Sistemas distribuídos aplicados à compressão e recuperação de imagens

Fabio Jorge Assad Gostaldon[1*] e Ailton Akira Shinoda[2]

*[1]Departamento de Engenharia de Petróleo, Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, Rua Mendeleiev, s/n, 13083-970, Campinas, São Paulo, Brasil. [2]Departamento de Engenharia Elétrica, Universidade Estadual Paulista, Ilha Solteira, São Paulo, Brasil. *Autor para correspondência. E-mail: fabioj.assad@gmail.com*

RESUMO. Processamento digital de imagens é uma área que demanda grande capacidade de processamento. Sob este fato, torna-se interessante a implementação de softwares que estejam baseados na distribuição do processamento em vários nós ou máquinas da rede de computadores. Especificamente neste trabalho são abordados algoritmos distribuídos de compressão e expansão de imagens que utilizam a imagem transformada discreta de cosseno. Os resultados mostram que a economia de tempo conseguida com os algoritmos paralelos, em relação aos seus equivalentes seqüenciais, é função da resolução da imagem e da complexidade dos cálculos envolvidos, ou seja, maior a eficiência quanto maior o tempo de processamento em relação ao tempo de comunicação entre os nós.

Palavras-chave: MPI, processamento digital de imagem, sistemas distribuídos.

ABSTRACT. Distributed systems applied to image compression and recovery. Digital image processing is a field that demands great processing capacity. As such, it becomes relevant to implement software that is based on the distribution of the processing into several nodes divided by computers belonging to the same network. Specifically discussed in this work are distributed algorithms of compression and expansion of images using the discrete cosine transform. The results show that the savings in processing time obtained due to the parallel algorithms, in comparison to its sequential equivalents, is a function that depends on the resolution of the image and the complexity of the involved calculation; that is, efficiency is greater the longer the processing period is in terms of the time involved for the communication between the network points.

Key words: MPI, digital image processing, distributed systems.

Introdução

O processamento de imagens é uma área interdisciplinar, em que diversos temas científicos são abordados, entre eles pode-se citar a compressão de imagens, a análise em multirresolução e em multifreqüência, a análise estatística, a codificação e a transmissão de imagens, entre outros. O termo imagem estava, inicialmente, associado ao domínio da luz visível, porém atualmente é muito freqüente falar de imagens quando uma grande quantidade de dados está representada sob a forma bidimensional (por exemplo: as imagens acústicas, sísmicas, de satélites, infravermelhas, magnéticas etc).

A primeira fase do processamento de imagens é a aquisição, armazenamento e recuperação dos dados, neste meio está incluída a compressão.

Existem três níveis de abstração para especificar uma imagem no computador: imagem contínua, discretizada, e codificada. Cada uma dessas especificações pode ocupar mais ou menos espaço para ser armazenado no computador. A compressão de imagens trata exatamente disso, dada uma imagem em uma determinada especificação as técnicas de compressão obtêm outra especificação que ocupe menor espaço de armazenamento.

A compressão de imagens é possível, pois em geral as imagens têm alta coerência o que resulta em muitas redundâncias na informação quando codificada. Por exemplo, um ponto, que represente a menor unidade da imagem, provavelmente, terá a mesma cor que seus vizinhos.

Gomes e Velho (1994) classificam as técnicas de compressão em três tipos diferentes: transformação do modelo, por discretização e por codificação.

Neste trabalho, foram utilizadas em conjunção as técnicas de transformação do modelo e por discretização. A técnica de compressão por transformação do modelo consiste em analisar a imagem por meio de transformadas, no caso deste trabalho, foi utilizada a transformada discreta de cosseno, de forma a obter um modelo mais conveniente para sua especificação e utilizar esse

modelo para codificar a imagem. O método de discretização utiliza um modelo funcional da imagem; neste caso o modelo é gerado pela transformada discreta de cosseno, e procura fazer a compressão reduzindo informações contidas nela. A forma mais simples de quantização por discretização da imagem é com a utilização de um método uniforme (Figura 3) ou adaptativo. Se a imagem possui somente informações de alta freqüência, utiliza-se esse método para fazer uma quantização com um número reduzido de bits.

