Anais XIII Simpósio Brasileiro de Sensoriamento Remoto, Florianópolis, Brasil, 21-26 abril 2007, INPE, p. 6257-6264.

Detecção de Paralelismo para Filtros Convolucionais

Bruno Lopes Vieira
Eliana Silva de Almeida
Alejandro César Frery

Universidade Federal de Alagoas – UFAL
BR-104 Norte km 97 – 57072-970 Maceió – AL, Brasil
blv@tci.ufal.br, {eliana.almeida,acfrery}@pesquisador.cnpq.br

Abstract. This paper proposes an algorithm to translate sequential convolutional filters programs into parallelized ones. This algorithm consists of mapping the R code of an image filter to a dependence graph (an intermediate language), where one can clearly observe the tasks that may be parallelized. From this graph, the algorithm generates the parallel code in the R programming language. As a case study, we used the Gaussian filter and obtained an considerable reduction in the numbers of operations, from 28 to 9 operations per pixel.

Palavras-chave: convolutional filters, parallelism deteccion, dependence graph, filtros convolucionais, detecção de paralelismo, filtros convolucionais.

1. Introdução

Filtros de imagens são algoritmos capazes de obter novas imagens a partir de transformações lógicas e/ou matemáticas dos dados de entrada. Podem se apresentar como um conjunto de instruções, tanto na linguagem de circuitos eletrônicos como em um programa de computador.

O uso de filtros é essencial na cadeia de processamento e análise de imagens, pois eles permitem combater elementos nocivos à compreensão das informações, como por exemplo ruídos e outros artefatos, e realçar aquelas características de interesse como, por exemplo, bordas. Além do resultado almejado, um dos pontos críticos na escolha de um filtro é o seu desempenho computacional. Para atender as constantes necessidades de melhoria de desempenho destas aplicações, uma alternativa é otimizar a performance dos algoritmos, tendo em vista que algumas aplicações requerem resposta quase que imediata. Uma outra abordagem, complementar à otimização de algoritmos, é o uso de plataformas paralelas ou distribuídas; os *clusters* de computadores de baixo custo são, em grande parte, responsáveis pela popularização desta abordagem antes limitada a computadores de grande porte dedicados.

Este trabalho propõe uma ferramenta destinada a detectar paralelismo em filtros de imagens convolucionais. Mostra-se que, dado o código de um filtro de imagem como entrada, este código é mapeado em uma linguagem intermediária denominada “grafos de dependências”. A partir desta representação intermediária, é possível produzir o correspondente paralelo do filtro original. Esta proposta tem o intuito de facilitar o processo de tradução de código seqüencial para a programação paralela, sem exigir conhecimentos deste paradigma. Detalha-se a implementação desta técnica para filtros especificados em código R.

O uso de grafos de dependências como linguagem intermediária para a otimização de programas foi proposto em Ferrante e Ottenstein (1987). Algumas das principais estruturas algorítmicas passíveis de paralelização estão descritas em Banerrjje et al. (1993). Para identificá-las, o presente trabalho propõem técnicas de análise de estruturas de repetição, possibilitando sua conversão em estruturas seqüenciais que podem ser representados por grafos orientados acíclicos. Convém notar que essa abordagem é uma metodologia aprovada pelo IEEE (2006).

2. Filtros de imagens convolucionais

Uma imagem monoespectral é uma função $f\colon S \to \mathbb{R}$, onde $S = \{0, \ldots, m-1\} \times \{0, \ldots, n-1\}$ é o suporte. O par $(s, f(s))$, com $s \in S$, chama-se pixel, e pode ser conveniente deixar em evidência as componentes da coordenada, isto é, usar (i, j) no lugar de $s \in S$.

