Nossa Experiência com a Paralelização de um Algoritmo de Processamento de Imagens Geográficas

Henrique Yoshikazu Shishido, Anderson Faustino da Silva,
Ronaldo Augusto de Lara Gonçalves, and Ligia Flavia Antunes Batista

Universidade Estadual de Maringá, Departamento de Informática,
Maringá, Paraná, Brasil
{anderson,hyshishido,ronaldo}@din.uem.br
ligia@utfpr.edu.br

Resumo O processamento de imagem digital tem sido usado em diversas áreas de pesquisa como instrumento para o reconhecimento de padrões e de filtragens específicas nas imagens. Entretanto, dependendo das características da aplicação o tempo de processamento pode ser extremamente alto e dificultar o avanço das pesquisas associadas. Nesse contexto, este trabalho propõe a paralelização de um algoritmo para o processamento de imagem digital utilizado no cálculo de índice de fragmentação multidimensional. Os resultados obtidos demostram o grande benefício da paralelização, pois a paralelização é capaz de reduzir o tempo de processamento em até 80% para alguns casos.

Palavras chave: Cluster, Memória Compartilhada Distribuída, Processamento de Imagem, Índice de Fragmentação

1 Introdução

Os avanços das pesquisas nas diversas áreas de conhecimento humano têm aumentado cada vez mais a demanda por plataformas de hardware e software, que possam oferecer maior capacidade de processamento para a execução de aplicações associadas a problemas complexos do mundo real. Aliado ao crescimento dos novos adventos tecnológicos, a popularização dos computadores tem permitido a criação de novas plataformas com um alto poder de processamento. Computadores *off-the-shelf* em conjunto com softwares de código aberto permitem a construção de um *cluster* com capacidade similar àquela oferecida por supercomputadores. Nesse contexto, a computação paralela é utilizada para implementar aplicações que demandam alto poder computacional [1].

Simulações de fenômenos físico-químicos, reconhecimento de padrões biológicos, previsões sísmicas e climatológicas e identificação de transformações na superfície do globo, entre outros, envolvem grande quantidade de processamento e podem levar dias para serem resolvidos com uso de programação sequencial. Este tempo elevado retarda o avanço das pesquisas, principalmente em áreas que exigem experimentos exaustivos e diversificados.

Muitas destas pesquisas fazem uso de processamento de imagens digitais no reconhecimento de padrões [2], sendo a convolução uma técnica comum de filtragem de

frequência que atua no domínio espacial da imagem a fim de se produzir o efeito que se deseja, conforme descrita em [3]. Entretanto, dependendo do tamanho, da quantidade e do tipo de filtragem das imagens, o processamento sequencial tradicional já não atende mais os requisitos de tempo de execução.

Nesse contexto, este trabalho propõe e avalia um modelo de paralelização para o processamento de imagens digitais baseado em convolução, com o objetivo principal de reduzir o tempo de execução do processamento. Como estudo de caso, usamos o método de convolução proposto por [3] para gerar o índice de fragmentação de uma imagem de sensoriamento remoto por satélite descrito em [4]. Esse modelo foi proposto para um sistema de memória distribuída e implementado utilizando o protocolo HLRC [5]. A avaliação de desempenho realizada comprova a importância de tal modelo.

Este artigo está organizado como se segue. A Seção 2 apresenta o referencial teórico relacionado ao presente trabalho. A descrição e a análise do algoritmo alvo são apresentados na Seção 3. A Seção 4 descreve a análise da aplicação com o objetivo de especificar a paralelização. A Seção 5 discorre sobre o modelo proposto e a sua implementação paralela. A avaliação experimental da versão paralela é discutida na Seção 6. E, por fim, a Seção 7 apresenta as considerações finais.

2 Referencial Teórico

As arquiteturas paralelas podem ser organizadas de diversas formas de acordo com a Taxinomia de Flynn [6]. Tanenbaum [7], ainda subdivide a classe MIMD, proposta por Flynn, em multiprocessadores e multicomputadores. Os multiprocessadores utilizam memória global e única, compartilhada entre os processadores, para realizar a comunicação entre os processos. Por outro lado, os multicomputadores possuem memória distribuída e comunicam-se por meio de passagem de mensagens. Dash [8] e Yu [9] conceituam uma categoria híbrida, denominada por memória compartilhada distribuída (DSM), cuja comunicação é realizada por meio do mapeamento das memórias compartilhadas de cada máquina dando a visão de uma única memória para todo o ambiente de execução. Dentro deste contexto, surge uma classe arquitetural que tem chamado atenção dos pesquisadores: o *cluster*.