Transformada discreta

Transformadas bidimensionais são utilizadas na grande maioria das aplicações em processamento de imagens. Uma transformada bidimensional é aquela que pode ser revertida, ou seja, possui inversa. Especificamente em compressão de imagens, para o caso abordado neste trabalho, quando a transformada é aplicada, apenas alguns coeficientes resultantes da mesma são aproveitados e alguns reescritos com menos bits, isso possibilita o armazenamento em menor espaço ou transmissão em menor tempo.

Quando necessário, é possível recuperar essa imagem compactada e aplicar a transformada inversa sobre os coeficientes armazenados, entretanto a imagem original é recuperada com degradação.

A mais comum das transformadas bidimensionais de compressão de imagens é a transformada discreta de cosseno (*Discret Transform Cossine* – DTC), que é base para os formatos de compressão mais populares como o JPEG e MPEG.

De maneira geral, os dados amostrados da imagem são submetidos à transformada discreta de cosseno e alguns coeficientes resultantes são escolhidos e de maneira estratégica (quantização por discretização) cada um é reescrito com uma quantidade de bits que proporciona a efetiva compressão dos dados.

A transformada discreta de cosseno para uma imagem NxN é definida como segue (Oppenheim e Schafer, 1998):

$$F[u,v] = \frac{1}{N^2} \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} f[m,n] X_{m,n}(u,v) \qquad (1)$$

em que:

u,v = posições discretas (0, 1, 2, ..., N-1);

f[m,n] = NxN *pixels* da imagem (0, 1, 2, ..., N-1);

$X_{m,n}(u,v) = \cos\left[\frac{(2m+1)u\pi}{2N}\right] \cos\left[\frac{(2n+1)v\pi}{2N}\right]$;

F[u,v] = resultado da DTC.

A DTC inversa é definida como:

$$\hat{f}[m,n] = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} c[u]c[v] X_{m,n}(u,v) \qquad (2)$$

em que:

m,n = índices resultantes dos *pixels* da imagem;

$\hat{f}[m,n]$ = resultado da DTC inversa.

Sistemas multiprocessados

Um meio de aumentar a velocidade dos computadores é a utilização de sistemas multiprocessados que são máquinas constituídas de muitas CPUs cada uma independente da outra, mas quando trabalhando coletivamente, possuem poder computacional muito grande

Resumidamente, classificam-se os sistemas multiprocessados de três maneiras: multiprocessadores de memória compartilhada (supercomputadores) que se comunicam por meio dela; multicomputadores que são constituídos por pares CPU-memória e não compartilham um espaço de memória (*Clusters*), comunicam-se por meio de troca de massagens em uma rede dedicada de alta performance, baixa latência e banda passante alta; e os sistemas distribuídos constituídos de várias estações de trabalho completa com CPU, memória e disco rígido local, monitor, teclado e mouse, interconectadas por uma rede comercial-padrão e comunicam-se por meio de troca de mensagens..

Um sistema distribuído utilizado neste trabalho, que segundo a definição de Tanenbaum e Steen (2001), é uma "coleção de computadores independentes que se apresenta ao usuário como um sistema único e consistente"; outra definição, de Coulouris *et al.* (2005), diz: "coleção de computadores autônomos interligados através de uma rede de computadores e equipados com software que permita o compartilhamento dos recursos do sistema: hardware, software e dados".

Neste tipo de sistema, o acoplamento é fraco, e isto pode ser encarado como vantagem ou desvantagem. Vantagem porque este tipo de sistema pode ser utilizado para muitos tipos de aplicação, enquanto tem-se como desvantagem dificuldade na programação para esse tipo de plataforma. O custo desse tipo de sistema também é reduzido, pois pode ser utilizada a rede de computadores já disponível nas dependências e, com o avanço exponencial da Internet, os computadores ligados a ela também podem integrar um sistema distribuído.

