SCIENTIA V.12Nº1 (2001) 11-25

Reconhecedor de Imagens Concorrente*

Eduardo Moschetta† Fernando Santos Osório,
Gerson Geraldo H. Cavalheiro

Programa de Pós-Graduação em Computação Aplicada – PIPCA
Centro de Ciências Exatas e Tecnológicas
Universidade do Vale do Rio dos Sinos
São Leopoldo – RS – Brasil
{eduardom, osorio, gersonc}@exatas.unisinos.br

ABSTRACT

A particular application exploiting multimedia resources is the search in large databases. In some critical situations the response time of such queries must be accomplished with a minimum delay. In this work the concurrent programming is proposed as a way to solve this problem in a specific application: the query by image. The implementation of this application used the same strategies adopted to build Anahy, a high performance programming environment for clusters.

KEYWORDS parallel programming, query by image, high performance computing

RESUMO

Dentre as diversas aplicações que exploram recursos multimídia, existem aquelas que realizam consultas a grandes bases de imagens. Em alguns casos, podem ocorrer situações críticas, onde o tempo de resposta dessas consultas deve ser realizado em tempo mínimo. A programação concorrente é utilizada neste trabalho

*Apoio: CNPq, FAPERGS, UNISINOS
† PIBIC - CNPq

ISSN 0104 – 1770

para propor uma solução a esta questão em um caso de estudo. A implementação da aplicação utilizou estratégias adotadas na construção de Anahy, um ambiente de programação para o processamento de alto desempenho sobre agregados de computadores.

PALAVRAS CHAVE Programação paralela, busca de imagens, processamento de alto desempenho

1 Introdução

Nas atuais tecnologias de computação, informações textuais estão sendo cada vez mais substituídas por dados gráficos. Como conseqüência, sistemas de bancos de dados foram reformulados para armazenar esses novos tipos de informações e novos métodos de manipulação desses dados foram desenvolvidos. Nesse contexto, um problema real é, a partir de uma coleção de imagens, localizar uma imagem específica (Das, Riseman, e Draper 1997; Smith e Chang 1995). Este trabalho aborda o reconhecimento de imagens, especificamente a busca de um fragmento de imagem em outras imagens. O objetivo é oferecer um sistema de consulta e recuperação de imagens em bancos de dados a partir de fragmentos (imagens-exemplo), também conhecido como *Query by Image*. O princípio básico é percorrer todas as imagens do banco procurando, em cada uma dessas, pelo fragmento proposto.

Com o surgimento de bancos de dados cada vez maiores, há uma grande demanda por sistemas que buscam alguma informação específica no conteúdo destes. Um site de busca da Web, que procura um conjunto de palavras fornecido pelo usuário em uma coleção de textos da Internet, é um exemplo clássico desses sistemas.

Porém, imagens são informações complexas e de grande gra-

nulosidade num sistema, sendo que a busca de alguma informação nessas imagens gera uma grande carga de processamento. Além disso, com a crescente necessidade de lidar com grandes bases desses dados, tal processamento se tornaria inviável em arquiteturas com baixo grau de paralelismo.

Neste trabalho é apresentada uma solução concorrente para este problema, onde a carga total de processamento, busca do fragmento, é compartilhada entre diferentes nodos de um agregado de computadores. Este artigo discute o algoritmo de busca utilizado (seção 2), os algoritmos de *matching* aplicados (seção 3), as técnicas de balanceamento de carga desenvolvidas (seção 4) e aspectos da implementação realizada (seção 5). O artigo é finalizado com uma breve apresentação de resultados obtidos tanto na qualidade dos resultados quanto no tempo necessário para obtê-los (seção 6), tendo como conclusão (seção 7) uma discussão dos resultados obtidos com o desenvolvimento deste trabalho no contexto do projeto Anahy.

