Reconhecimento de Imagens em Aplicações Críticas*

Eduardo Moschetta† Fernando S. Osório Gerson Geraldo H. Cavalheiro

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

Resumo

Muitas aplicações multimídia oferecem sistemas de consultas de imagens-exemplo em grandes bases de imagens. A maioria dessas aplicações são críticas, ou seja, requerem um tempo de resposta mínimo. Este artigo apresenta uma solução para esse problema através da programação concorrente. Proposta inicialmente com o objetivo de testar e validar o ambiente de programação Anahy, a implementação realizada tem como foco principal minimizar esse tempo de resposta. Para sua compreensão, são apresentados detalhes da implementação concorrente e métodos de escalonamento de tarefas aplicados, bem como conceitos básicos sobre busca de imagens. Ainda, para validar a solução proposta, é apresentada uma análise de desempenho feita para medir tanto a qualidade do matching *quanto o tempo de resposta da aplicação no reconhecimento de imagens.*

1 Introdução

A era na qual vivemos caracteriza-se por incessantes buscas de informações. Os novos recursos de multimídia e Internet tem possibilitado a substituição das tradicionais informações textuais por representações sob forma de imagens. A questão que se coloca é a forma de recuperação destas informações, sobretudo quando estas encontram-se inseridas no contexto de aplicações onde o tempo de reação é considerado crítico.

Exemplos destas classes de aplicações podem ser encontrados nos mais amplos domínios. Um exemplo clássico pode ser representado pela busca de uma determinada imagem em um banco de imagens. Nesse caso, o critério de busca leva em consideração um fragmento de imagem, sendo o objetivo localizar imagens que contenham características similares presentes nesse fragmento, como cor, forma ou textura. Alguns sistemas de recuperação de imagens, também chamados de "Content Based Image Retrieval" (CBIR) ou "query by example" (QBE), podem ser encontrados em [16], [12], [4] e [15]. Outros exemplos podem ser retirados de aplicações em tempo real, como a localização de uma bagagem estraviada em um aeroporto ou identificação de possíveis alvos bélicos (i.e., identificar a presença de aviões militares em determinadas regiões do espaço aéreo).

Dentre os problemas que envolvem esta nova forma de manipulação de informação, encontra-se o alto custo computacional necessário para realizá-la, sendo que este custo pode ser medido tanto em capacidade de processamento quanto em quantidade de memória necessária para armazena mento das informações. Uma solução para esta questão pode ser buscada na utilização do processamento paralelo.

Técnicas de recuperação de imagens vem sendo desenvolvidas desde a década de 1970, porém mais recentemente começaram a ser realizados trabalhos [5] [3] que objetivam minimizar o alto custo computacional gerado por esses sistemas, aplicando técnicas de programação concorrente. Neste trabalho é apresentada a implementação de uma solução concorrente [10, 9] para o problema discutido acima. Mais especificamente, da busca de uma imagem padrão (frag mento) em imagens pertencentes a um banco de imagens. Na implementação proposta são consideradas apenas questões envolvendo o tempo de processamento, não outros custos, tais como consumo de memória (disco). Esta implementação foi realizada sob arquiteturas do tipo SMP (*Symmetric Multi-Processors*) e agregados de computadores.

A abordagem adotada consiste na divisão do trabalho de busca entre os processadores. Para tanto, são considerados dois níveis de granularidade de trabalho: o maior, caracterizado pela divisão do banco de imagens entre os nodos, e o segundo, de granularidade mais fina, que consiste na busca do fragmento em uma determinada região de cada imagem, utilizando paralelismo hibrído para tratar ambos níveis de

*Projeto Anahy. CNPq/Protem-CC (68.0120/01-7), FAPERGS (00/2740-8) e UNISINOS (37.00.001/00-0).

†PIBIC/CNPq

granularidade.