Um *cluster* é uma arquitetura de multicomputadores interligados por um meio de uma rede de interconexão dedicada, normalmente acessado do mundo externo por meio de um servidor de acesso e execução, comportando-se portanto como um sistema único do ponto de vista do usuário. Os *clusters* estão sendo cada vez mais explorados devido ao seu custo/benefício [10], proporcionado pela facilidade de aproveitamento de máquinas que já não atendem as expectativas de uso enquanto isoladas. Todos os aspectos relativos à distribuição de dados, tarefas e comunicação entre os computadores podem ser abstraídos do usuário [11] por meio de uma plataforma de programação paralela baseada nos modelos de passagem de mensagem, memória compartilhada ou ainda memória compartilhada distribuída.

O modelo de memória distribuída usando passagem de mensagem [12] é, às vezes, uma alternativa que não agrada tanto os programadores. A passagem de mensagens exige que o programador faça chamadas explícitas no código para enviar ou receber

dados de um computador a outro. Esse processo abre uma maior possibilidade a erros de programação e torna difícil a depuração de código.

No ponto de vista do programador, o paradigma de memória compartilhada é atraente devido a facilidade de abstração. Assim, várias pesquisas têm sido realizadas para o desenvolvimento de ambientes de memória compartilhada distribuída. Neste modelo, cada memória encontra-se respectivamente distribuída com o seu processador, mas cada processador tem uma visão de um espaço único e global da memória [13].

2.1 O Protocolo HLRC

O protocolo *home-based lazy release consistency* (HLRC) [5] é uma implementação de um modelo relaxado de memória compartilhada distribuída baseado em residência. Inicialmente desenvolvido para uma plataforma VIA, possui atualmente uma versão com suporte ao protocolo TCP/IP [14]. A vantagem deste modelo é o fato de facilitar a programação, retirando a responsabilidade do desenvolvedor no controle do envio e recebimento de mensagens.

A implementação do protocolo utiliza o conceito de intervalos para sua gerência. Um intervalo delimita um espaço no tempo onde deve exitir sincronização, ocasionando a realização de consistência entre as memórias. Um intervalo é delimitado pelo uso de primitivas de sincronização, que pode ser o uso de *locks* ou barreiras. No final de cada intervalo são calculados os *diffs* para todas as páginas modificadas. Após criados, estes são enviados para os seus respectivos *homes*. Os *homes* aplicam os *diffs* a medida que estes chegam e, em seguida, os discarta. Desta forma é garantido que o *home* sempre tenha a cópia mais atual de um dado.

3 Aplicação Alvo: Processamento de Imagens e o Índice de Fragmentação

Os métodos de filtragem permitem que determinadas características de uma determinada imagem sejam visualizadas de uma forma melhor, transformando-a em uma forma adequada à aplicação [15]. Tais métodos estão estruturados em dois domínios: espacial e de frequência. O domínio espacial está relacionado à influência da vizinhança de um *pixels* sobre ele, enquanto o domínio de frequência refere-se ao processamento exclusivo de um *pixel*.

No domínio espacial, um ponto *p(x,y)* depende do nível de cinza original do próprio ponto e de seus *pixels* vizinhos. A convolução é uma técnica que atua sobre o domínio espacial através de matrizes denominadas de janelas ou máscaras. Esta técnica pode fazer uso de diferentes algoritmos: soma ponderada dos pontos vizinhos, mediana dos pontos, número de vizinhos distintos, entre outros. A cada elemento da janela está associado um valor numérico denominado de coeficiente. A aplicação da janela com centro na coordenada *(a,b)*, sendo *a* uma linha e *b* uma coluna, consiste na substituiçao do valor do *pixel* central por um novo valor que depende de operações que envolvem o coeficiente dos *pixels* vizinhos. Em filtragens de imagens digitais é possível aplicar os filtros em janelas de quaisquer dimensões. Porém, alguns programas definem um tamanho máximo das janelas devido a limitações computacionais [2].