2 Método de Procura de uma Subimagem

O método de busca consiste em percorrer todas as imagens contidas em um banco de dados na tentativa de localizar as que possuam características similares a um determinado fragmento. Para tanto, o algoritmo implementa uma cabeça de leitura, de dimensões idênticas ao fragmento, que desliza-o sobre cada imagem do banco. O procedimento de leitura compara todas as regiões de cada imagem com o dado fragmento, retornando, a cada comparação, o valor de erro (diferença) encontrado no *matching* entre o fragmento e a região. Nesse instante, a região é aceita se este erro encontra-se dentro de um limite máximo especificado pelo usuário. O índice de erro depende do algoritmo utilizado para o *matching*. A seção 5 descreve como esse índice é calculado nos diferentes algoritmos implementados.

O algoritmo de procura seqüêncial é esquematizado na Figura 1. Para cada imagem do banco, é criada uma cabeça de

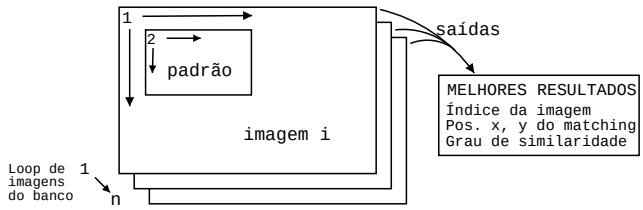

Figura 1. Algoritmo de *Matching*

leitura que desloca o fragmento pixel a pixel pela mesma, conforme mostra as setas identificadas por "1" na figura. Esse deslocamento se dá em ambas direções (da esquerda para a direita e de cima para baixo), a fim de comparar o fragmento com todas as regiões possíveis da imagem. Para cada deslocamento, é aplicado um algoritmo de *matching* que verifica a similaridade entre o fragmento e a região da imagem sobreposta pela cabeça. Na prática, esse algoritmo necessita percorrer todos os pixels do fragmento (setas indicadas por "2").

3 Algoritmos de Matching

No momento, encontram-se implementados dois algoritmos de *matching*: comparação ponto a ponto e comparação de histogramas de cores. Para ambos, assume-se que o padrão e as imagens estejam aproximadamente na mesma escala.

Na comparação ponto a ponto (pixel a pixel), o *matching* se resume em comparar todos os pixels do fragmento com os pixels da região da imagem sobreposta pela cabeça de leitura. A principal vantagem desse método é a sua simplicidade e o fato de ser desnecessário manter informações adicionais no processamento. Entretanto, sua capacidade de reconhecimento é bastante limitada: para haver um *matching*, uma região da imagem deve ser igual ou extremamente semelhante ao fragmento procurado.

O outro método de *matching*, comparação de histogramas de

cores, é mais utilizado que o primeiro na área de reconhecimento. Um histograma de cor demonstra a distribuição de cores da imagem, onde associa-se a cada itensidade de cor "c" presente na imagem, a sua freqüência de ocorrência, ou seja, o número de pixels da imagem que utilizam a cor "c" (Gomes e Velho 1994).

Na comparação de histogramas, a posição dos pixels não é tão relevante como na comparação ponto a ponto, mas sim o conjunto total de cores da região. Por isso, esse método retorna muito mais resultados que o primeiro, uma vez que fragmentos comparados não precisam ser extremamente semelhantes. Em contrapartida, sua versatilidade conduz muitas vezes a resultados falsos, além de consumir mais memória que a comparação ponto a ponto, em decorrência da manutenção de histogramas na memória – histogramas podem consumir bastante memória, sendo que seu tamanho depende do sistema de cores utilizado (RGB ou HSV).

Seu funcionamento é simples: a cada deslocamento da cabeça de leitura, gera-se o histograma da região sobreposta pela cabeça comparando-o com o do fragmento. A execução desse algoritmo é muito mais rápida que o algoritmo ponto a ponto, visto que, na implementação desse algoritmo, dados calculados em um movimento da cabeça de leitura são aproveitados pelo *matching* seguinte (reaproveitamento dos cálculos realizados apenas atualizando-se as bordas). Com isso, permite-se concluir que, quanto maior o fragmento, mais rápida será a execução, na qual mais dados podem ser aproveitados em um menor número de deslocamentos.