O restante deste trabalho é organizado como segue. A seção 2 apresenta o método de busca e os algoritmos de localização de fragmentos. Na seqüência, a seção 3 discute o modelo da implementação concorrente adotado e a seção 4, o algoritmo de balanceamento de carga implementado. O trabalho é finalizado com uma avaliação do desempenho da aplicação (seção 5), tendo como conclusão (seção 6) uma discussão dos resultados obtidos com o desenvolvimento deste trabalho no contexto do projeto Anahy.

2 Método de Busca

O método de busca consiste em percorrer todas as imagens contidas no banco de dados na tentativa de localizar as imagens que possuam características similares a um determinado fragmento. Para tanto, o algoritmo implementa uma cabeça de leitura, de dimensões idênticas ao fragmento, que desliza sobre cada imagem do banco de imagens. O procedimento de leitura compara todas as regiões de cada imagem com o dado fragmento. Para cada comparação, o algoritmo retorna o valor de erro (diferença) encontrado no *matching* entre o fragmento e a região – a região é aceita caso este erro encontre-se dentro do limite de erro máximo aceito pelo usuário.

O valor de erro é determinado pelo cálculo de similaridade entre as regiões. O método de cálculo difere entre os possíveis algoritmos de *matching*. No caso mais simples, onde adota-se uma comparação ponto a ponto, considera-se a soma das distâncias das cores entre os pixels [8].

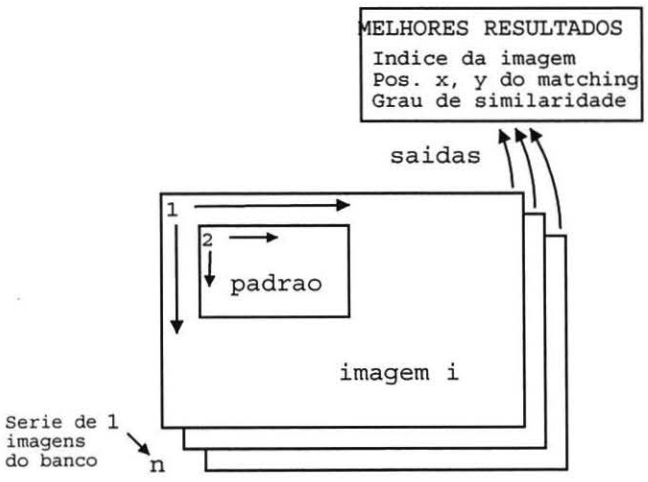

Figura 1. Algoritmo de Procura

O algoritmo de procura seqüêncial é esquematizado na Figura 1. Nesse algoritmo, as imagens são processadas uma a uma. Para cada imagem do banco, é criada uma cabeça de leitura (janela deslizante) que desloca o fragmento, pixel a pixel, pela imagem conforme direções identificadas por "1" na figura. Para cada deslocamento dessa cabeça, aplica-se um algoritmo de *matching* para verificar a similaridade entre o fragmento e a região da imagem sobreposta pela janela, conforme indica "2" na figura. O deslocamento da cabeça de leitura na imagem se dá em ambas direções (da esquerda para a direita e de cima para baixo), a fim de percorrer todos os blocos possíveis da mesma.

Existem inúmeras técnicas de *matching*: Antani [1] e Rui [14] apresentam uma pesquisa dos principais sistemas e técnicas utilizadas atualmente nas tarefas de indexação e recuperação de imagens. Para esse trabalho, foram implementados apenas dois algoritmos de *matching*: comparação ponto a ponto e comparação de histogramas de cores. Ambos algoritmos necessitam percorrer todos os pixels da região da imagem sobreposta pela cabeça, deixando visível a grande carga computacional gerada por estes e, consequentemente, pelo algoritmo de procura do fragmento. Algumas medidas de desempenho são mostradas na seção 5.

Quanto à qualidade de *matching* gerada por esses algoritmos, pode-se dizer que é boa para determinados tipos de aplicações. A comparação ponto a ponto permite encontrar uma imagem que contenha um fragmento idêntico ou extremamente similar ao procurado, enquanto que a comparação de histogramas prioriza uma semelhança de cores entre dois fragmentos, que é determinada por uma comum distribuição das cores de suas regiões. O histograma de cores representa a contagem do número de ocorrências de cada cor presente em uma determinada região [7]. No caso da 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.