Turner [4] apresenta o índice de fragmentação para medir o grau de variabilidade da paisagem e revelar as possíveis influências da atividade humana sobre a mesma. O índice fragmentação pode ser adotado como uma medida local de textura, com valores no domínio entre zero e um, sendo calculado para cada ponto da imagem por meio da aplicação de uma convolução específica na região vizinha do ponto. Se todos os *pixels* posicionados em relação à janela de convolução forem de classes diferentes, seu valor será 1, indicando máxima variabilidade na região em análise. Em contrapartida, se todos os *pixels* da imagem forem iguais, tem-se o coeficiente de variabilidade 0 [16]. Em termos de reconhecimento de padrões, o índice de fragmentação multidimensional (IFM) considera, além da vizinhança do *pixel* em cada banda, seu contexto nas múltiplas bandas para classificá-lo [17].

Para este trabalho, foi adotada a versão sequencial do algoritmo de IFM implementada pelo Grupo de Pesquisas em Geodésia da Universidade Estadual de São Paulo (UNESP). Esta versão foi utilizada como referência e serviu de código base para a criação da versão paralela. O algoritmo implementado pelo Gruopo da UNESP considera imagens com qualquer número de bandas e, após o processamento, gera uma única banda de saída. Para cada posição da janela, os *pixels* da imagem são mapeados para classes e indexados numericamente em intervalos a que cada *pixel* pertence. O tamanho desses intervalos é determinado a partir da divisão do número total de níveis de cinza da imagem pelo número de elementos da janela (*n*). Após o mapeamento dos *pixels* da janela para as classes de intervalos, realiza-se a contagem do número de *pixels* distintos (*c*). O IFM (*F*) é calculado pela Equação 1:

$$F = \frac{c-1}{n-1} \tag{1}$$

Supondo uma imagem em escala de cinza de 8 bits com janela 3 por 3. O primeiro procedimento de processamento atua em que cada posição da janela que possui apenas um valor que representa o número digital de brilho do *pixel*. Os valores de brilho são mapeados conforme as classes de intervalos definidos pela divisão dos intervalos. A atribuição do novo valor de brilho para o *pixel* central depende do número de valores presentes na janela em cada iteração. Assim, pela Equação anterior, tem-se o índice de fragmentação, o qual é multiplicado ao final do processamento do *pixel* associado por 255, para converter os índices em tons de cinza e assim permitir sua visualização como uma imagem.

O tempo de execução do Algoritmo IFM cresce com o aumento dos seguintes atributos: *dimensão da imagem*, *o número de bandas de cada imagem* e o *tamanho da janela* (máscara). A dimensão da imagem e o número de bandas alteram linearmente o tempo de execução desde que os demais parâmetros sejam os mesmos. O terceiro parâmetro influencia exponencialmente o tempo de execução.

3.1 Descrição do Algoritmo Sequencial

Para facilitar o entendimento geral do programa e do método de paralelização proposto, o Algoritmo 1 apresenta a versão sequencial utilizada.

Primeiramente, são inicializadas as variáveis de acordo com os parâmetros passados pelo usuário. Nesta etapa, são fornecidos como parâmetros *a máscara* e *o arquivo da*

Algoritmo 1 A versão sequencial do programa.

```
01 CarregarImagem();
02 AtribuirJanelaDeVizinhos();
03 DividirIntervalo();
04 RealizarConvolucao();
05 CriarBufferParaNovaImagem();
06 se lin < totalLinhas então
07    se col < totalColunhas então
08       DefinirValoresDoPixel();
09       se col > 1 então
10          EncontrarIntervalo();
11          RetornarNovoPixelCentral();
12          DefinirValorMapeado();
13          ArmazenaValorMapeado();
14       senão
15          DeslocarMascara();
16 senão
17    GravarImagem();
```

imagem de entrada. Após ler os dados de inicialização, é realizada a alocação das estruturas matriciais de entrada e saída de acordo com a dimensão da imagem de entrada. Posteriormente, os intervalos são divididos de acordo com o tamanho da máscara para a representação em tons de cinza.