Dada a matriz de convolução de lado ℓ ímpar

$$A = (a_{i,j})_{\frac{\ell-1}{2} \leq i,j \leq \frac{\ell-1}{2}},$$

com $a_{i,j} \in \mathbb{R}$, define-se a imagem $g\colon S \to \mathbb{R}$ resultante de filtrar a imagem f por convolução com a máscara A como sendo

$$g(k,\ell) = \begin{cases} \displaystyle\sum_{-\frac{\ell-1}{2} \leq i,j \leq \frac{\ell+1}{2}} a_{i,j}\, f(k-i, \ell-j) & \text{se } (k,\ell) \in S', \\ f(k,\ell) & \text{caso contrário,} \end{cases} \tag{1}$$

onde $S' = \{\frac{\ell-1}{2}, \ldots, m - \frac{\ell+1}{2}\} \times \{\frac{\ell-1}{2}, \ldots, n - \frac{\ell+1}{2}\}$. Freqüentemente tem-se que $\sum_{i,j} a_{i,j} = 1$.

Não pertencem à classe dos filtros convolucionais os filtros não lineares (como, por exemplo, o da mediana) nem os adaptativos (como, por exemplo, o de Lee). O uso da classe de filtros convolucionais permite obter resultados muito importantes.

Tal como visto na equação (1), os filtros convolucionais são baseados em operações matriciais como multiplicação e soma. As operações matriciais, além de muitas outras, são classicamente paralelizáveis (ver Quinn, 2003) e, com elas, os filtros convolucionais.

3. Grafos de dependências

Entende-se por um grafo de dependências um grafo acíclico não-orientado, cujo uso representa unidades atômicas de execução de um programa, unidas por dependências de fluxo de dados e/ou controle. Dessa forma, através dessa categoria de grafos pode-se representar todas as dependências dentre as estruturas de processamento de um algoritmo. Nele, temos uma linguagem intermediária, ou seja, uma linguagem não implementável, porém, capaz de representar qualquer estrutura algorítmica computável, independente da linguagem em que foi escrita, através de um mapeamento, como pode ser visto em Ferrante e Ottenstein (1987).

Exemplificando, observe-se o código 1 mapeado num grafo de dependências na figura 1.

Código 1 Pseudocódigo para aplicação de dois filtros de imagens

```
1: for ( x in 1 : 2 ) do
2:    for ( y in 1 : 2 ) do
3:       aplicaFiltroNPixelAPixel ( x, y )
4:       aplicaFiltroMPixelAPixel ( x, y )
5:    end for
6: end for
```

Observe-se na figura 1 que cada laço da repetição está explicitado. Considerando que não importa a ordem de execução de cada filtro (o resultado ao se aplicar o filtro M e o N em seguida é o mesmo de N e em seguida M), ou seja, não há dependência de dados. Pode-se concluir que os filtros podem ser aplicados em paralelo. Da mesma forma, conclui-se que o conteúdo do primeiro laço de repetição pode ser executado em vias paralelas (o segundo laço deve ser repetido 2 vezes, ver código 1, as quais podem ser executadas ao mesmo tempo).