3 Implementação Concorrente

Na aplicação descrita na seção anterior, podem ser identificadas diferentes fontes de custo computacional (considerando o tempo de execução). A primeira delas está ligada ao número de imagens do banco de imagens e ao tamanho (em pixels) destas. A segunda, à carga de processamento gerada pelo próprio algoritmo de *matching* entre um fragmento e uma região, o qual depende também do tamanho (em pixels) do fragmento. O somatório destes custos, dado um fragmento e um banco de imagens, corresponde ao trabalho total da aplicação – tempo total de execução.

Visto que esse tipo de aplicação gera uma grande carga de processamento, é viável que se utilize técnicas de programação concorrente para aumentar o desempenho da mesma. A implementação concorrente realizada considera diferentes níveis de concorrência que podem ser explorados nesta aplicação. Um primeiro nível explora o fato de que a busca de um fragmento em uma imagem independe da busca desse

mesmo fragmento em outra imagem. Assim, pode-se ter várias buscas de um fragmento em diferentes imagens simultaneamente. Em um nível mais fino, são consideradas as operações de *matching* do fragmento com as diferentes regiões de uma imagem. O resultado de cada uma destas operações também é computado de forma independente.

Estes dois níveis de concorrência definem dois índices de granulosidade para as atividades do programa em execução. O primeiro, mais grosso, determina o número de imagens que um nó deve explorar na busca do fragmento proposto. O segundo, mais fino e interno ao nó, o número de comparações simultâneas que podem estar sendo processadas em uma ou mais imagens do nó.

Na implementação proposta, a granulosidade grossa é inicialmente distribuída entre os nodos, dividindo-se o banco de imagens e, consequentemente, definindo um subconjunto diferente de imagens para cada um desses nodos. Para tanto, cada nó fica responsável pela busca do padrão em seu subconjunto de imagens, sendo que sua execução evolui independentemente da execução em outros nodos.

Para a granulosidade fina, visando explorar a concorrência interna do nó, cria-se um *pool* de execução, composto por várias cabeças de leitura (*threads*), que podem percorrer diferentes regiões, de diferentes imagens, de forma concorrente. Essa concorrência, além de aumentar a disponibilidade dos processadores ao cálculo (em especial se a arquitetura do nó for SMP, como é o nosso caso), permite sobrepor com cálculo efetivo parte das perdas ocorridas durante a comunicação entre nodos [2]. A necessidade de comunicações entre nodos durante a execução da busca é explicada a seguir (seção 4).

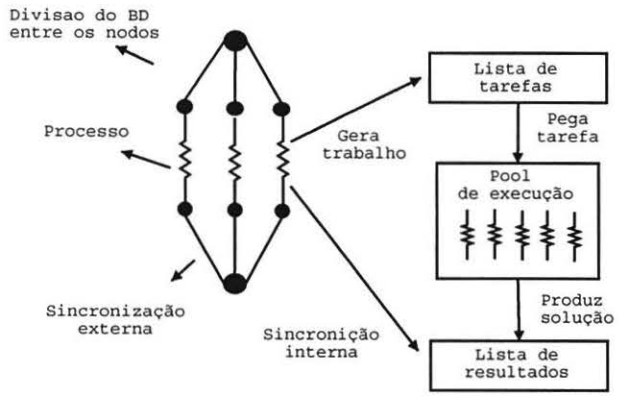

Figura 2. Modelo de Concorrência

O grafo que representa a execução da aplicação é apresentando na Figura 2, em uma arquitetura composta po *n* nodos. A figura destaca a existência, em cada nó, de um *pool* de execução composto por *threads*, responsáveis por realizar as operações de *matching*. As sincronizações internas são necessárias para obter todas soluções encontradas pelas cabeças de leitura em diferentes tarefas (regiões de imagem). Igualmente, a sincronização externa, necessária apenas uma vez, permite que os nodos retornam a um nó mestre os resultados obtidos em seus subconjuntos de imagens.