Quanto à comparação ponto a ponto, pode-se dizer que sua execução é mais lenta. O total de comparações a serem feitas, dado um fragmento de dimensões (w_f x h_f) e uma imagem a ser consultada de dimensões (w_i x h_i), pode ser previamente calculado como sendo:

$$total = (w_f * h_f) * |w_i - w_f| * |h_i - h_f| \qquad (1)$$

ou seja, para um fragmento de tamanho (270 x 270) e uma imagem de dimensões (540 x 540), por exemlo, teria-se 5.314.410.000 comparações a serem executadas.

4 Escolanamento de Tarefas

Na aplicação descrita, podem ser identificadas diferentes fontes de custo computacional (considerando o tempo de execução). A primeira delas está ligada ao tempo de execução do algoritmo de procura, proporcional ao número de imagens do banco e ao tamanho (em pixels) destas. A segunda, ligada à carga de processamento gerada por um determinado algoritmo de *matching* entre um fragmento e uma região, a qual depende também do tamanho (em pixels) do fragmento. Somando-se esses dois custos, dado um fragmento e uma base de imagens, tem-se o trabalho total da aplicação – tempo total de execução.

Visto que esse tipo de aplicação gera uma grande carga computacional, torna-se bastante atrativa e viável a idéia de utilizar técnicas de programação concorrente para aumentar o desempenho da mesma. A implementação concorrente realizada (Moschetta, Osório, e Cavalheiro 2002) considera diferentes níveis de concorrência que podem ser explorados nesta aplicação, sendo apresentados nas subsecões seguintes.

4.1 Escalonamento Entre-nós

O primeiro nível, com uma granulosidade mais grossa, explora o fato de que a busca de um fragmento em uma imagem independe da busca do mesmo em outra imagem. Logo, várias buscas de um mesmo fragmento podem ser executadas em diferentes imagens simultaneamente. Na aplicação, esse nível determina um índice de granulosidade que indica o número de imagens que um nó deve explorar na busca do fragmento proposto.

Na implementação realizada, essa granulosidade é inicialmente distribuída entre os nodos do agregado, dividindo-se o banco de imagens e, conseqüentemente, definindo um subconjunto diferente de imagens para cada nó. Assim, cada nó fica responsável pela busca do fragmento em seu subconjunto, sendo que sua execução evolui independentemente da execução em outros nodos.

Nota-se que o escalonamento inicial entre os nodos é feito de forma estática. Porém, deve-se considerar que esse, muitas vezes,

ocasiona uma distribuição irregular das tarefas, em decorrência do fato de bancos possuírem imagens de tamanhos variados. Logo, em um instante da execução, nodos podem estar atrasados em seu processamento por causa de sua carga de trabalho maior, enquanto outros nodos estarão ociosos por já terem acabado a busca. Essa situação também pode acontecer quando a aplicação é executada sobre uma arquitetura com nodos heterogêneos – possuindo diferentes velocidades de processamento.

Com vistas a resolver esse problema, a aplicação emprega balanceamento de carga em tempo de execução para que essa irregularidade, quando detectada, seja tratada. A solução adotada consistiu em definir um nó mestre que se encarrega apenas de controlar o fluxo de execução das tarefas, executando um algoritmo de controle. Esse controle é feito através de trocas de mensagens entre nodos escravos e mestre antes e depois do processamento de uma imagem pelo escravo. Sendo assim, no decorrer da execução, o nó mestre tem informações suficientes sobre o estado global da busca, podendo facilmente "migrar" imagens de um nó para outro ao detectar que esse está livre para execução de novos cálculos.

4.2 Escalonamento Intra-nó

No segundo nível, de granulosidade mais fina e interna ao nó, são consideradas as operações de *matching* do fragmento com diferentes regiões de diferentes imagens. Essas operações podem ser executadas de forma independente, podendo-se concluir que várias comparações simultâneas podem estar sendo processadas em uma ou mais imagens do nó.

