Determinação de Similaridade de Histogramas com Programação Paralela

Arthur Francisco Lorenzon[1], Alessandro Bof Oliveira[1,] Fábio Diniz Rossi[2]

[1]Universidade Federal do Pampa – UNIPAMPA
CEP: 97546-550 – Alegrete – RS – Brasil

[2]Instituto Federal Farroupilha - Campus Alegrete
RS 377 - Km 27 - Caixa Postal 118 - Alegrete - RS – Brasil

alorenzon@gmail.com, aboliveira@inf.ufrgs.br,
fdrossi@al.iffarroupilha.edu.br

Resumo. *O processamento de imagens é a área da computação que manipula imagens através de vários métodos ou técnicas de processamento voltadas às mais diversas aplicações. Alguns desse métodos ou técnicas necessitam de grande poder de processamento, e para suprir essa necessidade são utilizados hardwares específicos e/ou técnicas de paralelismo. Neste trabalho apresentamos as vantagens da paralelização com OpenMP para a determinação de similaridade de histogramas aplicado ao rastreamento de veículos.*

1. Introdução

O rastreamento de veículos constitui um problema de grande importância na área de processamento de imagens, pois encontra inúmeras aplicações como vigilância, monitoramento de tráfego (contagem e classificação de veículos) e detecção automática de eventos (como acidentes ou direção perigosa). A abordagem de rastreamento de veículos utilizando filtro de partículas tem mostrado bons resultados. Nessa abordagem uma região de interesse a ser rastreada (a região que contenha um veículo), tem associada a si uma distribuição de cores ou níveis de cinza (histograma), que posteriormente será comparada através de medidas de similaridade com distribuições de outras regiões da imagem para a predição da posição estimada da região de interesse [1].

Os métodos de filtro de partículas utilizam uma abordagem de amostragem das regiões da imagem para estimar a posição do veiculo rastreado [2]. Essa amostragem é realizada numa busca direcionada próxima a posição de onde se espera encontrar o veículo rastreado. A utilização de uma amostragem é utilizada para reduzir o custo computacional da geração e comparação de histogramas utilizados pelo método de filtro de partícula. Essa abordagem apresenta limitações quanto a precisão da estimativa da posição do veículo a qual dependera da qualidade da amostragem.

Neste trabalho que faz parte de um projeto maior que visa avaliar interfaces de programação paralela (*Pthreads, MPI, Cuda, OpenMP e Cilk++)* aplicadas a transformações gráficas, propõe-se o desenvolvimento de um algoritmo em paralelo utilizando o OpenMP para a determinação de similaridade de histogramas entre o veículo a ser rastreado e todas as regiões da imagem. Optou-se pela utilização do OpenMP pois é uma API que vem sendo utilizada com bons resultados na paralelização de algoritmos de processamento de imagens.

2. OpenMP

O OpenMP (*Open Multi-Processing*) é uma API (*Application Programming Interface*) multi-plataforma para processamento paralelo baseado em memória compartilhada para as linguagens C/C++ e Fortran, que consiste em um conjunto de diretivas para o compilador, funções de biblioteca e variáveis de ambiente [4].

A maior parte das variáveis no código do OpenMP são visíveis a todas as *threads* por padrão. Porém algumas variáveis individuais são necessárias para evitar condições de corrida e existe a necessidade de passar valores entre a parte sequencial e a região paralela, para que os dados do ambiente sejam introduzidos como as cláusulas atribuídas ao compartilhamento de dados.

Essa API foi especificada por um grupo dos grandes fabricantes de *hardware/software* com o intuito de ser portável e escalável, com uma interface de utilização simples e que pudesse ser utilizado tanto para aplicações de grande porte, quanto para aplicações desktop. O OpenMP usa um modelo *fork/join*, onde existe um fluxo de execução principal (*master thread*) e quando necessário, novas *threads* são disparadas para dividir o trabalho (fases paralelas). Por fim, é realizado um *join*.

3. Métodos

Os histogramas utilizados para calcular a similaridade das regiões são extraídos de regiões retangulares da imagem em níveis de cinza (luminância) definidas por uma janela de $N \times M$ *pixels*, utilizando 256 bins (número de *pixels* com um determinado nível de cinza). Os histogramas são normalizados pelo número total de pixels contido na região retangular. O histograma do veículo é definido pela região retangular que contenha o veículo a ser localizado. Para cada *pixel* da imagem, nós definimos um retângulo (com as mesmas dimensões do retângulo que contém o veículo) e extraímos um histograma o qual será comparado com o histograma da região do veículo. Dessa forma podemos calcular a similaridade de todos os *pixels* da imagem com a região que contenha nosso veículo.

A comparação entre histogramas é feita através de uma medida de similaridade calculada através da distância de Hellinger descrita em termos do coeficiente de Bhattacharyya [3]. A distância de Hellinger é definida como:

$$d = \sqrt{1 - \rho[p, q]}$$

Onde d é a distância de Helinger e $\rho[p,q]$ é o coeficiente de Bhattacharyya definido como:

$$\rho[p, q] = \sum_{n=1}^{m} \sqrt{p_n q_n}$$

Sendo p_n o *n*-ézimo bin do histograma de níveis de cinza da região do veículo e q_n o *n*-ézimo bin do histograma calculado para uma região da imagem.