Os algoritmos mostrados a seguir correspondem, respectivamente, ao algoritmo de controle (1) empregado pelo nó mestre e o algoritmo de cálculo (2) utilizado pelos nodos escravos.

Algoritmo 1 Algoritmo de Controle

```
distribui_carga(); //distribui carga entre nodos
....
for ( x=0; x < imagecount; x++ ) {
/* indica que um nó começou a processar imagem,
também recebe resultados disponíveis desse nó até o momento */
recebe_dados();
....
/* atualiza estruturas de controle */
atualiza_estado();
/* gera ação (retira imagem, migra imagem, ...) */
gera_ação();
....
/* responde com uma ação ao nó que o enviou */
envia_ação();
}
recebe_resultados(); //recebe restante dos resultados
```

Algoritmo 2 Algoritmo de Unidade de Trabalho

```
/* enquanto houver imagens a processar */
while ( not imagelist.empty() ) {
/* envia resultados obtidos ate o momento ao mestre */
envia_dados();
....
/* processa imagem (criação das cabeças de leitura,
sincronização das cabeças etc.) */
processa_imagem();
....
recebe_ação(); //recebe ação do mestre
executa_ação(); //executa acão recebida
}
envia_resultados(); //envia restante dos resultados
```

As ferramentas utilizadas na implementação desses algoritmos foram as bibliotecas MPI (*Message Passing Interface*) para realizar as comunicações entre nodos e *threads* POSIX para explorar a concorrência interna de cada nó.

4 Escalonamento de Tarefas

O escalonamento inicial das tarefas (imagens) entre os nodos da arquitetura é feito de forma estática. Porém, este balanceamento inicial pode distribuir a carga de modo irregular, visto que o fator de balanceamento assumido é a quantidade de imagens, e não o tamanho destas. Logo, apesar de poder haver uma igual distribuição das imagens pela arquitetura, determinados nodos poderão ter subconjuntos com imagens maiores, ou seja, uma carga de trabalho maior. Também deve ser considerado que a capacidade de processamento também pode variar entre os nodos, caso estes sejam heterogêneos – processadores com velocidade de processamento diferentes.

Com vistas a resolver esse problema, a aplicação realiza balanceamento de carga em tempo de execução para que essa irregularidade, quanto 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 (algoritmo 1). Durante a evolução global da aplicação, nodos podem terminar seu processamento fazendo com que o algoritmo de balanceamento de carga detecte esse evento. Esse verifica o estado da execução e, caso haja imagens não-processadas em outros nodos, supervisiona uma migração de imagens entre os nodos.

Na implementação realizada, optou-se por tornar as imagens (tarefas) acessíveis a todos nodos, sendo que cada uma possui um identificador global (numérico) que a localiza na aplicação. Portanto, tanto a migração de imagens como a distribuição inicial dessas nos nodos manipula apenas identificadores de imagens, e não com a imagem em si. A abertura da imagem na memória é postergada até o início da sua execução, a fim de que, caso futuramente possa ser enviada a outro nodo, não seja aberta desnecessariamente aumentando o custo de processamento.

O balanceamento de carga da aplicação é realizado com base nas trocas de informações entre mestre e escravo a cada processamento de uma imagem. Isso permite que o mestre tenha conhecimento do estado global de execução da busca, e aja sempre a favor dele.

Mais especificamente, esse balanceamento ocorre da seguinte maneira (algoritmos 1 e 2): o nó escravo, antes de começar o processamento de uma imagem, envia dados ao mestre para que esse o identifique. O mestre recebe os dados, identifica o nó que o enviou e atualiza seu estado, basicamente decrementando o número de imagens a serem ainda processadas pelo mesmo. Na seqüência, pesquisa o estado do nó (e dos outros nodos, caso necessário), gerando uma determinada ação. Logo após, envia essa ação para o nó e volta à espera de uma nova comunicação.

