Execução paralela de algoritmos de processamento de imagens usando memória compartilhada*

Danilo E. Gondolfo† , Franklin C. Flores , Ronaldo A. L. Gonçalves

UEM - Universidade Estadual de Maringá
Departamento de Informática
Av. Colombo, 5790. Jardim Universitário.
CEP 87020-900 Maringá, PR

daniloegea@yahoo.com.br, fcflores@din.uem.br, ronaldo@din.uem.br

Resumo. *Este trabalho tem como objetivo apresentar a avaliação de uma implementação de aplicações paralelas de processamento de imagens em um ambiente de memória compartilhada com o uso de pthreads. Foram desenvolvidas aplicações paralelas com o propósito de dividir o processamento de imagens entre um certo número de threads, as quais implementam o Filtro de Sobel, utilizado para detecção de bordas, e o Filtro da Mediana, utilizado para redução de ruídos. A análise dos resultados mostrou que a paralelização das aplicações se mostrou muito eficiente, pois permite a redução no tempo de processamento em até 90% nos casos aqui avaliados.*

1. Introdução

Nos dias de hoje tornou-se comum a aquisição de computadores com múltiplos núcleos de processamento. Mas para que todo o potencial dessas máquinas seja usado, é necessário que as aplicações estejam preparadas para explorar o paralelismo. Aplicações das diversas áreas de conhecimento podem ser projetadas com este propósito. Neste trabalho, mostramos que é possível reduzir o tempo de processamento de aplicações de processamento de imagens que exploram tais recursos de hardware.

O processamento de imagens atualmente tem sido usado em muitas áreas, como por exemplo nos exames médicos, onde os computadores auxiliam os médicos a detectarem doenças [1]. Imagens médicas de alta resolução podem ter vários gigabytes de tamanho e consumir um tempo de processamento alto. O processamento paralelo pode ajudar a diminuir esse tempo através da divisão do problema em problemas menores, os quais podem ser executados por múltiplos núcleos de processamento.

Os algoritmos aqui analisado pertecem à classe dos filtros que atuam no domínio espacial da imagem. Algoritmos de filtragem espacial possuem várias aplicações dentro do processamento de imagens. Eles podem ser usados para a suavização, redução de ruídos e detecção de bordas em imagens, entre muitas outras aplicações [2] [3]. Algoritmos deste tipo envolvem uma grande quantidade de computação e, neste aspécto,

* Projeto apoiado pela Fundação Araucária (PAC-Clusters)
† Aluno bolsista do PIBIC CNPq-FA-UEM

implementações sequenciais normalmente possuem um desempenho insatisfatório. Este tipo de problema pode ser facilmente reduzido com a exploração do processamento paralelo. Neste trabalho paralelizamos algoritmos que implementam o Filtro de Sobel, utilizado para detecção de bordas, e o Filtro da Mediana, utilizado para redução de ruídos.

Este trabalho está organizado da seguinte maneira. A seção 2 apresenta trabalhos relacionados. A seção 3 aborda o processamento de imagens e suas aplicações em diversas áreas, e introduz as aplicações que foram paralelizadas. A seção 4 descreve os conceitos sobre multiprocessamento simétrico, memória compartilhada e programação multithread. A seção 5 apresenta as soluções paralelas dos algoritmos e as implementações realizadas. A experimentação e avaliação aparecem na seção 6. Na Seção 7 estão as conclusões sobre o trabalho.

2. Trabalhos Relacionados

Tendo em vista que o processamento de imagens digitais possui aplicações nas mais diversas áreas, inclusive em situações onde o tempo de processamento é crucial, pesquisadores destas áreas tem desenvolvido métodos para torná-lo mais eficiente. Alguns desses são brevemente descritos a seguir.