4. Algoritmo de Similaridade entre Histogramas

Para realizar o cálculo da similaridade entre os histogramas foi desenvolvido um algoritmo em linguagem C, utilizando funções de manipulação da imagem em tempo real através da biblioteca OpenCV (*Open Source Computer Vision*) [5], a qual foi utilizada para carregar e manipular valores de pixels da imagem.

O algoritmo está dividido em três etapas, como pode ser visto na sequência da Figura 1.

(a) **(b)** **(c)**

Figura 1: Cálculo do Histograma

Após o carregamento da imagem, como visto na Figura 1 (a), é realizado o cálculo do histograma do objeto a ser rastreado em outras imagens (pode-se visualizar o objeto à ser rastreado circulado de preto). A Figura 1 (b) é obtida após a realização do algoritmo que calcula a similaridade entre histogramas de uma mesma imagem. Podemos notar que no local onde havia o objeto a ser rastreado apresenta tons de cinza mais escuros. Já na Figura 1 (c) obtém-se uma imagem gerada com a escala de tons de cinza após a realização dos cálculos. Como resultado observa-se que as partes onde há similaridade entre os histogramas o nível de cinza da imagem tende a ser próximo de 0.

Como o tempo para calcular a similaridade entre os histogramas utilizando um algoritmo sequencial é lento e exige poder computacional, desenvolvemos um algoritmo paralelo com OpenMP. Podemos visualizar como foi realizada a paralelização através do Algoritmo 1.

```
1             int chunk = 10;
2             #pragma omp parallel private(x, y, hist) shared (chunk)
3             {
4             #pragma omp for schedule(static, chunk)
5                     for(....){
6                             for(...){
7                             Calculo do histograma em níveis de cinza da imagem.
8                             Calculo da similaridade entre histogramas.
9                             }
10                    }
11            }
```

Algoritmo 1: Paralelização com OpenMP

Onde, na linha 1 é definido o tamanho dos blocos de iteração. Na linha 2 e 3, define-se o ínicio de uma região paralela indicando a existência de variáveis privadas (x, y, hist) e uma variável compartilhada (chunk). O *#pragma omp for* da linha 4 faz com que o laço *for* dentro da região paralela seja dividido entre as *threads* e a cláusula *schedule(static,chunk)* define que as iterações serão distribuidas de forma estática entre as *threads*. A linha 11 indica o fechamento da região paralela.

5. Resultados

O ambiente que escolhemos para desenvolvimento e execução das aplicações foi um computador Intel Core i7 2,67 Ghz. Sua arquitetura possui 4 núcleos físicos com tecnologia *hyperthreading*, o que permite a execução de *threads* em até 8 núcleos lógicos. A avaliação foi realizada utilizando um código sequencial comparado com código em OpenMP com 2, 4 e 8 *threads*.

Para uma maior confiabilidade dos resultados o sistema operacional (*Debian*

Etche – Kernel 2.6.24) encontrava-se sempre no mesmo estado e foram realizadas 100 execuções com cada algoritmo, excluíndo-se das médias finais de desempenho, os 10 melhores e os 10 piores resultados.

Podemos acompanhar na Tabela 1 que as versões paralelas mostraram um ganho de desempenho satisfatório se comparado com a versão sequencial do algoritmo.

Algoritmo	Tempo (s)	SpeedUp
Sequêncial	6,99	---
OpenMP 2 Threads	3,57	1,95
OpenMP 4 Threads	1,81	3,86
OpenMP 8 Threads	1,68	4,16

Tabela 1: Média das Execuções e SpeedUp.

Entre as versões paralelas, as que tiveram menos diferença entre si, foram a solução com 4 e 8 threads. Isto provavelmente aconteceu devido à arquitetura do processador que utilizamos usar núcleos lógicos sobre núcleos físicos, portanto houve um pequeno ganho de desempenho através do *hyperthreading*, mas não tão significativo, ou com a utilização de 8 threads houve uma maior concorrência entre os recursos o que acarreta perda de desempenho. Estas hipóteses ainda não foram analisadas.

6. Conclusões e Trabalhos Futuros

Através das avaliações, validamos uma solução paralela para determinação de similaridade de histogramas, que mostrou no melhor caso, um speedup de 4.16 quando utilizadas 8 threads.

Nós pretendemos desenvolver como passo seguinte em nossa pesquisa, um algoritmo de rastreamento o qual utilize o algoritmo de determinação de similaridade de histogramas em paralelo apresentado nesse trabalho, com o objetivo de melhorar a precisão da localização dos veículos. Pretendemos também explorar o paralelismo para desenvolver algoritmos com diferentes medidas de similaridade simultâneas, bem como avaliar a utilização de outras bibliotecas e técnicas de programação paralela.

Referências

[1] A. B. Oliveira e J. Scharcanski. Vehicle Counting and Trajectory Detection Based on Particle Filtering. In XXIII Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI), Gramado, 2010.

[2] C. Bishop, Pattern Recognition and Machine Learning. Singapore: Springer, 2006.

[3] F. Aherne, N. Thacker, and P. Rockett. The bhattacharyya metric as an absolute similarity measure for frequency coded data. Kybernetika, vol. 34, no. 4, pp. 363–368, 1998.

[4] X. Tian, J. P. Hoeflinger, G. Haab, Y-K Chen, M. Girkar and S. Shah. A compiler for exploiting nested parallelism in openmp programs. Parallel Computing 31(10-12):960–983, 2005.

[5] G. R. Bradski and A. Kaehler. Learning opencv, 1st edition, O'Reilly Media, Inc., 2008.