A ação enviada determina a próxima operação do nó escravo que enviou os dados, sendo avaliada pelo mesmo assim que terminar o processamento da imagem corrente. Essa ação pode conter três possíveis valores (e conseqüências):

1. Caso o mestre detectou que o nó não possui mais imagens a processar, busca-se o nó (vítima) com maior número de imagens não-processadas. Se for encontrada, é atualizado o estado da vítima no mestre, e o valor da ação consistirá no identificador da imagem roubada.

2. Se um nó foi vítima no balanceamento anteriormente e possui um número x de imagens roubadas do seu subconjunto, o valor da ação é este número x.

3. Se nenhuma dessas duas condições é verificada, a ação fica com um valor nulo.

Ao terminar de processar uma imagem, a expectativa é que o nó escravo tenha disponível a próxima ação a ser executada. A hipótese assumida é a sobreposição pelos tempos de cálculo (realização do algoritmo de procura) das operações de balanceamento de carga e comunicações. Recebendo a ação, o nó pode adicionar a imagem x ao seu subconjunto de imagens (caso 1), retirar n imagens de seu subconjunto (caso 2) ou continuar a execução normalmente (caso 3). Para esse último caso, a execução do algoritmo continuará somente se houver ainda imagens a processar.

Quanto ao envio de resultados, esses são os dados que o escravo envia ao mestre a cada ínicio do processamento de uma imagem. 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.

O escalonamento de grãos finos consiste em, ao longo da execução do algoritmo de procura, gerar tarefas que vão ser executadas por diferentes cabeças de leitura. Caso não haja uma cabeça de leitura disponível no momento, a execução de um trabalho é sincronizada com o término da cabeça que mais tempo está executando determinada tarefa.

Após sua lista de imagens não-processadas tornar-se vazia (algoritmo 2), o nó sincroniza todas cabeças de leitura ativas e envia o restante dos resultados ao mestre. Do mesmo modo, após o término do algoritmo de controle, o mestre sincroniza-se com todos os nodos a fim de receber esses resultados enviados.

5 Resultados Práticos

Para a validação da aplicação descrita nas seções anteriores, foi realizada uma série de experimentos sobre diferentes arquiteturas. Os experimentos foram divididos em duas categorias: a primeira voltada à determinação da qualidade

dos resultados de *matching* e a segunda à análise do desempenho.

Os experimentos foram realizados utilizando uma arquitetura SMP (2 processadores PIII 1 GHz, 512 Mb RAM) e sobre um agregado composto de quatro nodos bi-processados (2 × PIII 600 MHz, 2 × PIII 1 GHz, cada nodo com 512 Mb RAM, Ethernet 100). A título comparativo também foram realizados experimentos em uma arquitetura monoprocessada (PIII 1 GHz, 512 Mb RAM).

5.1 Qualidade de 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 [11]. Essa base, já utilizada em alguns sistemas de reconhecimento de imagens, contém fotos coloridas de 100 objetos. No processo de geração dessas imagens, cada objeto foi fotografado em 72 ângulos diferentes, com um giro de 5^o entre cada foto. Logo, o banco de imagens possui no total 7200 imagens, sendo que todas imagens possuem a mesma dimensão (128 x 128).

Nesse artigo, encontra-se documentada a busca de cinco diferentes objetos – nesse experimento, o fragmento é 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	P a P	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. A aplicação conseguiu reconhecer objetos mesmo quando estes possuíam uma forma assimétrica, muito embora as taxas de reconhecimento de objetos simétricos tenham sido mais elevadas. A xícara, a pêra e a lata de refrigerante apresentaram um ótimo nível de *matching* decorrente da sua forma regular sob diferentes ângulos. A última, mesmo possuindo uma embalagem com desenhos detalhados, apresentou um ótimo nível de reconhecimento resultante do fato de possuir uma determinada cor predominante (vermelho). Já o carro e a caixa de remédi apresentaram um índice apenas razoável de *matching*, dado o índice de assimetria que apresentam.