No trabalho desenvolvido por Saito [1] são demonstradas técnicas de paralelização de algoritmos para o processamento de imagens médicas. São utilizados algoritmos de suavização e detecção de borda. Os experimentos foram realizados em um cluster com o uso da biblioteca MPI (Message Passing Interface) e a linguagem Java. Foram desenvolvidas aplicações sequencias e paralelas dos filtros da mediana e Sobel com tamanhos 3x3, 5x5, 7x7, 9x9 e 11x11 e, devido à grande quantidade de cálculos envolvidos nos processos, o uso da computação paralela se mostrou bastante vantajoso.

Em [4] são descritos comparativos de desempenho entre as linguagens C++ e Java no processo de convolução com paralelismo real e com concorrência. Além de avaliar o desempenho de programas sequencias e paralelos também foi avaliado o uso de diferentes paradigmas de programação (procedural e orientado a objetos). Foi percebido que usando o paradigma procedural, a linguagem C++ obteve melhores resultados que o orientado a objetos e a Java obteve um tempo de resposta menor que a C++ orientado a objetos na maioria das aplicações multithread. Ao final dos experimentos ficou evidente que as aplicações paralelas tiveram ganhos consideráveis de desempenho.

Técnicas sequenciais e paralelas para otimização de desempenho foram utilizadas em três classes de algoritmos de processamento de imagens [5] sobre a arquitetura x86: transformada rápida de Fourier, convolução e histograma. Tais técnicas permitiram alcançar ganhos de 1.19x até 2.68x em um sistema com quatro núcleos de processamento através de uma melhor utilização de vetores, uso de multiprocessamento e uso eficiente dos barramentos.

Em [6] são apresentadas maneiras de integrar CPUs e GPUs no processamento paralelo de imagens biomédicas, utilizando técnicas como matrizes de co-ocorrência, convolução e histograma. É apresentado como essa combinação em 16 nós em cluster podem ultrapassar 10 TFLOPS (trilhões de operações de ponto flutuante por segundo) de processamento. Para a distribuição dos dados e comunicação entre os nós foi utilizado o sistema DataCutter. O desempenho alcançado ao se aumentar o número de nós de 1

para 16, utilizando apenas GPUs, foi de 31.3 vezes. Nesta configuração foram usadas máquinas onde cada uma era equipada com dois processadores dual-core Opteron X2 2218 de 2.6 GHz e duas placas de vídeo Nvidia Quadro FX 5600.

3. Processamento de Imagens

Uma imagem pode ser definida como uma função de duas variáveis, no caso de imagens bidimensionais. Uma imagem bidimensional é formada por uma matriz de pixels (picture elements), onde cada célula desta matriz contém um valor que representa a cor ou nível de cinza naquele ponto. Operações com imagens são basicamente operações matemáticas com essas funções, onde uma imagem de origem pode ser transformada em outra a partir de uma função conhecida [7].

O processamento de imagens digitais é uma área da Ciência da Computação que estuda transformações entre imagens. Atualmente o processamento de imagens é aplicado em praticamente todas as áreas da ciência, desde processos industriais, como verificação de defeitos em produtos e contagem de pequenas peças, passando pelas áreas médicas como em reconhecimento de fraturas e tumores, até a pesquisa espacial, na conversão de imagens de galáxias e nebulosas em frequências fora do espectro visível por nossos olhos para imagens que possamos enxergar e analisar.

Dentre as áreas de aplicação do processamento de imagens podemos citar:

- Realce de imagens. O realce de imagens evidencia detalhes, como contornos, podendo ser usado na análise de impressões digitais para contornar as papilas dos dedos e remover borrões que dificultam o processo de leitura por equipamentos biométricos [8];
- Detecção de padrões. A detecção de padrões pode ser usada por sistemas de reconhecimento de face em imagens ou vídeos ou identificar veículos em situação irregular através do reconhecimento de placas [9];
- Segmentação de imagens. A segmentação é usada para particionar a imagem em estruturas com conteúdo semântico relevante para a aplicação em questão, bem como para destacar a localização, topologia e forma de objetos [9].