Além da rede de comunicação, é necessária uma camada de software que possa gerenciar o uso

paralelo das estações de trabalho. Para tanto, existem bibliotecas especializadas para tratamento da comunicação entre processos e a sincronização de processos concorrentes. Um dos sistemas de troca de mensagens mais utilizados na atualidade é o MPI (*Message Passing Interface*).

Message Passing Interface (MPI)

MPI é uma biblioteca de troca de mensagem desenvolvida para ser padrão de comunicação entre processos em ambientes de memória não-compartilhada (Groop e Huss-Lederman, 1998). O MPI define um conjunto de rotinas para facilitar a comunicação (troca de dados e sincronização) entre processos, ela é portável para qualquer arquitetura, tem aproximadamente 125 funções para programação e ferramentas para análise de performance. A biblioteca MPI possui rotinas para programas em linguagem C, C++, Fortran 77/90. Os programas são compilados e ligados à biblioteca MPI.

No padrão MPI, uma aplicação é constituída por um ou mais processos que se comunicam, acionando-se funções para o envio e recebimento de mensagens entre os processos. Inicialmente, na maioria das implementações, um conjunto fixo de processos é criado. Porém, esses processos podem executar diferentes programas. Por isso, o padrão MPI é algumas vezes referido como MPMD (*Multiple Program Multiple Data*). Elementos importantes, em implementações paralelas são a comunicação de dados entre processos paralelos e o balanceamento da carga. Os processos podem usar mecanismos de comunicação ponto a ponto (operações para enviar mensagens de um determinado processo a outro). Um grupo de processos pode invocar operações coletivas (*collective*) de comunicação para executar operações globais. O MPI é capaz de suportar comunicação assíncrona e programação modular, por meio de mecanismos de comunicadores (*communicator*) que permitem ao usuário MPI definir módulos que encapsulem estruturas de comunicação interna.

Existem muitas implementações do MPI, algumas delas com direito de propriedade, mas existem outras de uso livre com o mesmo padrão de funcionamento e uso. Neste trabalho, foi utilizada a implementação LAM (*Local Area Multcompulter*) que é um ambiente de programação desenvolvido para criação de sistemas em uma rede local heterogênea de computadores. Utilizada com um cluster ou uma rede de estações de trabalho, estes podem ser operado como se fosse um único computador paralelo.

Material e métodos

Este trabalho foi realizado nas dependências do LLPP (Laboratório Linux de Processamento Paralelo), localizado no Departamento de Engenharia Elétrica da Unesp/Ilha Solteira. O laboratório possui quatro computadores AMD 2.6 Ghz, 512 Mb de RAM interligados com *switch* da 3com a 100 Mbits s^{-1}. No caso do LLPP, o *login* do usuário é autenticado em um servidor NIS e o seu diretório *home* é exportado via NFS. Assim, em qualquer estação da rede, o usuário pode *logar* utilizando seu próprio *login* e senha, e terá acesso aos seus dados.

Antes da implementação dos algoritmos, fez-se necessária a definição da estrutura de arquivos onde são armazenadas as imagens. Os algoritmos, base deste trabalho, são definidos em Embree e Danieli (1998) assim como a estrutura de arquivos que é chamada de *DSPfile* e tem a sua forma ilustrada na Figura 1.