Nessa implementação, visando explorar a concorrência interna do nó, cria-se, no início da busca, um *pool* de execução composto por várias cabeças de leitura (*threads*), que podem percorrer diferentes regiões, de diferentes imagens, de forma concorrente. *Threads* realizam cálculos definidos através de tarefas, as quais são geradas ao longo da execução do algoritmo de procura. Ao terminar o respectivo cálculo, *threads* retornam os resultados das operações de *matching* e tornam-se ociosas, esperando por novas tarefas.

Essa concorrência, além de aumentar a disponibilidade dos pro-

cessadores ao cálculo (em especial se a arquitetura do nó for SMP, como é o caso), permite sobrepor com cálculo efetivo parte das perdas ocorridas durante as comunicações (Cavalheiro 2001).

O grafo que representa a execução da aplicação (em ambos níveis de granulosidade) é apresentado na Figura 2, em uma arquitetura composta por n nodos. As sincronizações internas são necessárias para obter todas soluções encontradas pelas cabeças de leitura do *pool* de execução em diferentes tarefas (regiões de imagem). Igualmente, a sincronização interna, necessária apenas uma vez, permite que os nodos retornem a um nó mestre os resultados obtidos em seus subconjuntos de imagens.

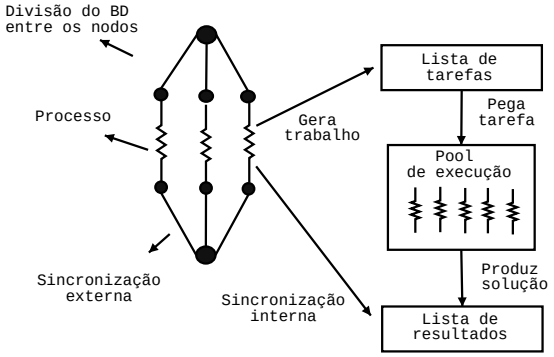

Figura 2. Modelo de Implementação

5 Implementação

Essa seção encontra-se dividida em duas partes: a primeira, relativa a aspectos da implementação concorrente, trata sobre algumas escolhas feitas para o modelo de concorrência da aplicação, esclarecendo também alguns pontos já descritos anteriormente; a segunda, restrita a aspectos gráficos da aplicação, aborda alguns detalhes da implementação dos algoritmos de *matching* para ambos modelos de cores (HSV e RGB).

5.1 Implementação concorrente

Na implementação realizada, considera-se que o banco de imagens seja acessível a todos os nodos do agregado. No início da busca, cada imagem do banco possui um identificador global (númerico) que a identifica em qualquer nó do agregado. Desse modo, o balanceamento de carga resume-se apenas a transferência de identificadores de imagens entre os nodos, sendo feito de forma rápida e eficiente. Outra vantagem é que a abertura da imagem na memória pode ser postergada até o ínicio da busca do fragmento na mesma, evitando que imagens sejam abertas de forma desnecessária, caso seja preciso migrá-las futuramente para outro nó.

Outro detalhe importante é quanto ao envio de resultados do escravo ao mestre. Esse evento ocorre não só no final do algoritmo de procura, mas a cada comunicação do escravo ao mestre, enviando-lhe resultados obtidos até o momento. Isso visa aproveitar a comunicação necessária com o mestre e evitar uma comunicação final com uma grande quantidade de dados. Essa comunicação é feita de forma assíncrona, de modo que o processo possa começar logo em seguida a busca do fragmento na imagem corrente.

Como ferramentas para implementar esses recursos, foram utilizadas MPI (*Message Passing Interface*), para comunicação interprocessos, e a biblioteca *Pthreads*, a fim de prover a concorrência leve (*multithreading*).

5.2 Algoritmos de *matching* e modelos de cores

Na implementação atual, o histograma de cores para RGB é implementado através de três vetores (R, G e B), sendo que cada um contém 256 entradas a fim de manter as freqüências de ocorrências das cores representadas pelos índices dos vetores. Nota-se que o histograma é uma estrutura de dados grande (possui 768 entradas), o que faz com que a memória seja um fator limitante nesse algoritmo, ainda mais que esse número aumenta proporcionalmente ao número de cabeças de leitura ativas no momento. Na comparação ponto a ponto, nenhuma estrutura de dados adicional

é necessária para sua execução.