Este algoritmo é marcado por funções que contém muito laços de repetição aninhados. A sub-rotina de início da convolução realiza L x C chamadas ao procedimento de cálculo do índice do pixel central. Esta função percorre cada pixel da matriz para classificar o intervalo de cada elemento e, também, realiza a contagem de números distintos contidos na janela. As iterações dos laços de repetição aninhados dependem diretamente dos parâmetros iniciais que definem quantas iterações serão realizadas. Para cada cálculo do *pixel* central realizado na rotina de convolução é calculado o valor do *pixel* central da janela. No final do processo de convolução em todos os *pixels* da imagem é gravada uma imagem de saída.

4 Análise para a Paralelização do Algoritmo

O primeiro passo para paralelizar o Algoritmo IFM foi analisar o problema em busca de segmentos particionáveis. Após analisar e compreender as suas peculiariedades, observou-se que o cálculo de cada *pixel* central da imagem de saída pode ser realizado de maneira independente do cálculo de *pixels* vizinhos.

Na versão sequencial, para cada janela é processado um *pixel* central de acordo os parâmetros iniciais do algoritmo. Para o cálculo de cada *pixel* central é gerada uma janela a partir da imagem de entrada em que são computados os intervalos distintos entre cada *pixel*. Este cálculo independe de qualquer outro resultado previamente calculado, havendo a independência de dados. Então, é possível prosseguir o cálculo dos próximos *pixels* sem a necessidade de aguardar outras instruções serem executadas.

Deste modo, distribuir segmentos da imagem de entrada entre os elementos de processamento do *cluster* é uma maneira coerente para se particionar o domínio da imagem. Assim, cada nó do *cluster* será responsável por calcular cada *pixel* central do segmento de imagem recebida e, posteriormente, devolver o segmento de saída calculado à uma máquina mestre. Este modelo de particionamento independe do número de nós existentes, pois o cálculo da divisão da imagem é dada pela razão entre a *altura da imagem* e *número de nós* do *cluster*. Assim, se temos uma imagem de entrada de dimensão 8460 x 9530 *pixels* e um *cluster* formado por 12 nós de processamento, podemos processar segmento de imagem de 705 x 9530 *pixels* em cada nó.

É importante observar que este trabalho trata o particionamento e distribuição dos dados entre os nós sem considerar as características individuais de cada nó. Neste trabalho não tratamos o balanceamento de carga para o caso de haver variação do poder computacional de cada nó. Poreém, existem diversas técnicas para executar o balanceamento de carga, como as descritas por [18], que geralmente adotam soluções baseadas em análise da capacidade de processamento do *hardware* de cada nó e o *status* dos recursos computacionais disponíveis.

5 Modelo e Implementação Paralela

Um modelo de paralelização foi desenvolvido utilizando o protocolo HLRC a partir do algoritmo IFM sequencial. Este modelo básico aplica a convolução sobre todos os pontos da imagem.

O Modelo Básico calcula todos os *pixels* da imagem de saída da mesma maneira que a versão sequencial, porém de forma paralela, usando uma abordagem de memória compartilhada distribuída. Neste modelo, apenas um nó da aplicação lê a imagem de entrada e todos os parâmetros necessários para o processamento. Em seguida, cada nó calcula a sua faixa de dados de modo proporcional ao número total de nós disponíveis. Este particionamento é feito nas linhas da imagem. Por exemplo, supondo que uma imagem de entrada possua a dimensão igual a 12 linhas e 80 colunas e um *cluster* composto por 3 nós escravos, cada nó terá um segmentos de 4 linhas e 80 colunas. Após o particionamento dos dados, cada nó executa o Algoritmo IFM sobre sua porção. Quando todos os nós terminam, o nó, que carregou a imagem para a memória compartilhada, grava em disco a imagem de saída resultante.

