DVMF - Distributed Vector Median Filter: Eliminação de ruído impulsivo em imagens coloridas de forma distribuída *

Guilherme I. Lazzari[1], Eduardo W. Basso[2]

[1]Programa de Pós-Graduação em Computação Aplicada
Centro de Ciências Exatas e Tecnológicas
Universidade do Vale do Rio dos Sinos
Sao Leopoldo - RS - Brasil
guila@exatas.unisinos.br

[2]Unidade de Gestão de Conhecimento Computação
Universidade Luterana do Brasil
Canoas - RS - Brasil
ewbasso@yahoo.com.br

Introdução

Algoritmos de processamento de imagens são largamente usados em conjunto com outras áreas da computação em aplicações como identificação de vegetação a partir de imagens de satélites, localização de objetos em um certo ambiente e aplicações médicas. Aplicações que fazem uso de imagens como base para a solução de determinado problema necessitam que estas imagens estejam com boa qualidade. Entende-se por boa qualidade fotos com uma iluminação adequada, sem distorções, com o foco correto, sem ruído e alguns outros pontos que a aplicação exigir.

O problema é que na maioria das vezes a captura das imagens não é feita, ou nem sempre é possível, obter fotos com a qualidade ideal. Como exemplo cita-se a identificação de vegetação utilizando Redes Neurais Artificiais. Para que a rede possa aprender corretamente a identificar os diferentes tipos de vegetação, fica clara a necessidade de fotos de satélite com boa qualidade. Caso contrário, a aplicação certamente não terá resultados satisfatórios.

Um dos problemas que mais ocorre em imagens deste tipo são os ruídos que danificam a foto. Algoritmos de eliminação de ruído consomem alto custo computacional. Em imagens coloridas este problema se agrava, uma vez que temos três componentes de cor (RGB) para cada *pixel*. Este artigo propõe então uma implementação distribuída baseada em Grid de um algoritmo vetorial para eliminação de ruído impulsivo. O propósito é tratar uma grande quantidade de imagens coloridas de forma distribuída, buscando assim o ganho de desempenho. A implementação funciona sobre Linux e Windows separadamente, usando o paradigma de orientação a objetos e a linguagem C++.

*Desenvolvido na disciplina de Processamento Distribuído, ministrada pelo professor. Dr. Gerson H. Cavalheiro na Universidade do Vale do Rio dos Sinos, 2003.

Filtro da Mediana Vetorial (VMF)

Filtros lineares e de estatística de ordem podem ser aplicados com facilidade a imagens em tons de cinza, onde cada ponto é representado por um único valor escalar. O primeiro consiste na operação de convolução de uma máscara sobre a imagem a ser processada. Já os filtros de estatística de ordem são filtros não lineares que utilizam dados estatísticos e de comparação para obter seu resultado [GON 00].

O problema ocorre em imagens coloridas, pois cada ponto é descrito por três valores representando a intensidade nos componentes vermelho, azul e verde [BAS 03]. O filtro implementado de forma distribuída para este artigoé conhecido como Filtro da Mediana Vetorial (VFM), onde a filtragem é efetuada através da aplicação de uma janela de ordem *n* a cada ponto da imagem. Verifica-se a necessidade de uma implementação distribuída devido principalmente a três fatores: imagens coloridas, tamanho normalmente grande das imagens e o tamanho da janela para eliminação do ruído. Chamamos esta implementação de DVMF, ou seja, Filtro da Mediana Vetorial Distribuído.

A escolha da amostra é dada pelo ponto P_I que minimiza o somatório de distâncias euclidianas no espaço RGB entre a cor deste ponto e as cores das demais amostras da janela. O coeficiente de escolha C_I de um ponto P_I é dado por [BAS 03]:

$$C_I = \sum D(P_I, P_J), \text{ com } D\left(P_I, P_J\right) = |P_I - P_J|$$

onde *N* é o número de amostras na janela e *I* é o índice da amostra escolhida.