Na obtenção desses resultados, pôde-se observar que objetos como a lata de refrigerante, por exemplo, apresentaram muitas imagens falsas, visto que no banco existem outras latas de refrigerantes com forma e distribuição de cores semelhantes. Nota-se que em determinadas situações, esta característica do programa pode ser até mesmo desejada. O método de reconhecimento tratado não emprega técnicas para tratar resultados falsos na ocasião do *matching*.

5.2 Avaliação de desempenho

A segunda série de experimentos realizados considerou o desempenho da aplicação em relação ao tempo de processamento. Foram realizados experimentos em uma arquitetura SMP, explorando unicamente o uso de *threads*, e em uma arquitetura distribuída.

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 analisar o comportamento da execução sob diferentes granulosidades, foram selecionados fragmentos de três tamanhos, pequeno (13: x 135), médio (272 x 272) e grande (540 x 540).

Para auxiliar na análise dos resultados, a Tabela 2 apresenta o tempo de busca com um algoritmo seqüencial desses fragmentos no banco de imagens proposto. Salienta-se que os tempos do algoritmo ponto a ponto foram estimados, visto que seu processamento é bastante demorado. Para tal estimativa utilizou-se, em cada fragmento, o tempo obtido na busca em uma imagem de dimensões 574x572, a qual reflete a média de dimensões das imagens do banco, multiplicado pelo número de imagens do banco (505).

Nessa tabela, pode-se observar que o número de operações realizadas é determinante no tempo de execução:

Tabela 2. Tempos de execução seqüencial

Fragmento	Ponto a Ponto	Histogramas
Pequeno	267.418 s	3.621 s
Médio	567.970 s	2.811 s
Grande	18.146 s	219 s

- O tempo de execução na comparação de histogramas tende a diminuir com o aumento do tamanho do fragmento. Na implementação desse algoritmo, dados calculados num movimento da cabeça de leitura são aproveitados pelo *matching* seguinte (reaproveitamento dos cálculos substituindo-se apenas as bordas). Portanto, quanto maior o fragmento, mais dados podem ser aproveitados em um menor número de deslocamentos.
- Fragmentos médios em relação ao tamanho das imagens tendem a aumentar o tempo de execução da comparação ponto a ponto. Neste caso, verifica-se que, muito embora o número de deslocamentos da cabeça de leitura diminua em relação a fragmentos pequenos, o número de operações de comparação aumenta. O pior caso é quando estes dois fatores encontram-se em uma situação intermediária, onde tal que o número de operações seja elevado.
- Por fim, o fragmento classificado como grande possui praticamente o mesmo tamanho da imagem, o que reduz o número de deslocamentos da cabeça de leitura. Também verifica-se que muitas imagens não são tratadas por apresentarem um tamanho inferior ao fragmento. Estes dois fatores implicam que os tempos de busca de fragmentos grandes sejam menores (conforme Tabela 2).

No geral, os resultados apresentados na Tabela 2 permitem concluir que a busca de fragmentos em um grande banco de imagens envolve grande custo computacional. A seguir, esses resultados vão ser comparados com outros gerados por execuções em uma arquitetura SMP e em um agregado de computadores. Finalizando a seção, é conduzida uma análise desses resultados.

5.2.1 Execução em um SMP

Na realização dos testes apresentados na Tabela 3, foram utilizadas 50 cabeças de leitura (*threads*), sendo que cada cabeça percorria 10 linhas consecutivas de qualquer imagem disponível. O experimento rodou numa máquina biprocessada, a fim de que o uso de *threads* fosse melhor aproveitado, gerando um paralelismo real para a aplicação.

Outro dado obtido de resultados intermediários permitiu verificar que o tempo de processamento aumenta de forma linear, para ambos algoritmos de busca. Foram realizadas medidas de desempenho para consultas ao banco de dados utilizando subconjuntos do total das imagens disponíveis. O aumento do número de imagens utilizadas neste subconjunto foi diretamente refletido no tempo total de processamento. Este fato nos permitiu realizar a extrapolação apresentada na Tabela 2, para os fragmentos pequeno e médio na comparação ponto a ponto.