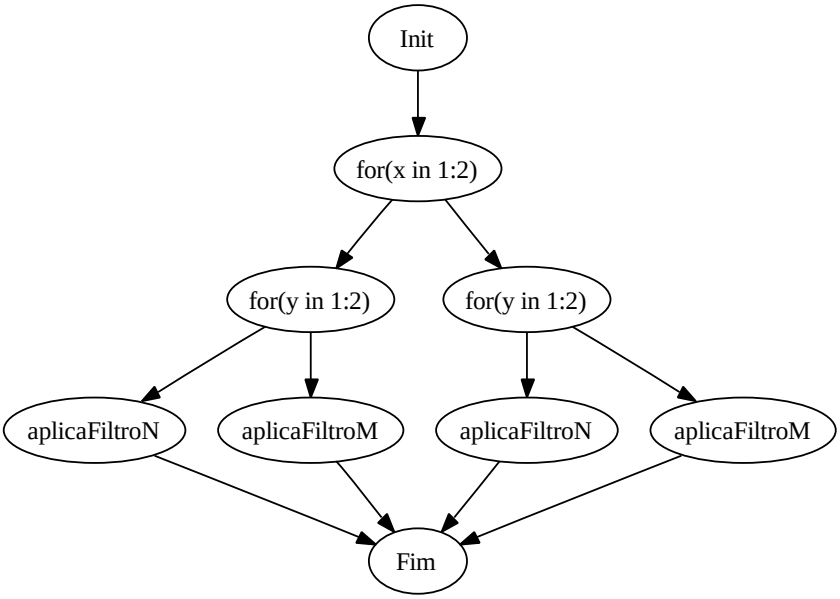

Figura 1. Grafo gerado a partir do código 1

4. Detecção de paralelismo

Para que se possa compreender os princípios adotados no trabalho, faz-se necessário o entendimento da linguagem adotada. Optou-se por utilizar a linguagem `R` por ela estar sendo fortemente utilizada nas aplicações de filtros de imagens, visto que apresenta diversas ferramentas para computação científica, além de ter seu uso gratuito e código fonte disponível (utiliza a licença GPL – ver `www.fsf.org`). A princípio, utiliza-se apenas a base estruturada da linguagem, tendo, portanto, como base léxica:

- funções
- procedimentos
- laços de repetição (`for`, `repeat`, `while`)
- estruturas condicionais (`if`)

Analisando-se os códigos de filtros de imagens disponíveis, observa-se com grande freqüência que ocorrem laços de repetição `for` encadeados com variávies de controle alteradas no código. Ou seja, observe-se o código 2.

Código 2 Pseudocódigo com `for` encadeado

```
1: Lx = imageWidth
2: Ly = imageHeight
3: for ( x in 1 : Lx ) do
4:     for ( y in 1 : Ly ) do
5:         aplicaFiltroNPixelAPixel ( x, y )
6:         aplicaFiltroMPixelAPixel ( x, y )
7:     end for
8: end for
```

Para mapear tal caso de forma a explicitar a possibilidade de paralelização, optou-se por criar a convenção da figura 2.

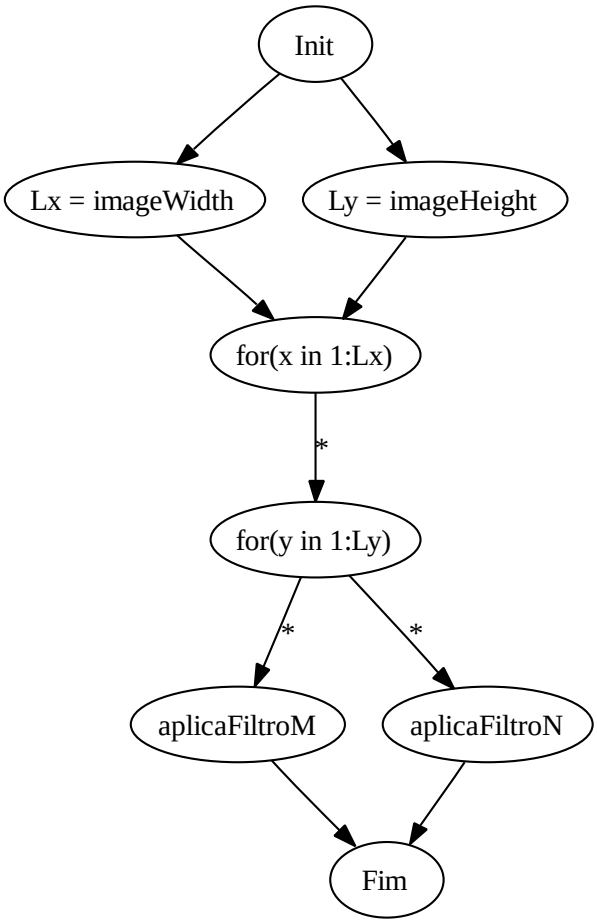

Figura 2. Grafo com laço `for` encadeado n vezes

Note-se a marca "$*$" no arco direcional dentre os laços `for` e os filtros. Elas indicam que as etapas podem ocorrer simultaneamente. Vale ressaltar que um laço deste tipo pode estar disfarçado em estruturas `repeat` ou `while` (vide código 3).

Um conceito importante a compreensão do trabalho é o de dependência de dados: um determinado processo ter de esperar pelo fim do processamento de algum outro para continuar, pois precisa de seu valor. Observe-se exemplificação no algoritmo 4, onde a linha 2 depende da linha 1 e a 3 da 2, ambos em relação a dados.