6 Análise Experimental

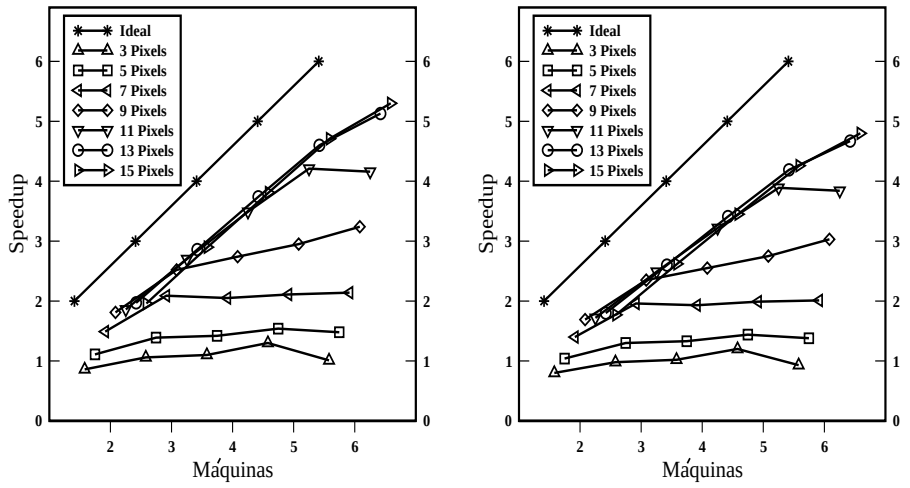
O Laboratório de Computação de Alto Desempenho (LECAD) do Departamento de Informática da Universidade Estadual de Maringá possui um *cluster* com nós escravos homogêneos onde todos os experimentos deste trabalho foram executados. Este *cluster* é composto por seis máquinas da Sun Microsystems. Cada máquina possui um processador AMD Opteron 1218 de 2.6Ghz de dois núcleos, 2MB de *cache* L2, 4 GB de memória RAM, 1TB de armazenamento e rede *Ethernet Gigabit*.

O ambiente de execução dos nós é formado pelo sistema operacional Linux CentOS com *kernel* 2.6.18, GCC 4.1.2, versão TCP/IP do protocolo HLRC, e pela ferramenta de monitoramento Ganglia [19]. Toda esta plataforma operacional é gerenciada pela

ferramenta ROCKS [20]. Além disso, foi adotada a biblioteca OpenCV 2.1 para a leitura e gravação das imagens [21].

6.1 Desempenho da Versão Paralela

A Figura 1 apresenta o *speedup* obtido pela versão paralela. Nesta figura é apresentado o *speedup* relativo, como também o *speedup* real. O *speedup* relativo foi calculado tendo como versão sequencial a versão paralela executada em apenas um processador. Por outro lado, o *speedup* real foi calculdado tendo como versão sequencial o melhor código sequencial, ou seja, um código eficiente sem nenhuma primitiva paralela.



(a) Speedup Relativo (b) Speedup Real

Figura 1. Speedups obtidos pela versão paralela.

Como é demonstrado na Figura 1(a), o ganho de desempenho somente é obtido quando a quantidade de pixels aumenta. Utilizando uma máscara de até sete *pixels*, o *speedup* não ultrapassa dois. Isto é ocasionado pelo fato de uma máscara pequena proporcionar pouco processamento, aumentando o *overhead* da comunicação.

Com uma máscara de nove *pixels* a versão paralela começa a se mostrar uma boa alternativa. Contudo, utilizando esta máscara o *speedup* ainda não é o ideal, sendo de apenas 3,24.

O melhor *speedup* foi obtido com o uso de uma máscara de quinze *pixels*, chegando bem próximo do linear, ou seja, 5,30 para uma execução com seis nós. Este é um resultado muito bom, mostrando que a versão paralela é escalável a medida que o tamanho da entrada aumenta.

Como pode ser observado na Figura 1(b), o *speedup* real é menor que o relativo. Isto devido ao *overhead* da paralelização. Mesmo utilizando a versão paralela em um único nó, existe o *overhead* das operações paralelas, que no caso da aplicação implementada chega a ser de 10% da melhor versão sequencial.

O *overhead* imposto pelas operações paralelas ocasionam um aumento do tempo de execução e consequentemente um aumento do *speedup*. Contudo, a paralelização de uma determinada aplicação deve levar em consideração, como base de comparação, o tempo da melhor versão sequencial. Pois desta forma, tem-se um *speedup* que realmente representa o ganho efetivo.

Assim como o *speedup* relativo, o *speedup* real somente passa a ter um valor considerável quando o tamanho da máscara ultrapassa nove *pixels*. Para a maior entrada, o *speedup* real é de 4,80. Mesmo com uma redução de 10% do *speedup* relativo ainda se tem um bom desempenho para a versão paralela. Isto demonstra que a versão paralela implementada foi efetiva para melhorar o desempenho da melhor versão sequencial.