3.1. Convolução

A convolução é uma formulação matemática que pode ser usada no processamento de imagens no processo de filtragem espacial (o próprio plano da imagem) [9] e consiste na aplicação de um filtro de tamanho específico sobre conjuntos de pixels vizinhos, onde o tamanho do conjunto é definido pelo tamanho do filtro. É um método clássico, muito aplicado para a suavização de imagens ou para realce de borda.

A técnica de convolução é usada no realce de imagens, onde uma determinada intensidade, ou faixas de intensidades, de cor será evidenciada. A convolução também pode ser usada para realçar bordas (limites entre objetos distintos) e suavizar imagens. O Algoritmo 1, demonstra o processo de convolução aplicado a uma imagem **f** com dimensões MxN pixels utilizando um filtro **w** mxn pixels, o resultado é uma imagem **g**. Este algoritmo não leva em consideração os limites de borda da imagem.

Algoritmo 1 Processo de convolução

$x1 \leftarrow \lfloor m/2 \rfloor$
$y1 \leftarrow \lfloor n/2 \rfloor$
para $x \leftarrow 0$ **até** $M - 1$ **faça**
    para $y \leftarrow 0$ **até** $N - 1$ **faça**
        $soma \leftarrow 0$
        para $i \leftarrow -x1$ **até** $x1$ **faça**
            para $j \leftarrow -y1$ **até** $y1$ **faça**
                $soma \leftarrow soma + w(i, j) * f(x - i, y - j)$
        $g(x, y) \leftarrow soma$

3.1.1. O Filtro de Sobel

O Filtro de Sobel é utilizado no processo de detecção de bordas. Esta técnica consiste na aplicação de duas máscaras na imagem, sendo que cada valor das máscaras que cobre a imagem é multiplicado pelo valor do pixel coberto e somado, após o término de cada máscara são obtidos dois valores G_x e G_y e um valor, obtido pela equação $\sqrt{G_x^2 + G_y^2}$, que é armazenado no pixel central coberto pelo máscara. Na Figura 1 podemos observar o resultado da aplicação do filtro.

(a) Imagem original (b) Imagem após aplicação do Filtro de Sobel

Figura 1: Filtro de Sobel

3.2. O Filtro da Mediana

O filtro da mediana pode ser usado na remoção de ruídos em imagens e, assim como o filtro de sobel, atua no domínio espacial da imagem. Este filtro consiste em selecionar o valor mediano entre os valores dos pixels cobertos pela máscara, através da ordenação desses valores. Neste trabalho foi usado o algoritmo de ordenação quick sort neste processo. Um filtro simples como este pode consumir um tempo de processamento considerável. Em uma máquina Core2Duo 2.23GHz com 4GB de memória DDR3 1067MHz, o processo de filtragem com duas threads levou aproximadamente 50 segundos em uma imagem com dimensões 11520x6480.

Na Figura 2 podemos ver como a filtragem de imagens pode auxiliar na redução de ruídos. Na Figura 2(b) pode-se observar um ruído conhecido como sal-pimenta. Na Figura 2(c) obtivemos uma imagem próxima a original com o uso do Filtro da Mediana. Pode-se observar que a aplicação do filtro fez com que uma borda surgisse nos limites da

imagem, isso é devido ao fato de que este processo grava os resultados das operações no centro do filtro, de tamanho 3x3 neste caso, fazendo com que nada seja gravado nos pixels da borda. Vale lembrar que este comportamento é apenas um detalhe da implementação usada no trabalho.

(a) Imagem original (b) Imagem com ruído sal-pimenta (c) Imagem após aplicação do Filtro da Mediana

Figura 2: Uso da filtragem espacial na redução de ruídos - Filtro da Mediana

4. Processamento paralelo

Os processadores com multiplos núcleos vem se popularizando cada vez mais. Atualmente é comum a oferta e disponibilização de computadores com vários núcleos e vários gigabytes de memória. Embora tenhamos acesso a essas máquinas, não são todas as aplicações que aproveitam o multiprocessamento, talvez pela dificuldade encontrada no processo de paralelização de programas.