O algoritmo desenvolvido para mapear um filtro de imagens implementado em `R` em um grafo de dependências consiste nos seguintes passos:

1. Iniciar com o vértice `Init`;
2. interpretar cada comando como um vértice;
3. interpretar cada dependência de controle como uma seta contínua;
4. ao se deperar com um laço `repeat`, caso a condição de saída seja um contador, convertê-lo em um laço `for`;
5. ao se deparar com um laço de repetição `while`, caso a condição de saída seja um contador, convertê-lo em um laço `for`;
6. ao se deparar com um laço de repetição `for`:

Código 3 Laço `for` disfarçado em `while`

```
 1: Lx = imageWidth
 2: Ly = imageHeight
 3: x = 0
 4: repeat
 5:    x = x + 1
 6:    y = 0
 7:    repeat
 8:       y = y + 1
 9:       aplicaFiltroMPixelAPixel ( x, y )
10:       aplicaFiltroNPixelAPixel ( x, y )
11:    until y = Ly
12: until x = Lx
```

Código 4 Dependência de dados

```
1: X = 1
2: Y = X + 1
3: X = X + Y
```

- caso seja possível determinar o número de repetições, gerar uma seqüência paralela para cada repetição;
- caso não seja possível determinar o número de repetições, gerar apenas uma seqüência com a marca "$*$";
- caso os comandos internos em paralelo possuam dependências de dados, seguem em vértices seqüenciais;

7. chamadas a procedimentos seguem em ramos paralelos;
8. caso haja dependência de dados dentre os procedimentos, seguem em vértices seqüenciais;
9. demais casos seguir seqüencialmente.

5. Estudo de caso

Um filtro convolucional muito utilizado é o filtro gaussiano. Ele é definido por uma máscara quadrada cujos coeficientes são proporcionais a uma densidade gaussiana bivariada de média nula e matriz de covariância . Os parâmetros do filtro são o tamanho da máscara e o espalhamento dessa densidade; para filtros definidos sobre máscaras quadradas de lado K ímpar, quanto maior o espalhamento, medido por $\sigma > 0$, maior será o efeito de borramento na imagem filtrada.

Os coeficientes do filtro gaussiano são dados por

$$a_{i,j} = Z_{K,\sigma} \exp\Big\{-\frac{(i^2 + j^2)}{2\sigma^2}\Big\},$$

onde $-(K-1)/2 \leq i, j \leq (K-1)/2$ e $Z_{K,\sigma}$ é a constante de padronização que garante $\sum_{i,j} a_{i,j} = 1$.

Este filtro é apropriado para combater ruído aditivo, mas introduz um certo borramento na imagem de saída. Os coeficientes são todos positivos e simétricos em relação ao centro da máscara.

A seguir mostramos as máscaras para os filtros de lados $K = 3$ (primeira linha) e $K = 5$ (segunda linha) com $\sigma = 1$ e $\sigma = 10$:

$$\begin{pmatrix} 0.075 & 0.12 & 0.075 \\ 0.124 & 0.20 & 0.124 \\ 0.075 & 0.12 & 0.075 \end{pmatrix}, \begin{pmatrix} 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 \\ 0.11 & 0.11 & 0.11 \end{pmatrix},$$

$$\begin{pmatrix} 0.0030 & 0.013 & 0.022 & 0.013 & 0.0030 \\ 0.0133 & 0.060 & 0.098 & 0.060 & 0.0133 \\ 0.0219 & 0.098 & 0.162 & 0.098 & 0.0219 \\ 0.0133 & 0.060 & 0.098 & 0.060 & 0.0133 \\ 0.0030 & 0.013 & 0.022 & 0.013 & 0.0030 \end{pmatrix}, \begin{pmatrix} 0.039 & 0.040 & 0.040 & 0.040 & 0.039 \\ 0.040 & 0.040 & 0.041 & 0.040 & 0.040 \\ 0.040 & 0.041 & 0.041 & 0.041 & 0.040 \\ 0.040 & 0.040 & 0.041 & 0.040 & 0.040 \\ 0.039 & 0.040 & 0.040 & 0.040 & 0.039 \end{pmatrix}.$$

Para introduzir ao processo de detecção de paralelismo, implementa-se uma função geradora de máscaras (código 5) e uma convolucionadora (código 6).