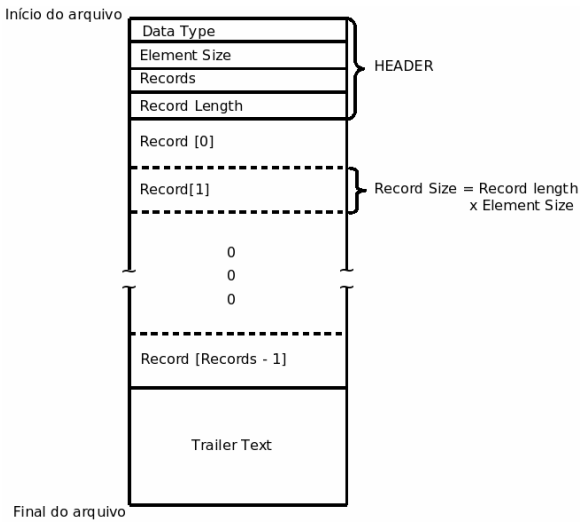

Figura 1. Estrutura de um arquivo de dados do tipo DSP.

A imagem digital monocromática é representada por uma matriz, em que o valor de cada uma de suas posições define a intensidade luminosa e o nível de cinza de cada *pixel*. Com isso, para armazenar essa imagem em um *DSPfile* cada linha da matriz é armazenada em um *Record*.

O algoritmo paralelo de compressão, implementado em linguagem C, faz a leitura da imagem em um *DSPfile* e então opera uma técnica conhecida como código de bloco, que é usado para reduzir o tempo de transformação de uma imagem. Essa técnica divide a imagem em várias subimagens menores de tamanho 8x8. É exatamente neste ponto que o paralelismo é explorado, o número de subimagens resultantes é dividido adequadamente em conjuntos de forma que, se

novamente agrupadas, formam oito linhas da imagem original, e assim são enviadas, para os nós, quais conjuntos eles serão responsáveis.

A distribuição dos blocos entre os nós é feita seguindo uma divisão chamada de blocos linha. A Figura 2 ilustra melhor a distribuição, e toma como exemplo uma situação em que se tem uma imagem que pode ser dividida em Z blocos na direção horizontal e M blocos na direção vertical, cada bloco tem dimensão 8x8 e estão disponíveis M/3 nós do sistema distribuído.

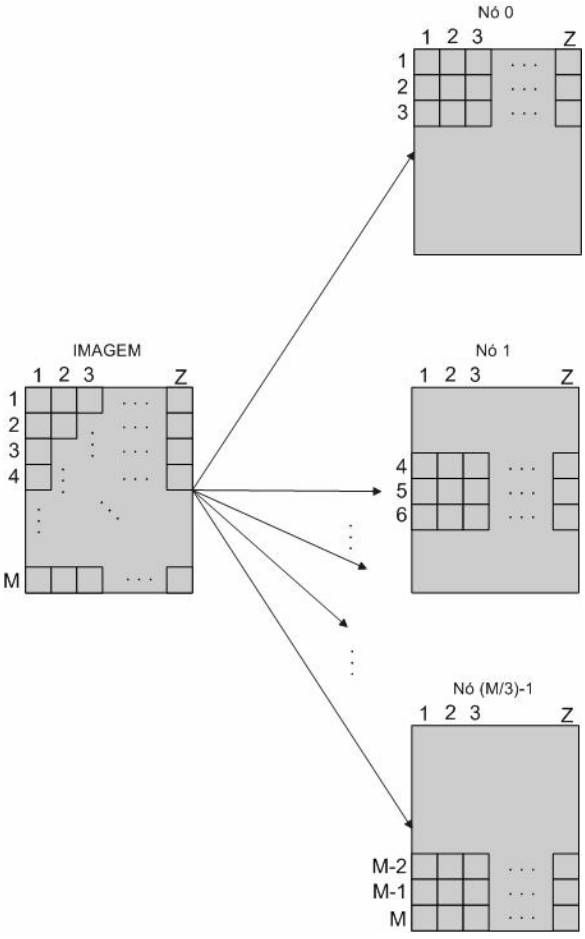

Figura 2. Exemplificação da distribuição dos blocos entre os nós.