O processamento paralelo consiste na divisão de uma aplicação em várias tarefas menores e na distribuição destas entre os núcleos disponíveis no sistema. Para que isso seja possível o sistema operacional deve suportar execução multithread entre diferentes processadores. Atualmente as organizações mais comuns de múltiplos processadores são SMP (Symmetric Multi-Processor) e clusters [10]. A programação paralela em ambientes de memória compartilhada traz uma grande quantidade de benefícios, dentre os quais podemos citar:

- Aumento de desempenho. Em sistemas com grande potencial de paralelização (como por exemplo os sistemas de processamento de imagens digitais) é notável o desempenho que se consegue adquirir com o uso de ambientes multiprocessados. Aplicações científicas podem tirar proveito do uso de vários processadores para resolverem grande problemas em um tempo consideravelmente menor;
- Baixo custo. Máquinas com múltiplos processadores em geral são mais baratas que clusters contendo vários computadores;
- Escalabilidade. Um sistema multiprocessado com memória compartilhada pode prover recursos para uma quantidade maior de usuários. Um servidor Web por exemplo pode tirar proveito de um equipamento com múltiplos núcleos para atender a uma quantidade maior de requisições de páginas.

Este trabalho se baseia no uso de programação multithread sobre uma organização SMP de memória compartilhada.

4.1. Ambientes de memória compartilhada

O termo memória compartilhada se refere a uma região de memória comum que pode ser acessada concorrentemente por threads envolvidas em um processamento paralelo. Em consequência desse fato, a sincronização entre threads deve ser tratada no desenvolvimento do software.

A sincronização se faz necessária porque uma thread depende do resultado da computação de outra para continuar sua execução ou porque duas ou mais threads tentam manipular dados em uma mesma região de memória. Um dos efeitos do não tratamento deste problema é o acesso a dados desatualizados que não fazem sentido para a aplicação ou a modificação de dados em momento inoportuno, o que pode causar resultado incorreto ou até mesmo o bloqueio da aplicação. Uma biblioteca de threads deve fornecer mecanismos para o tratamento destas situações.

O falso compartilhamento ocorre quando um determinado conjunto de bytes é carregado em uma linha da cache e diferentes processos alteram regiões distintas dessa linha. Apesar de não ocorrer a escrita exatamente no mesmo local, a arquitetura irá invalidar a linha em questão, causando perdas no desempenho. O processo de convolução é um bom exemplo de aplicação que pode favorecer o falso compartilhamento, e nesse caso, a estratégia de paralelização adotada é de fundamental importância.

Além de threads, também é possível trabalhar com memória compartilhada em nível de processos. Neste caso a memória deve ser compartilhada por meio de algum recurso do sistema operacional. Em sistemas baseados em UNIX, bibliotecas que possibilitam o compartilhamento de certas regiões de memória entre processos distintos são fornecidas ao usuário.

4.2. pthread

O padrão POSIX thread (IEEE 1003.1c) [11], ou apenas pthread, define uma interface para manipulação de threads. O padrão pthreads adveio da necessidade de se criar uma interface padrão para threads em uma época onde cada fabricante de hardware disponibilizava as suas próprias interfaces, o que dificultava a portabilidade das aplicações. Atualmente, praticamente todos os sistemas UNIX-like usam a interface definida pelas pthreads e também existe uma versão da biblioteca para Windows.

A biblioteca pthread usa o modelo explícito de threads, ou seja, o programador deve criar, manter e terminar as threads manualmente. Embora isso dê ao desenvolvedor um controle maior sobre o sistema, também torna a tarefa difícil em alguns momentos. A divisão da tarefa depende de quantos núcleos de processamento o sistema tem disponível. Os sistemas UNIX possuem mecanismos que facilitam a identificação do número de processadores e o desenvolvedor pode fazer a divisão com base nesses dados.

No caso específico do Linux, o qual é usado nos experimentos deste trabalho, quando a aplicação cria uma thread o que está sendo criado na verdade é um processo. O Linux não possui chamadas de sistema que criam threads, mas sim processos, denominada *clone()*, que também é usada na implementação das chamadas *fork()* e *vfork()* [12].