Código 5 Função R que gera máscaras de convolução gaussianas

```
1:  mascaraGaussiana ← function ( k, sigma ) {
2:    cont ← 0
3:    a ← matrix ( 0, k, k )
4:    for ( i in 1 : k )
5:      for ( j in 1 : k ) {
6:        a[i,j] ← exp ( - ( ( i * i + j * j ) / ( 2 * sigma * sigma ) ) )
7:        cont ← cont + a[i,j]
8:      }
9:      z ← 1 / cont;
10:   for ( i in 1 : k )
11:     for ( j in 1 : k )
12:       a[i,j] ← exp ( - ( ( i * i + j * j ) / ( 2 * sigma * sigma ) ) ) * z
13:   return ( a )
14: }
```

Código 6 Função R que efetua convoluções

```
1:  convolucionar ← function ( k, l, imagem, mascara ) {
2:    max ← dim ( mascara ) [1]
3:    g ← matrix ( 0, max, max )
4:    kM ← k + trunc ( max / 2 ) + 1
5:    lM ← l + trunc ( max / 2 ) + 1
6:    for ( i in 1 : max )
7:      for ( j in 1 : max )
8:        g[i, j] ← g[i,j] + mascara[i,j] * imagem[kM - i,lM - j]
9:    return (g[k,l])
10: }
```

Utilizando essas funções, define-se uma função que filtra uma imagem convolucionalmente com uma máscara gaussiana no código 7.

Seguindo o algoritmo proposto na seção 4, constrói-se o grafo de dependências do código 7 na figura 3. Nela, observe-se que as atividades que seguem em vértices não-seqüenciais podem ser executadas em paralelo.