Modelo Proposto

O sistema de distribuição do processamento do filtro da mediana vetorial (DVMF) segue a arquitetura mestre-escravo. Neste sistema há um processo Mestre que gerencia as tarefas a serem feitas, por exemplo, uma lista de imagens a serem processadas. O Mestre divide o trabalho entre processos Escravos que estão aptos a realizar tarefas, ou seja, processar as imagens [MAO 95].

A arquitetura deste sistema foi idealizada conforme o conceito de Grids Computacionais, onde computadores quaisquer podem voluntariar-se como novos trabalhadores ou deixar de colaborar a qualquer momento. A comunicação entre os processos é feita através de interface sockets do protocolo TCP/IP. Sockets são usados tipicamente para troca de dados via TCP ou UDP. O uso do protocolo de transporte com suporte a fluxo (TCP) foi adotado para este modelo por ser importante devido, principalmente, à necessidade de transmissão das imagens, onde deve-se garantir a ordem e a não perda dos pacotes. No modelo UDP esta característica não está presente, não garantindo ordem e consistência dos pacotes.

A idéia principal do modelo, portanto, é aproveitar ao máximo o processamento de computadores disponíveis.

No processo Mestre há um socket servidor que espera pelas conexões dos sockets clientes, que representam os Escravos que farão parte do Grid. Internamente ao processo Mestre, os escravos estão divididos em três listas: ***ReadyList*** é a lista de prontos, processos que já concluíram suas tarefas e estão esperando por novas imagens; ***BusyList*** é

uma lista de ocupados, que ainda estão trabalhando de alguma forma (recebendo, processando ou transmitindo uma imagem); ***FrozenList*** consiste em uma lista de congelados, que indicaram não estarem aptos para processar imagens.

Todos os Escravos, quando se conectam ao Mestre, são colocados no final da *ReadyList*. Como a participação no Grid é voluntária, a qualquer momento um Escravo pode informar não estar apto a trabalhar, sendo retirado da lista que estiver (*ReadyList* ou *BusyList*) e colocado na lista de congelados (*FrozenList*), sendo cancelado qualquer trabalho que esteja fazendo. Neste caso, a imagem é recolocada em *ImageList*. Quando o Escravo volta a estar apto a trabalhar, este é retirado da *FrozenList* e inserido no final da lista de prontos. As imagens a serem processadas são carregadas através da indicação do diretório em que estão, via linha de comando, sendo armazenadas em uma lista (*ImageList*). Um escalonador é responsável por atribuir as imagens aos Escravos.

Se houverem imagens a serem processadas e Escravos aptos a trabalhar o escalonador realiza as seguintes atividades. Primeiramente remove o primeiro Escravo da lista de prontos e atribui a ele a primeira imagem de *ImageList*, que também é removida de sua lista. Como segundo passo ele insere o Escravo na lista de ocupados e carrega a imagem a ser processada. Os controles do arquivo são guardados na entrada da *BusyList*. Por fim, avisa ao Escravo que há uma imagem destinada a ele. O aviso de trabalho ao processo Escravo dispara uma seqüência de troca de pacotes para a transmissão da imagem a ser processada. Recebendo o último pacote da imagem, o Escravo então efetua o processamento do filtro da mediana vetorial e inicia uma seqüência de troca de pacotes para a devolução do resultado obtido ao Mestre. O sistema está sendo desenvolvido para efetuar apenas o filtro da mediana vetorial, mas outros filtros poderiam ser facilmente incorporados.

Após o término do processamento de uma imagem, o Escravo é removido da lista de ocupados e recolocado no final da lista de prontos. O arquivo da imagem original e o resultado obtido são movidos para um diretório de arquivos prontos chamado (*Ready*). Ao nome do arquivo é adicionada juntamente com o resultado a seqüência "_vmf". Também é gerado um arquivo de *log* que reporta cada processamento indicando o novo caminho da imagem, o início do trabalho, o tempo gasto e o Escravo que o efetuou.