5. Proposta e implementação

Neste trabalho foi feita a implementação do Filtro de Sobel e do Filtro da Mediana utilizando a biblioteca pthread. A estratégia de paralelização adotada foi a divisão da imagem de acordo com a quantidade desejada de threads. Os Algoritmos 2 e 3 mostram as versões sequencial e paralela do processo de filtragem baseada no Filtro da Mediana, mas pode-se perceber que a estratégia de paralelização adotada foi a mesma para o Filtro de Sobel.

No Algoritmo 2 temos o pseudo-código do Filtro da Mediana sequencial. Considere **arrayPixels** um vetor de inteiros de tamanho 9 (a quantidade de pixels dentro da máscara). As variáveis **y** e **x** representam a coluna e a linha da imagem, respectivamente. Os dois laços mais internos são usados para percorrer os valores cobertos por uma máscara 3x3. A cada iteração um valor de pixel é inserido no vetor que depois é ordenado para que o valor mediano (neste caso a quinta posição do vetor) seja recuperado e gravado no centro da máscara em uma nova imagem.

Algoritmo 2 Filtro da Mediana Sequencial

para $x \leftarrow 2$ **até** $imagemWidth$ **faça**
  para $y \leftarrow 2$ **até** $imagemHeight$ **faça**
    $z \leftarrow 1$
    para $i \leftarrow -1$ **até** 1 **faça**
      para $j \leftarrow -1$ **até** 1 **faça**
        $arrayPixels[z] \leftarrow pegaPixel(imagem, y + j, x + k)$
        $z \leftarrow z + 1$
    $ordena(arrayPixels)$
    $gravaPixel(imagem, y, x, arrayPixels[5])$

No Algoritmo 3 temos o pseudo-código do Filtro da Mediana paralelo. No programa principal estão os procedimentos que dividem a imagem entre as threads. A função **disparaThread** é responsável pela inicialização das threads que executarão o procedimento **filtroMediana**, sendo que é passado como parâmetro o intervalo de linhas onde cada thread irá atuar. Por motivos de simplificação, a possibilidade de sobrar linhas durante a divisão foi ignorada.

A demonstração do algoritmos paralelo para o Filtro de Sobel, cuja versão sequencial pode ser vista no Algoritmo 1, foi omitida pelo fato de ser praticamente idêntica ao algoritmo do Filtro da Mediana. A estratégia de paralelização usada é a mesma, diferindo apenas nas operações realizadas com os pixels cobertos pela máscara.

6. Experimentação e avaliação

O experimento foi executado em um hardware SMT (Simultaneous multithreading) Dual Xeon 2.4GHz sendo que cada processador possui 4 núcleos e que cada núcleo pode executar duas threads simultaneamente. Foi utilizada a biblioteca OpenCV 2.3 para a manipulação da imagem (carregamento, leitura e gravação de pixels e descarregamento em disco) e a aplicação foi compilada com o GCC 4.1.2 sem nenhuma otimização (-O0). Todos os testes foram realizados em um sistema CentOS Linux 5.3 com o kernel na versão 2.6.18.

Algoritmo 3 Filtro da Mediana Paralelo

```
procedimento filtroMediana(começo, final)
início
    para x ← 2 até imagemWidth faça
        para y ← começo até final faça
            z ← 1
            para i ← −1 até 1 faça
                para j ← −1 até 1 faça
                    arrayPixels[z] ← pegaPixel(imagem, y + j, x + k)
                    z ← z + 1
            ordena(arrayPixels)
            gravaPixel(imagem, y, x, arrayPixels[5])
fim

programa Mediana
início
    disparaThread(filtroMediana, 2, imagemHeight/númeroDeThreads)
    para i ← 1 até (númeroDeThreads − 1) faça
        disparaThread(filtroMediana,
                      ((imagemHeight/númeroDeThreads) ∗ i,
                      (imagemHeight/númeroDeThreads) ∗ (i + 1)))
    esperaThreads()
fim
```

As aplicações foram executadas várias vezes sobre uma imagem JPG com dimensões 17125x9625 em tons de cinza, variando-se o número de threads. Podemos ver os resultados da execução do Filtro de Sobel no gráfico da Figura 3. A aplicação teve uma grande redução no tempo de processamento, proporcionada por um paralelismo real, uma vez que não foi necessário adicionar controle de concorrência em nenhuma região de memória.