Em cada nó é aplicada a DTC nas subimagens definida pela Equação (3).

$$DTC\{\mathbf{A}\} = \frac{\mathbf{CAC}^T}{N^2} \quad (3)$$

em que a matriz **A** é a subimagem, N é a ordem da matriz **A** e **C** a matriz de cossenos constantes onde cada elemento é definido pela Equação (4).

$$c[u][n] = \cos\left[\frac{(2n+1)u\pi}{2N}\right] \quad (4)$$

Os coeficientes resultantes da aplicação da DTC são, então, quantizados, isto é, reescritos com menos bits, e com isso se perde alguma informação, causando degradação na qualidade da imagem recuperada. Os coeficientes resultantes da DTC também formam uma matriz 8x8 e é por meio da posição do coeficiente na matriz, como é apresentado na Figura 3, que se determina com quantos bits ele será reescrito. O que promove um fator de compressão de 4:1.

8	8	4	4	4	4	4	4
8	8	4	4	4	4	0	0
4	4	4	4	0	0	0	0
4	4	4	4	0	0	0	0
4	4	0	0	0	0	0	0
4	4	0	0	0	0	0	0
4	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0

Figura 3. Matriz com a quantidade de bits que cada coeficiente será reescrito.

A utilização do código de bloc o acentua um pouco a degradação da qualidade da imagem. Quando o tamanho do bloco é grande, a degradação ligada à eliminação de coeficientes distribui-se sobre um grande espaço na imagem. Conforme o bloco é reduzido, a degradação se torna mais intensa entre os blocos, e as suas fronteiras se destacam na imagem recuperada (Pratt, 1978). O sistema da visão humana é mais sensível a padrões regulares de interferência, como nas fronteiras dos blocos, do que aquelas randômicas e dispersas. Como resultado, é percebida alguma perda de qualidade na imagem.

Para determinar o melhor tamanho do bloco, primeiramente é preciso conhecer a relação entre os blocos e *pixels* adjacentes. Na maioria das imagens, a correlação significante existe apenas para 20 *pixels* adjacentes. Portanto, para blocos maiores que 16x16, o ganho de fidelidade é menor e menos significativo. Por isso, blocos maiores que 16x16 não se justificam (Pratt, 1978). Além disso, o tamanho de 8x8 escolhido para cada bloco é estratégico e visa à diminuição do tempo de execução do algoritmo, já que este é proporcional a N^2, em que N é a ordem do bloco como definido na Equação (3).

O fluxograma da Figura 4 exemplifica melhor a dinâmica empregada no algoritmo de

compressão de imagens.

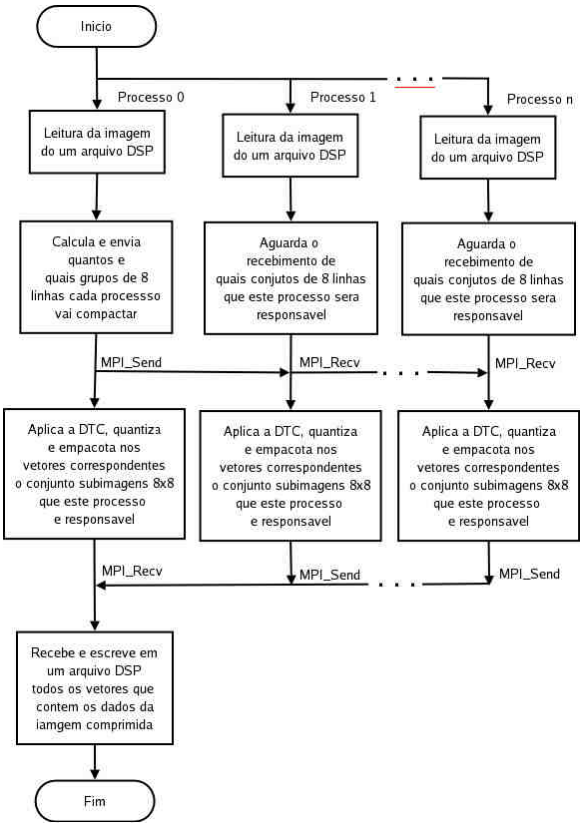

Figura 4. Fluxograma do algoritmo paralelo de compressão de imagens.

