implementações do *Diamond*

Search. O número de *threads* informados corresponde ao número de *threads* de serviço de OpenMP.

Tabela 2. Tempos e taxas de processamento das implementações

	Regiões Paralelas	2 Threads	4 Threads	8 Threads
Diamond Search	100	5.51	5.16	5.05

Conforme a Tabela 2, ocorre diminuição no tempo de processamento com utilização de paralelismo por quadro. Na execução com suporte de duas *threads*, a redução no tempo chegou a 34,5%. Execuções com maior número de *threads* de execução não apresentaram ganho significativo em relação à execução com duas. Além do fato de o processador ter apenas dois cores, isso também pode ser explicado pela grande quantidade de dados acessados na memória, que faz com que as regiões paralelas disputem o acesso aos dados.

5. Conclusões

Este trabalho apresentou uma avaliação do desempenho dos algoritmo *Full Search* e *Diamond Search* para o processo de estimação de movimento.

O algoritmo *Full Search* apresenta a melhor qualidade de compressão pois realiza um busca exaustiva dentro do quadro de referência. Dentre os paralelismos analisados para o *Full Search*, o algoritmo apresentou melhores tempos de execução quando o processamento de quadros e a escolha dos blocos de referência foram realizados em paralelo, alcançando um tempo de execução 54.54% menor que sua versão seqüencial. Um ligeiro ganho em tempo de execução foi percebido ao combinarem-se as duas melhores versões paralelas.

A técnica *Diamond Search*, uma heurística para diminuição da quantidade de operações na estimação de movimento, apresenta, em sua versão seqüencial, ganho em tempo de processamento sem apresentar significativa perda de qualidade em relação à técnica *Full Search*. O experimento de implementação concorrente da técnica *Diamond Search* relatado neste trabalho, utilizando OpenMP e executando em arquitetura dual-core, é capaz de atingir um *throughput* 51,8% maior que a sua versão seqüencial.

Os resultados indicam que o uso de técnicas e ferramentas de programação concorrente, como paralelismo de dados usando OpenMP, e de arquiteturas de processadores *multi-core* podem representar um importante papel em aplicações de vídeo e, em particular, no contexto da TV Digital.

Referências

[1] JVT - Joint Video Team of ITU-T and ISO/IEC JTC 1. Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 or ISO/IEC 14496-10 AVC). 2003.

[2] OpenMP Architecture Review Board. “OpenMP Application Program Interface”. http://www.openmp.org/mp-documents/spec25.pdf. 2005.

[3] Kuhn, Peter M. Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation. Kluwer Academic Publishers, Boston. 1999.

[4] Bhaskaran, V.; Konstantinides, K. Image and Video Compression Standards: Algorithms and Architectures. Massachusetts: Kluwer Academic Publisher. 1999.

[5] Richardson, I. H.264/AVC and MPEG-4 Video Compression – Video Coding for Next-Generation Multimedia. Chichester: John Wiley and Sons. 2003.

[6] Rosa, Leandro Zanetti Paiva da, el al. Investigation of Motion Estimation Algorithms Targeting High Resolution Digital Video Compression. WebMedia 2007, the XIII Brazilian Symposium on Multimedia and the Web. 2007.

[7] Lin, C.; Leou, J. An Adaptative Fast Full Search Motion Estimation Algorithm for H.264. In: ISCAS 2005 - IEEE International Symposium Circuits and Systems. Proceedings. Kobe: IEEE, 2005, p. 1493-1496.

[8] Yi, X.; Ling, N. Rapid *Block*-matching motion estimation using modif ed diamond search algorithm. In: IEEE International Symposium on Circuits and Systems, ISCAS 2005. Volume 6, 23-26 May 2005, Page(s):5489 – 5492.

[9] VQEG. (2007) The Video Quality Experts Group Web Site. Disponível em: <www.its.bldrdoc.gov/vqeg/>. Acesso em: abr. 2007.

[10] Leslie G. Valiant: A Bridging Model for Parallel Computation. Commun. ACM 33, 1990.