6.2 Análise Detalhada

A Figura 2 apresenta o tempo de execução paralelo (para uma execução com seis nós) decomposto em três componentes, a saber: tempo gasto em computação, tempo gasto em comunicação e tempo gasto pela gerência do protocolo (*overhead*).

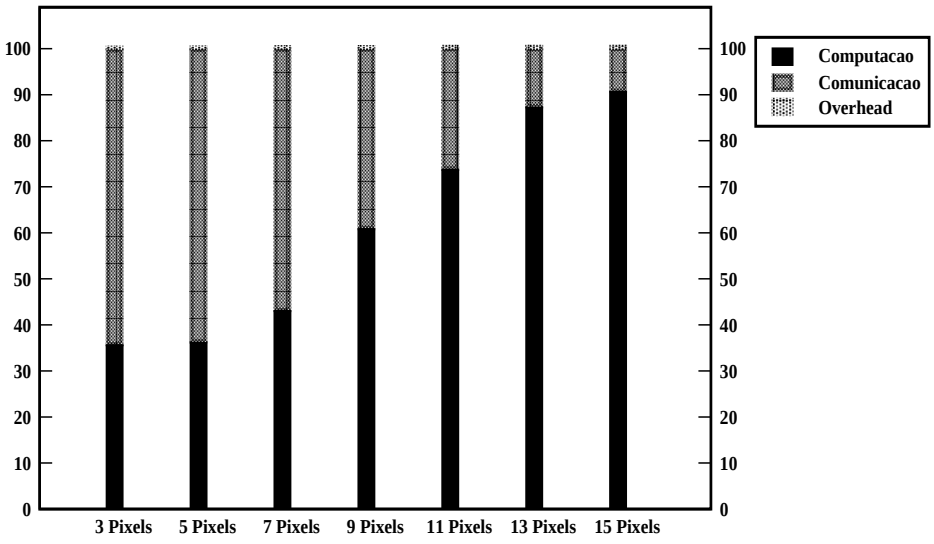


Figura 2. Breakdown do tempo de execução da versão paralela, para seis máquinas.

Como pode ser observado, estes dados corroboram com os resultados mostrados na seção anterior. Para quantidade pequenas de *pixels* a comunicação chega a consumir

64% do tempo total de execução. Este alto custo começa a reduzir somente quanto a quantidade de *pixels* é igual ou superior a nove.

Utilizando uma máscara de nove *pixels* o custo da comunicação é reduzido para 40%. Contudo, quando é utilizado uma mascará de quinze *pixels* o custo da comunicação é de apenas 10%. Isto explica o valor do *speedup* obtido pela execução com esta máscara.

Um ponto importante a ser destacado é o fato do tempo de gerência do protocolo HLRC ser muito baixo, chegando a ser desprezível. Isto demonstra que o modelo de memória compartilhada distribuída é uma boa alternativa a ser utilizada na paralelização de aplicações para *clusters*.

É importante ressaltar que paralelizar uma determinada aplicação não garante a obtenção de um ganho de desempenho considerável, em outras palavras, que o *speedup* será bem próximo do linear. Para que o ganho de desempenho seja efetivo uma característica essencial que deve ser endereçada pela estrutura paralela da aplicação é a redução ao máximo da sincronização entre as unidades paralelas. No tocante, a implementação realizada ter alcançado esta característica foi o que proporcionou a obtenção de um bom desempenho.

7 Conclusões

A computação paralela em *cluster* é indicustivelmente uma solução atrativa para aumentar o desempenho de aplicações com investimento relativamente pequeno. Entretanto, ainda existe uma carência quanto a proposição e utilização de metodologias de paralelização em aplicações complexas do mundo real. O presente trabalho objetivou diminuir esta lacuna, propondo e avaliando um modelo de paralelização de uma aplicação importante para a ciência geográfica.

O modelo proposto neste trabalho permitiu executar uma aplicação paralela reduzindo o esforço computacional. Este modelo implementado com o protoloco HLRC começa a ser vantajoso a partir de nove *pixels* de máscara obtendo um tempo de execução 66,97% menor do que a versão sequencial e 79,15% com quinze *pixels* de máscara. A inviabilidade da execução deste modelo com máscaras menores se dá devido aos do custo de comunicação entre os nós chegar a 64%. Assim, com os dados obtidos podemos comprovar que o modelo paralelo proposto apresenta desempenho e eficiência satisfatória para máscaras que utilizem mais de nove *pixels*.