O algoritmo de expansão funciona de forma similar ao de compressão. Faz-se primeiro a leitura do *DSPfile* gerado pelo programa de compressão e, então, a quantidade de *Record* (Figura 1) do arquivo é repartido entre os nós que primeiramente descompacta os coeficientes quantizados copiando os mesmos para uma matriz 8x8. É aplicada a DTC inversa nessas matrizes, o que resulta nas subimagens que são enviadas ao nó principal onde são adequadamente agrupadas formando novamente a imagem com alguma degradação em relação à original.

Os testes iniciaram-se aplicando os algoritmos de compressão e expansão sobre uma imagem com resolução 256x256. Nesta etapa, os algoritmos são executados em quatro nós.

Para a melhor percepção da economia de tempo, no uso dos algoritmos paralelo, estes foram aplicados sobre uma mesma imagem de resolução 256x256 (1024 blocos), 640x480 (4800 blocos), 2048x1536 (49152 blocos), 2560x1920 (76800 blocos) e 2816x2112 (92928 blocos). Este procedimento foi repetido variando-se a quantidade de nós da máquina paralela.

Como em termos absolutos, os valores de tempo gasto com a compressão e expansão não são muito altos, criou-se um cenário composto por uma seqüência de até 50 imagens de resolução 2816x2112, ou seja, 92.928 blocos, aumentando o tempo gasto na execução dos algoritmos e assim é possível realizar melhor análise. Deve-se ressaltar que as imagens utilizadas eram idênticas.

Resultados e discussão

A Figura 5 mostra a imagem original de Lenna e a Figura 6 apresenta a imagem de Lenna depois de ser compactada e recuperada utilizando quatro nós. Nota-se que a imagem recuperada é muito similar à original.

Figura 5. Imagem (256 x 256) original de Lenna.

Figura 6. Imagem de Lenna após ser comprimida e expandida.

O tempo de execução de compressão da imagem aumenta linearmente com o número de blocos, ou

seja, quanto maior o número de *pixels* da imagem. Isto pode ser observado na Figura 7.

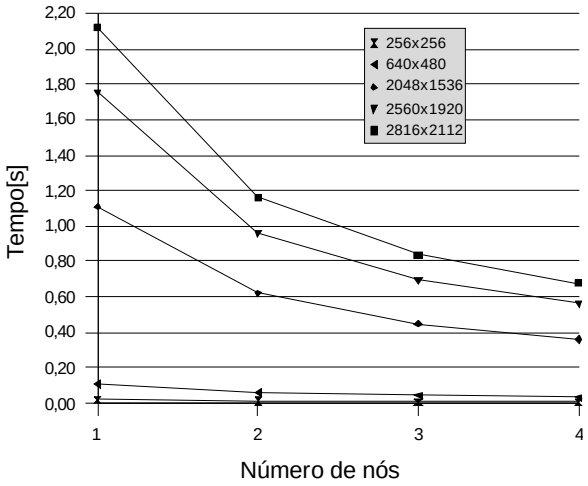

Figura 7. Tempo de compressão para imagens de diferentes resoluções.

Contudo, percebe-se que para apenas uma imagem a diminuição no tempo de execução é pouco sensível ao uso do algoritmo paralelo. Como esperado, o uso do algoritmo paralelo é mais eficiente em imagens com grande número de blocos, ou seja, com alta resolução. Utilizando esse fato e imaginado uma situação em que se deve comprimir uma seqüência de imagens de alta resolução, o tempo de execução para a compressão dessa seqüência aumenta com o aumento do número de imagens, essa relação é linear como se pode observar na Figura 8 e na Figura 9 onde é apresentado o *speedup* do programa.