O reaproveitamento de cálculo de sucessivos deslocamentos da cabeça de leitura na comparação de histogramas (descrito anteriormente) consiste em, dado um histograma inicial (calculado na primeira região a qual a cabeça de leitura é responsável por percorrer), decrementar a freqüência dos pixels que aparecem na primeira coluna/linha da região anterior e acrescentar a contagem dos pixels que aparecem na última coluna/linha da região corrente. Colunas são utilizadas como pontos de referência no deslocamento horizontal (uma coluna à direita), enquanto que linhas o são no deslocamento vertical (uma linha abaixo). Esse reaproveitamento não é possível na comparação ponto a ponto, o que justifica o seu maior tempo de execução.

Quanto ao cálculo da similaridade entre o fragmento e uma região da imagem, o método de obtenção desse difere entre os dois algoritmos de *matching*. Na comparação ponto a ponto, considera-se apenas a soma dos valores absolutos das diferenças entre as cores (cada cor possuindo um valor correspondente) dos pixels comparados (Jain, Kasturi, e Schunck 1995), através da fórmula

$$soma = \sum_{l=1}^{nLin} \sum_{c=1}^{nCol} \begin{array}{l} |corR_{pixel\_fragmento} - corR_{pixel\_imagem}| + \\ |corG_{pixel\_fragmento} - corG_{pixel\_imagem}| + \\ |corB_{pixel\_fragmento} - corB_{pixel\_imagem}| \end{array} \quad (2)$$

No caso da comparação de histogramas, a similaridade pode ser calculada considerando-se a diferença entre os histogramas, ou seja, a soma dos valores absolutos das diferenças entre as freqüências de todos valores nos vetores R, G e B:

$$soma = \sum_{i=0}^{255} \begin{array}{l} |freq_{R[i]\ fragmento} - freq_{R[i]\ img.\ BD}| + \\ |freq_{G[i]\ fragmento} - freq_{G[i]\ img.\ BD}| + \\ |freq_{B[i]\ fragmento} - freq_{B[i]\ img.\ BD}| \end{array} \quad (3)$$

onde i é o índice para os três vetores.

No momento, o reconhecedor também oferece suporte ao espaço de cor HSV (*Hue*, *Saturation* e *Value*). Nesse modelo, o H repre-

senta o matiz, S a saturação e V o brilho. Esse modelo é mais eficiente que o RGB, sendo mais adequado para medir a similaridade entre cores. O método de cálculo de similaridade bem como as estruturas adicionais (histogramas) são implementadas de forma semelhante nesse modelo. Contudo, esse modelo não é reconhecido em hardware, logo deve-se acrescentar um nível de conversão $RGB \rightarrow HSV$, degradando o tempo de execução do algoritmo de *matching*.

6 Resultados Práticos

A fim de validar-se aspectos da aplicação descrita nas seções anteriores, foi realizada uma série de experimentos. Esses foram divididos em duas categorias: a primeira, voltada a determinação da qualidade dos resultados de *matching* e a segunda à análise do desempenho.

6.1 Qualidade do *matching*

Para verificar a qualidade da implementação dos algoritmos de *matching*, foi utilizado um banco de imagens criado pelo Departamento de Ciências da Computação da Universidade de Columbia (Nene, Nayar, e Murase 1996). Essa base, já utilizada em alguns sistemas de reconhecimento de imagens, contém fotos coloridas de 100 objetos, todas com dimensões (128 x 128). No processo de geração dessas imagens, cada objeto foi fotografado em 72 ângulos diferentes, com um giro de 5^o entre cada foto.

Nesse artigo, encontra-se documentada a busca de cinco diferentes objetos, sendo que, nesse experimento, o fragmento é consirado como sendo a própria foto do objeto no ângulo 0^o. Dado um determinado objeto, o objetivo é localizar todas as fotos em que esse objeto aparece, independente de sua orientação espacial.

