Aplicação de Técnicas de Processamento Paralelo no Algoritmo *Full Search* de Estimação de Movimento

Guilherme Corrêa, Fabiane Rediess, Luciano Agostini, Gerson Cavalheiro

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

Resumo. *A compressão de vídeo tem se tornado essencial, pois permite reduzir consideravelmente a quantidade de dados a serem transmitidos e armazenados sem que haja perda significativa de qualidade. Um dos blocos de maior complexidade é o estimador de movimento, para o qual existem diversos algoritmos. O Full Search é um algoritmo que apresenta elevados tempos de execução quando codifica grandes vídeos, pois realiza uma busca completa pixel a pixel nos blocos contidos na área de pesquisa. A fim de reduzir este tempo de execução, este trabalho aplica algumas técnicas de programação concorrente utilizando diretivas OpenMP e analisa os tempos de execução obtidos.*

1. Introdução

A compressão de um vídeo é essencial para que a sua transmissão em tempo real se torne possível. Com a implantação do Sistema Brasileiro de TV Digital, diversas pesquisas vêm sendo realizadas com relação aos algoritmos a serem utilizados na etapa de compressão e codificação.

Ao se analisar um trecho de um vídeo, percebe-se que os quadros temporalmente vizinhos são muito similares. Essa característica dos vídeos produz uma grande redundância temporal, ou seja, se comparados dois quadros vizinhos pode-se perceber uma enorme quantidade de informações repetidas.

O objetivo principal da estimação de movimento é identificar a redundância temporal entre quadros vizinhos e mapeá-las através de vetores de movimento. Após a estimação, o quadro alvo da codificação pode ser reconstruído utilizando apenas as informações dos quadros anteriormente codificados e dos vetores de movimento gerados.

Para a realização da estimação de movimento, os quadros do vídeo são divididos em matrizes menores (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 pode conter o quadro inteiro, mas, normalmente, essa área é definida como uma fração do quadro para diminuir a complexidade computacional da estimação de movimento. Um algoritmo de busca determina a maneira como esse bloco se desloca dentro dessa área de pesquisa para que a melhor relação de distorção seja encontrada.

Este artigo aborda o algoritmo *Full Search* e analisa o desempenho das versões concorrentes, comparando-as com a versão seqüencial. A Seção 2 ilustra o funcionamento do algoritmo citado e a Seção 3 explica como foram implementadas as

suas versões concorrentes. A Seção 4 apresenta os resultados em termos de tempo de execução de todas as versões e a Seção 5 traça conclusões sobre os resultados.

2. Algoritmo *Full Search*

O *Full Search* [Bhaskaran 1999] 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-se o bloco de pixel em 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*) [Kuhn 1999], que calcula a distorção entre as regiões comparadas como sendo o somatório das diferenças absolutas para cada ponto do bloco atual e do bloco candidato da área de pesquisa. A função do cálculo de SAD é dada por (1).

$$\mathrm{SAD}_{x,y} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left|R_{i,j} - P_{i+x,j+y}\right| \quad (1)$$

Outro critério de distorção, o MSE (*Mean Square Error*) [Kuhn 1999], 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. A função do cálculo do MSE é dada por (2).

$$MSE_{ij} = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(R_{(ij)} - P_{(ij)})^2 \quad (2)$$

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*) [Richardson 2003] 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, ele é considerado o único algoritmo ótimo para estimação de movimento, pois o vetor de movimento gerado é sempre o melhor possível. Por essa razão, o *Full Search* possui uma grande complexidade computacional, demandando um grande tempo de execução para vídeos com grandes áreas de pesquisa.

3. Soluções Concorrentes

A fim de diminuir o tempo de execução do *Full Search*, foi realizado um estudo de aplicações de técnicas de programação concorrente. Para tanto, foram adicionadas diretivas OpenMP de compilação ao código original escrito em linguagem C.

Seis versões de códigos concorrentes foram elaboradas. 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` é 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` é 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 referencia para o cálculo de SAD é realizada em paralelo. Para isso, duas diretivas `#pragma omp parallel for` são inseridas no código: uma no laço mais externo, que caminha sobre as colunas do quadro, e outra no laço mais interno, que caminha 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)` é 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.

A sexta versão do *Full Search* (FS6) reúne todas as concorrências exploradas nas versões anteriores.

4. Resultados Obtidos

Cada versão do *Full Search* foi executada cinco vezes em um vídeo de 100 quadros, com resolução de 720 x 480 pixels. O vídeo utilizado foi o *src3_ref__625_480.yuv* [VCEG 2007]. Os quadros foram divididos em blocos de 16 x 16 pixels e a área de pesquisa utilizada foi de 46 x 46 blocos. O computador que executou o algoritmo possuiu um processador Intel Core Duo 1.6 2.0 GHz e 512 MB de memória RAM. O desvio padrão das médias dos tempos de execução ficou abaixo de 0,01%.

A versão seqüencial, quando compilada com o *Intel Compiler* 10.1, apresentou tempo de execução de 585,89 segundos. Quando o compilador utilizado foi o GCC 3.4.4, o tempo de execução foi de 686,13 segundos.

A Tabela 1 mostra os tempos de execução das versões concorrentes para o vídeo citado com 2, 4 e 8 *threads* trabalhadoras no *pool* de execução. Todas as versões concorrentes foram compiladas utilizando o *Intel Compiler* com a opção de compilação *openmp*. Um resultado importante é o aumento do tempo de execução de acordo com o aumento no número de *threads* utilizadas, isto pode se dever ao 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 Execução		
		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

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.

5. Conclusões

Este trabalho apresentou uma avaliação do desempenho do algoritmo *Full Search* de estimação de movimento. Dentre todos os paralelismos analisados isoladamente, o algoritmo apresentou melhores tempos de execução quando o processamento de quadros e a escolha dos blocos de referência foram realizados em paralelo. Um ligeiro ganho em tempo de execução foi percebido ao combinarem-se as duas melhores versões paralelas.

Referências

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

Kuhn, P. (1999) Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation. Boston: Kluwer Academic Publishers.

OpenMP. (2005) OpenMP ApplicationProgram Interface. Disponível em: <http://www.openmp.org/drupal/mp-documents/spec25.pdf>.

Richardson, I. (2002) Video Codec Design – Developing Image and Video Compression Systems. Chichester: John Wiley and Sons.

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

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