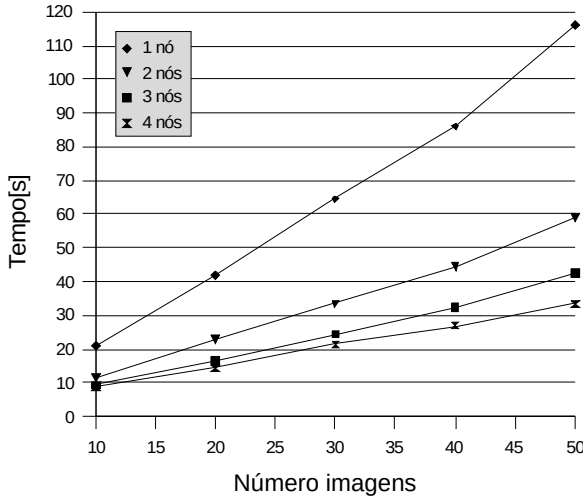

Figura 8. Tempos de execução para comprimir séries de imagens com 92.928 blocos.

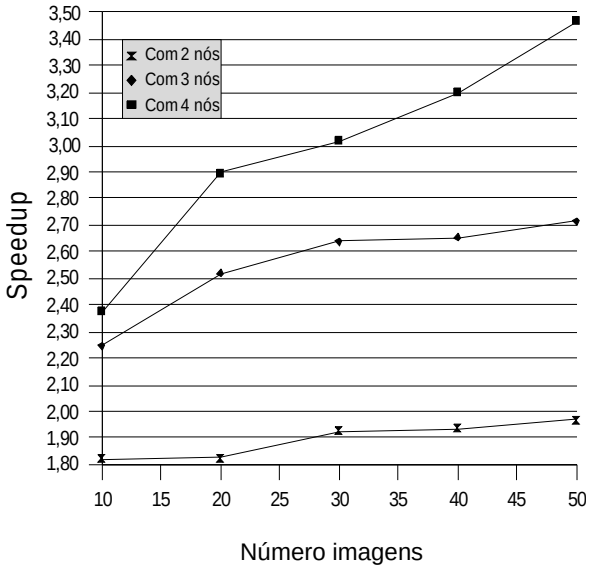

Figura 9. *Speedup* para comprimir as séries de imagens com 92.928 blocos.

Os resultados da utilização de técnicas de paralelização para o algoritmo de recuperação de imagens não se mostraram tão vantajosos quanto os de compressão. Isto ocorreu porque, após a compressão, a operação inversa para recuperar a imagem, mais especificamente a inversa da transformada discreta do cosseno, é mais simples, ou seja, a quantidade de operações e complexidade dos cálculos se torna tão pequena que o tempo gasto com o processamento dos dados é relativamente menor em relação ao tempo gasto com a comunicação entre os processos. Estes fatos são apresentados nos gráficos das Figuras 10 e 11.

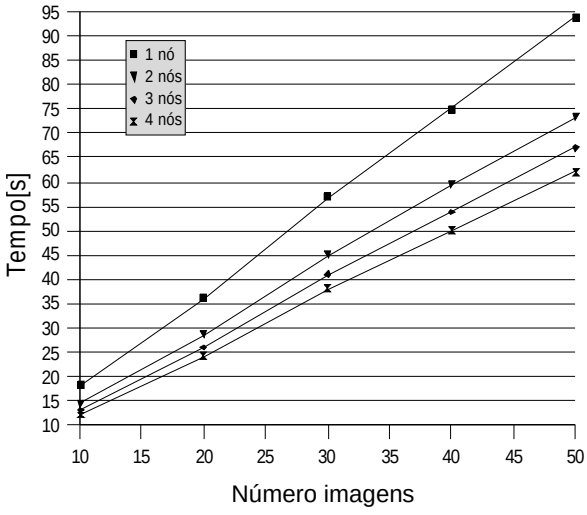

Figura 10. Tempo de execução para expandir a série de imagens comprimidas anteriormente.