A Tabela 1 resume os resultados obtidos. As linhas desta tabela correspondem aos casos de estudo (os cinco objetos selecionados). Na primeira coluna encontram-se as descrições de cada

objeto utilizado no experimento, seguidas pela sua identificação númerica no banco. A segunda e terceira colunas apresentam os índices de sucesso na localização do objeto utilizando os dois critérios de *matching* (respectivamente comparação ponto a ponto e histogramas) com suas respectivas taxas de acerto. Os resultados para comparação ponto a ponto foram obtidos para um erro máximo de 30, enquanto que os resultados da comparação de histogramas para um erro máximo de 2.

Tabela 1. Análise da qualidade dos resultados obtidos

Imagem	Ponto a Ponto	Histogramas
Xícara rosa (25)	100%	100%
Pêra (83)	48,6%	97,2%
Refrigerante (62)	6,9%	87,5%
Carro verde (27)	8,3%	40,3%
Caixa de remédio (54)	2,8%	76,4%

Dado os resultados obtidos, nota-se que a aplicação desenvolvida apresentou um bom nível de qualidade de *matching* para esses objetos utilizando-se o algoritmo de comparação de histogramas. Até mesmo o carro e caixa de remédio, caracterizados por sua assimetria, apresentaram um *matching* razoável. A lata de refrigerante, mesmo possuindo uma embalagem com desenhos detalhados, apresentou um ótimo nível de reconhecimento resultante do fato de possuir uma determinada cor predominante (vermelho). Porém, deve-se considerar que, na busca desse objeto, obteve-se muitos resultados falsos, uma vez que no banco havia outra latas de refrigerantes com forma e distribuição de cores semelhantes.

6.2 Avaliação de Desempenho

A segunda série de experimentos realizados considerou o desempenho da aplicação em relação ao tempo de processamento. Esses experimentos foram realizados em um agregado composto de quatro nodos bi-processados (2 × PIII 800, 2 × PIII 1 GHz, cada nodo com 512 Mb RAM, Ethernet 100).

As características deste conjunto de experimentos encontram-se relacionados abaixo:

- Banco de dados: o banco de imagens utilizado consiste em 505 fotografias de tamanhos variados (tendo a maior dimensões 650x721 e a menor 192x639); as imagens possuem dimensões médias de 571x583 pixels, com um desvio-padrão de 150x154 pixels.

- Fragmento: com vistas a análisar o comportamento da execução sob diferentes granulosidades, foram selecionados fragmentos de três tamanhos, pequeno (135 x 135), médio (272 x 272) e grande (540 x 540).

A Tabela 2 resume esses experimentos, para ambos algoritmos de *matching*, comparando-os com o mesmo processo sendo realizado numa máquina monoprocessada. As linhas da tabela referem-se aos três tamanhos de fragmentos propostos. A coluna `Seq` mostra uma estimativa do tempo necessário para processar a busca desses fragmentos no banco de imagens em uma máquina monoprocessada, calculada com base na busca em uma imagem de tamanho 572x574 pixels (caso médio no referido banco). Na seqüência, a coluna (`Conc`) apresenta o tempo da busca desses fragmentos em um agregado e a terceira compara as duas primeiras colunas, mostrando o ganho obtido na execução concorrente. Para essa, foram utilizadas 50 cabeças de leitura, sendo que cada uma era responsável por percorrer no máximo 10 linhas de uma imagem.