O Filtro da Mediana obteve um custo maior que o Filtro de Sobel em termos de tempo de computação. Isso é devido ao fato de que este filtro necessita ordenar os valores cobertos pela máscara para selecionar o valor mediano. Os resultados da execução do Filtro da Mediana podem ser vistos no gráfico da Figura 4.

Em ambos os casos nota-se que o aumento no número de threads não proporciona redução significativa no tempo de execução a partir de 8 threads, quando a redução no tempo de processamento alcançou 88% para o Filtro de Sobel e 87% para o Filtro da Mediana. A principal justificativa para este caso é que a partir deste número de threads a execução passa a ser hyperthreading, com compartilhamento de um mesmo núcleo de processamento (core) por duas ou mais threads. Os maiores ganhos nas reduções no tempo de processamento ocorreram quando o número de threads passou de 1 para 8.

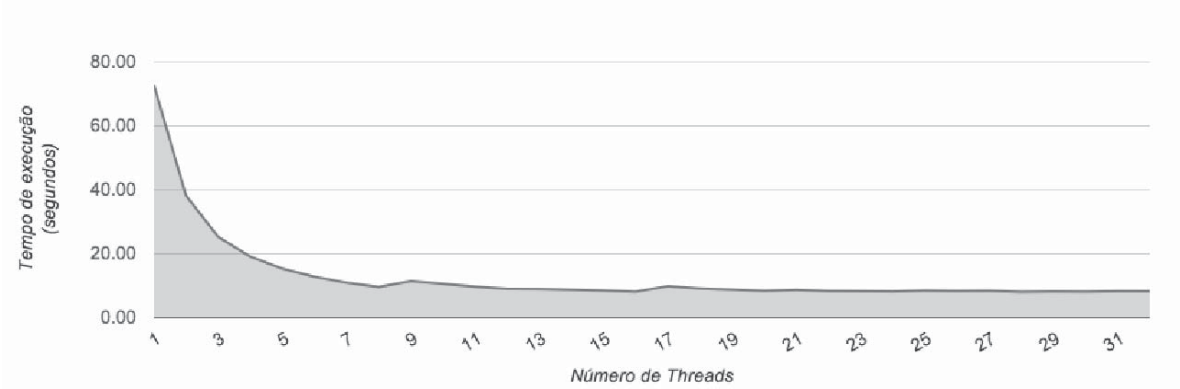

Figura 3: Tempo em função do número de threads - Filtro de Sobel

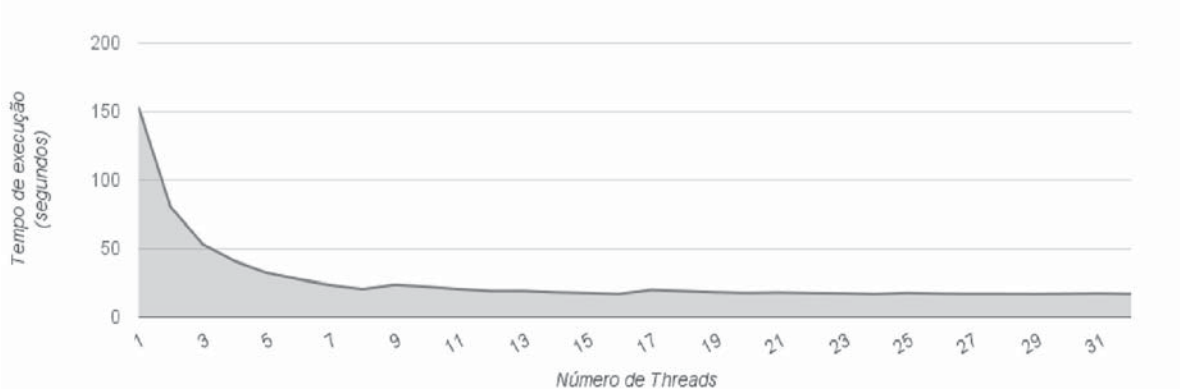

Figura 4: Tempo em função do número de threads - Filtro da Mediana

7. Conclusão

Os experimentos realizados neste trabalho permitiram observar que aplicações de algoritmos convolução e do Filtro da Mediana são ótimos para serem paralelizados, pois não precisam de controle de concorrência. Nos casos estudados aqui, a redução no tempo de processamento atingiu a escala máxima de 90%, mas o melhor custo benefício ocorreu com o uso de 8 threads.

Como trabalhos futuros, pretende-se ainda melhorar o desempenho de algoritmos de filtragem espacial transformando a imagem e as máscaras do formato matricial para o linear, na tentativa de evitar a grande quantidade de faltas da cache causadas pelo acesso matricial dos filtros.

Além disso, outros algoritmos de processamento de imagens serão paralelizados, como por exemplo algoritmos de histograma e transformadas de imagens. O objetivo é trabalhar com situações onde haja concorrência e verificar se nesses casos o paralelismo ainda apresenta vantagens sobre a execução sequencial.