O processo escravo é o responsável pelo processamento das tarefas designadas pelo mestre [TAN 95], no caso, aplicar o filtro da mediana vetorial sobre uma imagem colorida transmitida pelo mestre e devolver-lhe o resultado obtido. Existem três estados principais nos quais o processo escravo pode se encontrar, cada um indicando em qual das três listas do mestre o cliente está: *Ready* (pronto), *Busy* (ocupado) e *Frozen* (congelado). Ao ser executado, o escravo conecta-se ao Mestre e fica em estado *Ready*, ou seja, esperando ser designado ao processamento de uma imagem. Ao ser avisado de que há uma imagem a ser processada o escravo passa ao estado *Busy*, que possui ainda três sub-estados que determinam o andamento da execução da tarefa: recebendo, processando e transmitindo imagem.

A qualquer momento o escravo pode entrar em estado congelado, interrompendo assim a tarefa que possa estar sendo executada, e enviando uma mensagem de aviso ao Mestre. Um novo aviso é enviado ao Mestre quando o processo deixa de estar congelado, voltando ao estado pronto.

Resultados

Estando pronta a implementação para Windows foram realizados alguns testes visando observar o desempenho da aplicação. Para isso foram obtidas 20 imagens de tamanho 640x480 *pixels* e gerado ruído sobre elas. O plano de teste consistiu em aplicar o Filtro da Mediana Vetorial (VMF) de forma seqüencial e distribuída sobre dois grupos, um com 10 e um segundo as 20 imagens. No primeiro grupo a matriz de convolução M usada foi de 5x5 e no outro de 3x3. Os resultados são apresentados abaixo em milésimos de segundos, sobre um Grid com 4 computadores.

	10 Imagens (M = 5x5)	20 Imagens (M = 3x3)
Grid	191476 ms	85399 ms
Seqüencial	415196 ms	116093 ms

Uma análise que pode ser feita é que ao usarmos uma matriz 3x3 o custo de transmissão (ida e volta) do arquivo é muito elevado se comparado ao tempo de processamento das imagens. No caso de uma matriz 5x5 temos uma redução no *overhead* devido ao aumento no custo computacional para processar os arquivos. Isto ficará mais evidente após a conclusão de trabalhos futuros, relatados abaixo.

Conclusão

O uso de algoritmos de processamento de imagens pode ser encarado de forma distribuída, uma vez que a imagem nada maisé do que uma matriz de *pixels*. Para imagens coloridas a necessidade de distribuição fica mais clara, uma vez que estamos aumentando a quantidade de informação para processamento. O filtro abordado neste artigo serviu como motivação para a criação do modelo e avaliar sua viabilidade para aplicações que necessitam processar grande quantidade de imagens. O modelo distribuído proposto permite que outros processamentos sobre imagens sejam aplicados, bastando trocar o método ou função que filtra a imagem por outro algoritmo.

Como trabalhos futuros pretendemos em primeiro momento obter os resultados das simulações sobre uma grande quantidade de imagens e com um número *n* de máquinas sobre Windows e Linux, usando filtros que demandem mais tempo computacional. Em seguida, implementar um recurso que permita ao modelo detectar automaticamente se a máquina está ociosa, e então possa processar as imagens automaticamente.

Referências

[BAS 03]BASSO, Eduardo W. Filtros Vetoriais para a Remoção de Ruído em Imagens Coloridas. In: **CMP165 Introdução ao Processamento de Imagens**, UFRGS, 2003.

[GON 00]GONZALEZ, Rafael C., WOODS, Richard E. **Processamento de Imagens Digitais.** Traduzido por: Roberto M. César Júnior e Luciano da Fontoura Costa. São Paulo: Edgar Blücher, 2000. 509 p.

[TAN 95]TANENBAUM, Andrew S. Distributed operating systems. Upper Saddle River: Prentice Hall, 1995.

[MAO 95]Chengye Mao and Shengru Tu. Self-Adaptive Load Sharing for Distributed Image File Processing. URL: http://citeseer.nj.nec.com/37874.html.