Tabela 2. Desempenho em um agregado de computadores

Alg. mat.	Ponto a Ponto			Histogramas		
Fragmento	Seq.	Conc.	Ganho	Seq.	Conc.	Ganho
Pequeno	372.185 s	51.885 s	7,17	4.545 s	607 s	7,49
Médio	787.800 s	107.949 s	7,30	3.535 s	479 s	7,37
Grande	39.390 s	2.340 s	16,83	404 s	90 s	4,49

Nota-se que os ganhos foram bons e praticamente iguais para os fragmentos pequeno e médio, o que valida o algoritmo de balanceamento de carga, por ser capaz de realizar um bom balanceamento sob diferentes comportamentos e tempos de execução. No

caso do fragmento grande, os resultados exibidos são apenas indicativos, uma vez que imagens são descartadas e parte do cálculo assumido como trabalho não é efetivado. Uma imagem é descartada do processamento quando é menor que o fragmento em uma das dimensões, acelerando o processo de busca.

7 Conclusão

Neste artigo foi apresentada uma implementação concorrente de um algoritmo de busca de imagens. Desta implementação, foram destacados diferentes aspectos, tais como os diferentes níveis de granulosidade de cálculo e estratégias de exploração do hardware disponível. Também foram apresentados diferentes algoritmos de *matching* e como esses influenciam no comportamento da aplicação. Os resultados de desempenho obtidos são encorajadores, motivando a continuidade dos trabalhos.

Um aspecto importante a ressaltar é que, muito embora o problema tratado tenha interesse real, a implementação foi realizada com o objetivo de validar conceitos e estratégias adotadas na implementação de um ambiente de programação concorrente: Anahy. Anahy está sendo desenvolvido (Garzão, Villa'Real, e Cavalheiro 2001; Villa'Real, Dall'Agnol, e Cavalheiro 2002) com o propósito de oferecer ao programador um ambiente de processamento de alto desempenho, dotado de uma interface aplicativa e de um núcleo de escalonamento. As ferramentas utilizadas para implementação do reconhecedor de imagens, *threads* POSIX e MPI, são as atualmente utilizadas na implementação deste ambiente.

Em um outro sentido, a realização da implementação apontou aspectos que deverão ser considerados quando da introdução de extensões ao atual modelo de Anahy. Também, pôde ser observada a necessidade de integrar novas ferramentas no reconhecedor de imagens. Entre essas, considera-se atualmente novos algoritmos de *matching* e um algoritmo para eliminar resultados falsos da procura de um fragmento.

Referências

Cavalheiro, G. G. H. (2001, Janeiro). Introdução à programação paralela e distribuída. In Tiarajú A. Diverio e Philippe Navaux (Eds.), *1 Escola Regional de Alto Desempenho*. Gramado.

Das, M., E. Riseman, e B. Draper (1997). Focus: Searching for multi-colored objects in a diverse image database. In *IEEE Conf. on Comp. Vis. and Pattern Recognition*. Porto Rico.

Garzão, A. S., L. C. Villa'Real, e G. G. H. Cavalheiro (2001, Maio). Ferramentas para desenvolvimento de um ambiente de programação sobre agregados. In *Anais do Workshop em Software Livre*. Porto Alegre, Brasil.

Gomes, J. e L. Velho (1994). *Computação Gráfica: Imagem*. Rio de Janeiro, Brasil: IMPA/SBM.

Jain, R., R. Kasturi, e B. G. Schunck (1995). *Machine Vision*. Singapore: McGraw-Hill International Editions.

Moschetta, E., F. S. Osório, e G. G. H. Cavalheiro (2002, Janeiro). Reconhecedor de imagens usando técnicas de alto desempenho. In Tiarajú A. Diverio e Gerson G. H. Cavalheiro (Eds.), *2 Escola Regional de Alto Desempenho*. São Leopoldo.

Nene, S., S. Nayar, e H. Murase (1996). Columbia object image library: Coil. Technical Report CUCS-006-96, Columbia University.

Smith, J. R. e S. Chang (1995, Julho). Tr: Automated image retrieval using color and texture. Technical Report 414-95-20, Columbia University.

Villa'Real, L. C., E. C. Dall'Agnol, e G. G. H. Cavalheiro (2002, Janeiro). Construção de um ambiente de programação para o processamento de alto desempenho. In Tiarajú A. Diverio e Gerson G. H. Cavalheiro (Eds.), *2 Escola Regional de Alto Desempenho*. São Leopoldo.