Tabela 3. Tempos de execução em uma máquina SMP

Fragmento	P a P	Histograma
Pequeno	140.854 s	2.010 s
Médio	288.132 s	1.620 s
Grande	9.374 s	169 s

5.2.2 Execução em um agregado

A Tabela 4 documenta os tempos de execução em um agregado na busca de cada fragmento com ambos algoritmos de *matching*. Para cada nó do agregado foi criado um *pool* de trabalho constituído de 50 cabeças de leitura, sendo que cada uma era responsável pelo cálculo de 10 linhas de uma imagem qualquer.

Tabela 4. Tempos de execução em um agregado de computadores

Fragmento	P a P	Histograma
Pequeno	51.885 s	607 s
Médio	107.949 s	479 s
Grande	2.340 s	90 s

O banco de imagens, caracterizado por conter imagens de tamanho variável, foi de fundamental importância na avaliação desses resultados, uma vez que permitiu analisar o comportamento do algoritmo de balanceamento de carga. Na execução desses experimentos, constatou-se um bom comportamento do algoritmo de balanceamento, o qual, entre outros fatores importantes, não permitiu que nodos ficassem ociosos se houvesse carga de trabalho sobrando em outros nodos.

5.2.3 Análise dos resultados

A fim de avaliar o desempenho geral da aplicação, foi realizada uma comparação das execuções da busca dos três fragmentos na máquina SMP (Tabela 3) e no agregado (Tabela 4) com a busca dos mesmos em uma máquina monoprocessada (Tabela 2).

A Tabela 5 resume essas comparações, mostrando os ganhos obtidos em ambas execuções concorrentes. Os dados apresentados referem-se ao *speed-up* obtido, ou seja, da relação entre os tempos de execução seqüencial e paralelo ($sp = T_{seq}/T_{par}$). O tempo seqüencial utilizado refere-se ao tempo de busca em todo o banco de imagens, embora os resultados são igualmente válidos se fosse considerado o tempo médio de busca por imagem na execução paralela e o tempo seqüencial de busca em uma imagem (exceto no caso do fragmento grande).

Tabela 5. Ganhos obtidos

	SMP		Agregado	
Fragmento	P a P	Histograma	P a P	Histograma
Pequeno	1,90	1,80	5,15	5,97
Médio	1,97	1,74	5,26	5,87
Grande	1,94	1,30	7,75	2,43

O ganho obtido com a implementação concorrente é bem visualizado na Tabela 5. Nota-se que, com exceção do fragmento grande, os ganhos foram bastante próximos nos dois algoritmos, principalmente executando num agregado, 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, o descarte de imagens não permite avaliar o real ganho que seria obtido em tal situação. Ainda deve ser considerado que o agregado é composto de 4 nodos bi-processados, onde 4 processadores possuem 1 GHz e os 4 restantes 600 MHz. Esse fato impediu que os ganhos com a execução em um agregado fossem mais próximos a 8.

6 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. Os resultados de desempenho obtidos são encorajadores, motivando a continuidade dos trabalhos na linha de reconhecimento de imagens.

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. As ferramentas utilizadas para implementação, *threads* POSIX e MPI, são as atualmente utilizadas na implementação deste ambiente.

No entanto, não existe a preocupação de recuperar partes do código desenvolvidas especificamente para a aplicação descrita para o corpo do ambiente. Pelo contrário, o código em desenvolvimento para Anahy deve possuir características mais genéricas em relação ao construído para uma aplicação específica. Esta diferença permitirá a realização de análises da relação custo/benefício do uso de um ambiente de programação para a construção de uma aplicação concorrente.

Anahy está sendo desenvolvido [6, 13] 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. Neste contexto, a implementação realizada permitiu validar o modelo de descrição de concorrência de uma aplicação e de execução de tarefas que estão sendo propostos e implementados em Anahy. Também foi possível analisar o comportamento do *pool* de execução de tarefas e o conjunto do ambiente, o qual permite que a descrição da concorrência de uma aplicação seja feita de forma independente da arquitetura disponível para execução do programa.