Referências

[1] Saito, P. T. M., "Modelo de Paralelismo Para Processamento de Imagens Médicas," *Revista de Graduação UNIVEM*, Vol. 1, 2009, pp. 35–47.

[2] Branco, F. C., de Almeida, T. I. R., and de Souza Filho, C. R., "Filtros de convolução proporcionais para realce de imagens," *XII Simpósio Brasileiro de Sensoriamento Remoto*, Goiânia, Brasil, 2005, pp. 4013–4020.

[3] Miranda, J. I., "Detecção de Bordas com Filtro de Difusão Linear Complexa," *XV Simpósio Brasileiro de Sensoriamento Remoto*, Curitiba, Brasil, 2011, pp. 7580–7587.

[4] da Penha, D. O., Corrêa, J. B. T., Ramos, L. E. S., Pousa, C. V., and Martins, C. A. P. S., "Performance Evaluation of Programming Paradigms and Languages Using Multithreading on Digital Image Processing," *Proceedings of the 4th WSEAS International Conference on Applied Mathematics and Computer Science*, Rio de Janeiro, Brazil, 2005.

[5] Kim, D., Lee, V. W., and Chen, Y.-K., "Image Processing on Multicore x86 Architectures," *IEEE Signal Processing Magazine*, Vol. 27, Março 2010, pp. 97–107.

[6] Hartley, T. D., Catalyurek, U., Ruiz, A., Igual, F., Mayo, R., and Ujaldon, M., "Biomedical image analysis on a cooperative cluster of GPUs and multicores," *Proceedings of the 22nd annual international conference on Supercomputing*, ICS '08, New York, NY, USA, 2008, pp. 15–25.

[7] Gonzalez, R. C. and Woods, R. E., *Processamento Digital de Imagens*, Pearson Prentice Hall, São Paulo, 2010.

[8] Filho, O. M. and Neto, H. V., *Processamento Digital de Imagens*, Brasport, Rio de Janeiro, 1999.

[9] Pedrini, H. and Schwartz, W. R., *Análise de Imagens Digitais: princípios, algoritmos e aplicações*, Thomson Learning, São Paulo, 2008.

[10] Stallings, W., *Arquitetura e Organização de Computadores*, Pearson Pratice Hall, São Paulo, 8th ed., 2010.

[11] Association, I. S., "IEEE-SA - IEEE Std 1003.1c-1995 Interpretations," http://standards.ieee.org/findstds/interps/1003-1c-95_int/index.html, outubro 2011, 28.

[12] Love, R., *Linux Kernel Development*, Addison-Wesley, 3rd ed., 2010.