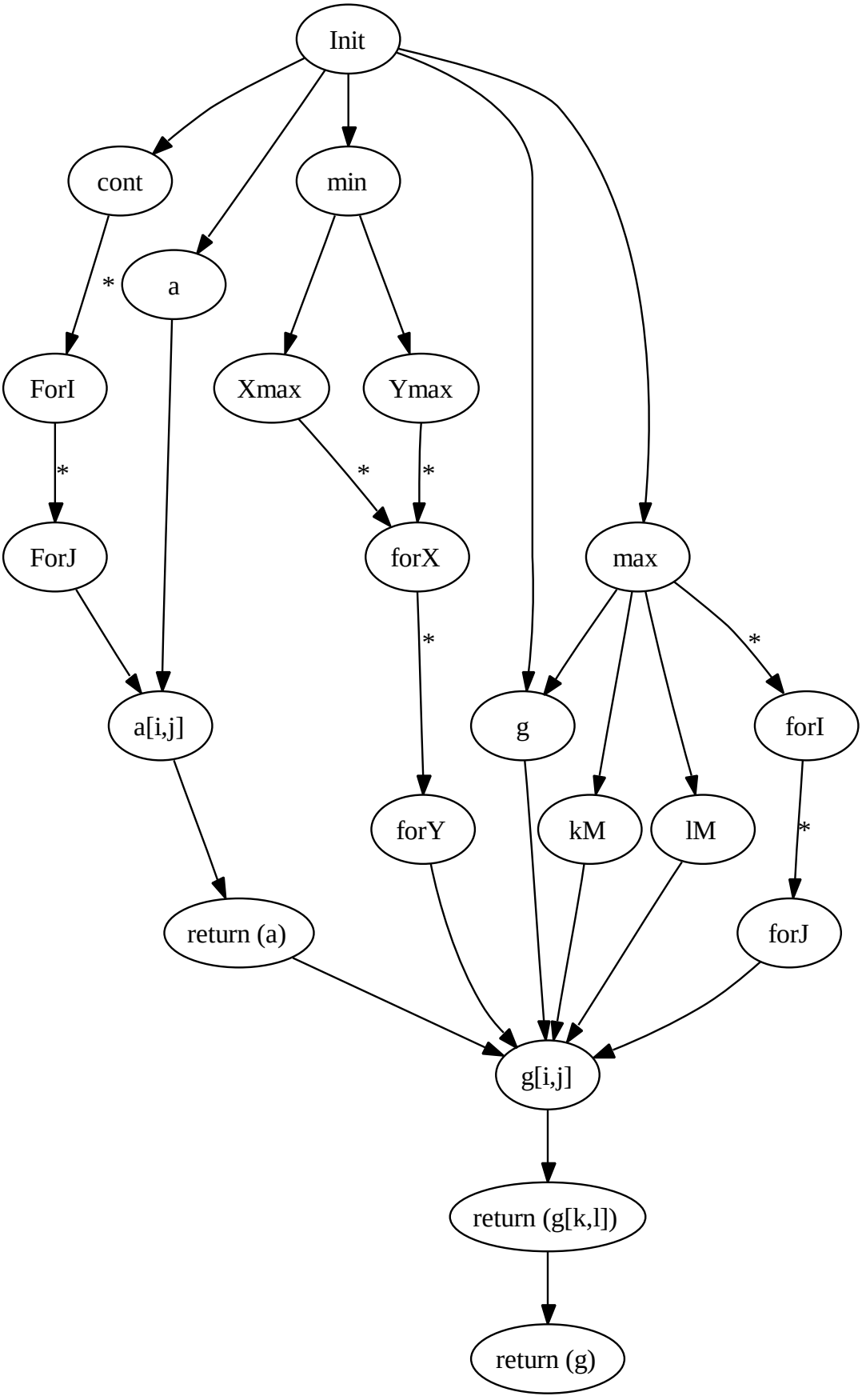

Figura 3. Grafo de dependências do código 7

Código 7 Filtro Gaussiano codificado em R

```
 1: filtroGaussiano ← function ( imagem, k, sigma ) {
 2:    mascara ← mascaraGaussiana ( k, sigma )
 3:    min ← ( k + 1 ) / 2
 4:    Xmax ← dim ( imagem ) [1] - min
 5:    Ymax ← dim ( imagem ) [2] - min
 6:    g ← imagem
 7:    for ( x in min : Xmax )
 8:       for ( y in min : Ymax )
 9:          g [x,y] ← convolucionar ( min, min, imagem ( x - ( k - 1 ) / 2 : x + ( k - 1 ) / 2, j - ( k
    - 1 ) / 2 : j + ( k - 1 ) / 2 ) , mascara )
10:    return ( g )
11: }
```

6. Resultados e conclusões

O uso de um grafo de dependências para representar um filtro gaussiano apresenta redução considerável no número de operações seqüenciais, reduzindo-o de 28 a 9 operações por pixel. Em uma imagem com, por exemplo, 1000×1000 pixels, utilizando-se uma máscara de lado $\ell = 3$, a economia em um cluster com cem processadores, o número total de operações seqüênciais se reduz de 3000012 para 30018. Assim sendo, o ganho no tempo de processamento é significativo, e será tanto mais relevante quanto mais interativa for a aplicação, isto é, quando se trata de uma abordagem exploratória.

Com o uso da interface oferecida por Yu (2006), pode-se implementar comunicação do R com a interface `MPI` (*Message-Passing Interface*), capaz de gerenciar múltiplos processadores. Assim, o código pode ser facilmente paralelizado na plataforma R.

Agradecimentos

Os autores agradecem ao CNPq por colaborar com a pesquisa.

Referências

Banerrjje, U., Eigenmann, R., Nicolau, A. e Padua, D. A. Automatic program parallelization, **Proceedings of the IEEE** v.81, p. 211 – 243, 1993.

Ferrante, J. e Ottenstein, K. J. The program dependence graph and its use in optimization, **ACM Transactions on Programming Languages and Systems** v.9, p. 319–349, 1987.

IEEE The world's leading professional association for the advancement of technology. Disponível em: ⟨`http://www.ieee.org`⟩, Acesso em: 3 ago. 2006.

Quinn, M. J. **Parallel Programming in C with MPI and OpenMP**, McGraw-Hill Professional, 2003.

Yu, H. **Rmpi: Interface (Wrapper) to MPI (Message-Passing Interface)**. Disponível em: ⟨`http://www.stats.uwo.ca/faculty/yu/Rmpi`⟩, R package version 0.5-2, 2006.