A atual versão de Anahy está disponibilizada em um protótipo funcional sob arquiteturas do tipo SMP, empregando a técnica de *pool* de execução. Os trabalhos visando uma arquitetura com memória distribuída estão em seu estado inicial.

Em um outro sentido, a realização da implementação apontou aspectos que deverão ser considerados quando da introdução de extensões ao atual modelo de Anahy. Entre estes aspectos, a necessidade de algoritmos que explorem informações sobre o consumo de memória por parte das tarefas e a introdução de novas construções de descrição de concorrência, permitindo, por exemplo, que um grupo de tarefas produzam resultados acumulados sobre um mesmo dado, de forma semelhante a uma escrita concorrente.

Referências

[1] S. Antani, R. Kasturi, e J. Ramesh. A survey on the use of pattern recognition methods for abstraction, indexing and retrieval of images and video. *Pattern Recognition*, 35:945–965, 2002.

[2] Gerson Geraldo H. Cavalheiro. Introdução à programação paralela e distribuída. In Tiarajú A. Diverio e Philippe Navaux, editores, *1 Escola Regional de Alto Desempenho*, Gramado, Janeiro 2001.

[3] C. Cheung, L. Po, e K. Wong. Web-based beowulf-class parallel computing on image database indexing and retrieval system. In *International Symposium on Intelligent Multimedia, Video and Speech Processing*, Maio 2001.

[4] M. Das, E. Riseman, e B. Draper. Focus: Searching for multi-colored objects in a diverse image database.

In *IEEE Conf. on Comp. Vis. and Pattern Recognition*, Porto Rico, 1997.

[5] J. Fan, S. Yiu, e L. Po. Image retrieval using parallel algorithm on the beowulf class supercomputer. In *International Workshop on Multimedia Data Storage, Retrieval, Integration and Applications*, pages 188–191, Janeiro 2000.

[6] Alex Sandro Garzão, Lucas Correia Villa Real, e Gerson G. H. Cavalheiro. Ferramentas para desenvolvimento de um ambiente de programação sobre agregados. In *Anais do Workshop em Software Livre*, Porto Alegre, Brasil, Maio 2001.

[7] Jonas Gomes e Luiz Velho. *Computação Gráfica: Imagem*. IMPA/SBM, Rio de Janeiro, Brasil, 1994.

[8] Ramesh Jain, Rangachar Kasturi, e Brian G. Schunck. *Machine Vision*. McGraw-Hill International Editions, Singapore, 1995.

[9] Eduardo Moschetta, Fernando S. Osório, e Gerson G. H. Cavalheiro. Reconhecedor de imagens concorrente. *Scientia*, 13(2). Unisinos. No prelo.

[10] Eduardo Moschetta, Fernando S. Osório, e Gerson G. H. Cavalheiro. Reconhecedor de imagens usando técnicas de alto desempenho. In Tiarajú A. Diverio e Gerson G. H. Cavalheiro, editores, *2 Escola Regional de Alto Desempenho*, São Leopoldo, Janeiro 2002.

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

[12] A. Pentland, R. W. Picard, e S. Sclaroff. Photobook: Content-based manipulation of images databases. In *SPIE Storage and Retrieval Image and Video Database II*, number 2185, Fevereiro 1994.

[13] Lucas Correia Villa Real, Evandro Clivatti Dall'Agnol, e Gerson G. H. Cavalheiro. Construção de um ambiente de programação para o processamento de alto desempenho. In Tiarajú A. Diverio e Gerson G. H. Cavalheiro, editores, *2 Escola Regional de Alto Desempenho*, São Leopoldo, Janeiro 2002.

[14] Y. Rui, T. S. Huang, e S. Chang. Image retrieval: Past, present and future. In *International Symposium on Multimedia Information Processing*, Dezembro 1997.

[15] John R. Smith e Shih-Fu Chang. Tr: Automated image retrieval using color and texture. Technical Report 414-95-20, Columbia University, Julho 1995.

[16] INRIA-IMEDIA Project Research Team. Ikona. http://www.rocq.inria.fr/imedia/ikona/index.html (visitado em 15 de maio de 2002).