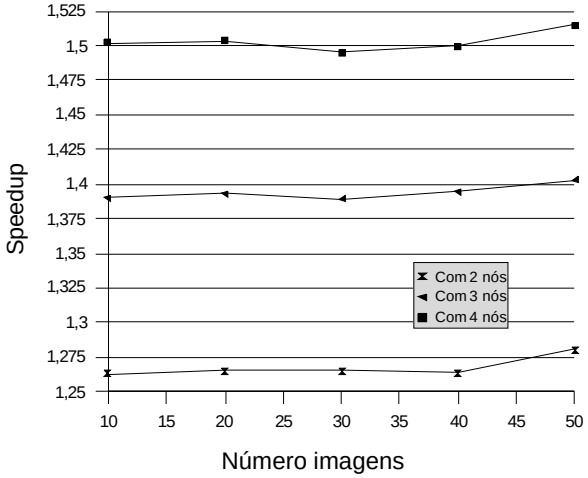

Figura 11. *Speedup* para expandir as séries de fotos de 92.928 blocos.

Pode-se observar que apesar do tempo de execução aumentar com o aumento do número de fotos da série, a economia de tempo que utiliza mais nós é pequena e se mantém praticamente constante um em relação ao outro, e por isso o *speedup* se mantém constante não importando o número de fotos da série. Também o valor do *speedup* não é muito satisfatório atingindo apenas o valor de 1,5 quando utilizados quatro computadores.

Conclusão

A utilização das técnicas de paralelização nos algoritmos de compressão e expansão não diminuiu sua eficiência com relação à qualidade da imagem resultante, isso devido à paralelização estar baseada no código de bloco já bem fundamentado teoricamente.

Na compressão de imagens, a utilização de algoritmos distribuídos foi altamente vantajosa. A economia de tempo foi significativa, chegando até a 3,5 vezes com quatro computadores para se comprimir 50 imagens de 92.928 blocos. Isso é resultado da alta quantidade de cálculos complexos a serem realizados em cada parcela da imagem que os processos efetuam.

Como mencionado, anteriormente, o mesmo não ocorreu para a expansão das imagens. A complexidade dos cálculos não se mostrou tão alta e, por isso, o ganho de tempo com o processamento distribuído aplicado na expansão não foi muito significativo. Com isso, conclui-se que o processamento paralelo é vantajoso em situações de processamento intenso, como já era esperado.

De maneira geral, os resultados se mostraram favoráveis ao uso do processamento distribuído em processamento de imagens.

Agradecimentos

Este trabalho foi financiado pela Fundação de Amparo à Pesquisa do Estado de São Paulo (Fapesp), por intermédio de bolsa de Iniciação Científica, processo 2005/04786-2.

Referências

COULOURIS, G. *et al. Distributed systems*: *concepts and design.* 4th ed. New Jersey: Prentice Hall, 2005.

EMBREE, P.M.; DANIELI, D. *Algorithms for digital signal processing.* 2nd ed. New Jersey: Hall PTR, 1998.

GOMES, J.; VELHO, L. *Computação gráfica: imagem.* Rio de Janeiro: IMPA, 1994.

GROOP, W.; HUSS-LEDERMAN, S. *MPI: the complete reference.* Cambridge: MIT Press, 1998.

KARNIADAKIS, G.; KIRBY, R.M. *Parallel scientific computing in C++ and MPI.* Cambridge: Cambridge University Press, 2003.

OPPENHEIM, A.V.; SCHAFER, S.W. *Discrete time signal processing.* 2nd ed. New Jersey: Prentice Hall, 1998.

PRATT, W.K. *Digital image processing.* New York: Wiley-Interscience, 1978.

TENENBAUM, A.S. *Sistemas operacionais modernos.* Rio de Janeiro: Livros Técnicos e Científicos, 1999.

TENENBAUM, A.S.; STEEN, M.V. *Distributed systems: principles and paradigms.* New Jersey: Prentice Hall, 2001.

Received on July 06, 2007.
Accepted on August 28, 2007.