Certamente, os benefícios do modelo aqui apresentado poderiam ser maiores se considerarmos uma convolução mais demorada. Para trabalhos futuros será desenvolvido políticas de seleção mais elaboradas, bem como o uso deste modelo em imagens de multibandas.

Referências

1. Ivanov, L.: A Modern Course on Parallel and Distributed Processing. Journal of Computing Sciences in Colleges **21**(6) (2006) 29–38
2. Crosta, A.P.: Processamento digital de imagens de sensoriamento remoto. Technical report, IG/Unicamp (1992)

3. Gonzalez, R., Woods, R.: Processamento de Imagens Digitais. 1th edn. Edgard Blucher, São Paulo (2000)
4. Turner, M.: Landscape Ecology: The Effect of Pattern on Process. Annual Review of Ecology and Systematics **20**(1) (1989) 171–197
5. n Rangarajan, M., Iftode, L.: Software Distributed Shared Memory over Virtual Interface Architecture: Implementation and Performance. In: Proceedings of the 4th Annual Linux Showcase and Conference. (2000) 341–352
6. Flynn, M.J., Rudd, K.W.: Parallel Architectures. ACM Computing Survey **28**(1) (1996) 67–70
7. Tanenbaum, A.S.: Organização Estruturada de Computadores. 5th edn. LTC, Rio de Janeiro (2007)
8. Dash, A., Demsky, B.: Software Transactional Distributed Shared Memory. In: Proceedings of the Symposium on Principles and Practice of Parallel Programming, New York/USA, ACM SIGPLAN (2009) 297–288
9. Yu, B.H., Huang, Z., Cranefield, S., Purvis, M.: Homeless and Home-based Lazy Release Consistency Protocols on Distributed Shared Memory. In: Proceedings of the Conference on Computer Science, Australasian, ACSC (2004) 117–123
10. Bell, G., Gray, J.: What's next in high-performance computing? ACM Communications **45**(2) (2002) 91–95
11. Bader, D.A., Pennington, R.: Applications. International Journal of High Performance Computing Applied **15**(2) (2001) 181–185
12. Lu, H., Dwarkadas, S., Cox, A.L., Zwaenepoel, W.: Message Passing versus Distributed Shared Memory on Networks of Workstations. In: Proceedings of the Conference on Supercomputing, New York, NY, USA, ACM (1995) 37
13. Protic, J., Tomasevic, M., Milutinovic, V.: Distributed Shared Memory: Concepts and Systems. IEEE Parallel e Distributed Technology **4**(2) (1997) 63–79
14. da Silva, R., Whately, L., Bentes, C., Amorim, C.: Implementação e Avaliação Preliminar de um novo Sistema Software DSM para Cluster de Computadores. In: Anais do Worshop de Computação de Alto Desempenho, Rio de Janeiro (2005) 37
15. Pedrini, H., Schwartz, W.: Análise de Imagens Digitais: Princípios, Algoritmos e Aplicações. 1th edn. Thomson Learning, São Paulo (2008)
16. Galo, M., Novo, E.: Índices de Paisagem Aplicados a Análise do Parque Estadual Morro do Diabo e Entorno. In: Anais do IX Simpósio Brasileiro de Sensoriamento Remoto, Santos, INPE/SELPER (1998) 17–28
17. Miller, D., Kaminsky, E., Rana, S.: Neural Network Classification of Remote-sensing Data. Computers and Geosciences **21**(3) (1995) 377–386
18. Gorino, F.V.R.: Balanceamento de carga em clusters de alto desempenho: Uma extensão para a lam-mpi. Master's thesis, Universidade Estadual de Maringá (1999)
19. Sacerdoti, F., Katz, M., Massie, M., Culler, D.: Wide Area Cluster Monitoring with Ganglia. In: Proceedings of the IEEE Cluster, USA, IEEE (2003) 117–123
20. Papadopoulos, P., Katz, M., Bruno, G.: NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters. Concurrency and Computation: Practice and Experience **15**(7) (2003) 707–725
21. Bradski, G., Kaehler, A.: Learning OpenCV: Computer vision with the OpenCV library. 1th edn. O'Really Media, USA (2008)