Reconstrução Paralelizada de Dados de Ressonância Magnética de Fluxo em Processadores Multi-Núcleo

Rosana Ribeiro Lima, João Luiz Azevedo de Carvalho (orientador)
Departamento de Engenharia Elétrica - ENE
Universidade de Brasília - UnB
Brasília - DF
rosanalima@gmail.com

***Abstract*— This paper aims to show an alternative way of fastening the reconstruction of flow images obtained by magnetic resonance imaging. Here, the parallelized reconstruction method will be presented in different situations: with uniformly or non-uniformly sampled data; with or without UNFOLD or *view sharing* techniques, and others. The result was a great reduction of the time required for the reconstruction. The main goal is to incentivize the parallelized use of multi-core processors (or multiple physically separated processors), which showed to be an efficient and practical tool for high speed processing in the field of biomedical engineering. Besides, other topics about MRI data processing were studied and will be discussed.**

***Resumo*—Este artigo pretende mostrar uma forma alternativa de acelerar a reconstrução de imagens de fluxo obtidas por equipamentos de ressonância magnética. Aqui, o método de reconstrução paralela será apresentado em diferentes situações: com dados amostrados uniformemente ou não-uniformemente; com ou sem técnicas de UNFOLD e *view sharing*, entre outros. O resultado foi uma grande redução no tempo necessário de reconstrução. O objetivo principal é incentivar o uso de multiprocessadores paralelamente (sejam eles múltiplos núcleos em um mesmo chip ou múltiplos processadores fisicamente distintos), o que mostrou ser uma ferramenta eficiente e prática no campo da engenharia biomédica. Além disso, outros tópicos sobre processamento de imagens de ressonância magnética foram trabalhados e são aqui discutidos.**

***Keywords*— MRI, parallelized reconstruction, multi-core processors.**

I. Introdução

As doenças cardiovasculares representam hoje uma das maiores causas de óbitos no mundo, segundo a Organização Mundial de Saúde (OMS). Nessa conjuntura, os avanços do conhecimento da medicina a fim de melhorar a condição de vida das pessoas devem ser acompanhados por avanços correspondentes na área da engenharia biomédica, e vice-versa, a fim de que os maiores benefícios sejam plenamente desfrutados. Reforça-se aqui, portanto, a importância do investimento de pesquisa neste campo, e particularmente no estudo do processamento de imagens de ressonância magnética (RM), que é o cenário do assunto deste trabalho. Na abordagem deste artigo, os fundamentos de RM serão primeiramente apresentados (seção I.A), seguidos de conceitos e técnicas relacionados ao processamento dessas imagens (seção I.B). Na segunda parte, consta a metodologia utilizada no trabalho (II), a pesquisa propriamente dita e os resultados obtidos (III). No terceiro bloco, será feita uma análise de resultados, conclusões e considerações (IV), além da bibliografia (V).

A. Fundamentos da Ressonância Magnética

Em um aparelho de ressonância magnética, os *spin*s do corpo que se quer visualizar estão naturalmente alinhados conforme um campo magnético externo aplicado, B_0. Em torno desse campo, ocorre o movimento de precessão dos *spin*s. A freqüência da precessão é diretamente proporcional ao seu campo magnético gerador. Como os prótons estão defasados entre si, a rede de magnetização resultante é na mesma direção e sentido de B_0. Seja essa magnetização chamada de M_0. Então, aplica-se um pulso de radio-freqüência (RF) em uma direção perpendicular ao campo original. Esse pulso tem uma componente de campo magnético que será denominada B_1, e tem intensidade mais fraca do que a de B_0. Se a freqüência com que o próton gira ao redor de B_0 for a mesma freqüência do pulso de RF, ocorrerá o fenômeno da ressonância: aparecerá outra precessão no próton, dessa vez ao redor de B_1. A figura 1 ilustra o fenômeno de precessão no próton.

R. R. Lima, Universidade de Brasília (UnB), Brasília - DF, Brasil

J. L. A. Carvalho, Universidade de Brasília (UnB), Brasília - DF, Brasil

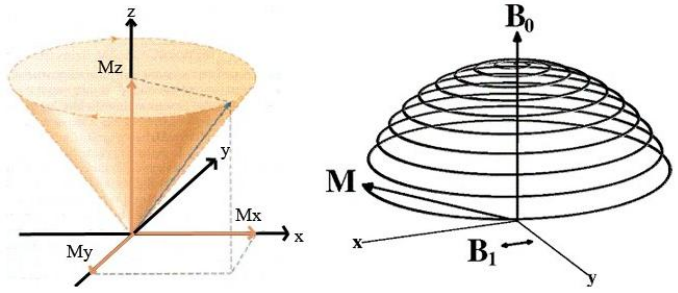

Figura 1: ilustração esquemática da precessão de um próton: (esquerda) sem o pulso de RF. M_x, M_y e M_z indicam os componentes da rede de magnetização. Como os prótons estão fora de fase entre si, a resultante de várias componentes M_x e M_y deve ser nula; (direita) no caso de ressonância, com o pulso de RF. Como este pulso acompanha a precessão em torno de B_0, surge outra precessão, em torno de B_1, proporcional à intensidade deste.

Após a aplicação do pulso, um vetor de magnetização é criado, perpendicular a B_0, chamado M_{xy}. Quando o pulso é desligado, entretanto, a magnetização no sentido xy tende a diminuir, enquanto no sentido do eixo z ocorre uma recuperação da magnetização M_z ao longo do tempo. O tempo decorrido nesses processos (diminuição de M_{xy}, aumento de M_z) varia segundo duas constantes (T_2 e T_1, respectivamente) e segue as equações:

$$(1) \quad M_z(t) = M_0\left(1 - e^{\frac{-t}{T_1}}\right)$$

$$(2) \quad M_{xy}(t) = M_0 e^{\frac{-t}{T_2}}$$

Em que T1 e T2 variam conforme propriedades do tecido. Na máquina de RM, vários pulsos de RF devem ser enviados, de forma que valores de t bem escolhidos serão capazes de gerar contraste entre os tecidos. No processo de imageamento, esses valores de t pré-determinados são chamados de TR (tempo de repetição, valor de t na equação 1) e TE (tempo de eco, valor de t na equação 2). Sabe-se que o maior contraste ocorre quando se escolhe um menor TR e um maior TE, entretanto esta situação corresponde também a um baixo valor de razão sinal-ruído (SNR), pois correspondente ao maior decaimento. Uma solução de compromisso deve ser buscada então para esses parâmetros, sempre ressaltando o que é mais importante na aquisição de dados realizada. Importante salientar que há outros parâmetros que devem ser ponderados para a melhor construção final da imagem, como o número de aquisições, o tempo de inversão, entre outros. Uma leitura aprofundada sobre este assunto pode ser feita no capítulo 17 da referência [1].

O sinal de magnetização emitido pelos prótons no eixo x é justamente o sinal que será captado pela máquina. Entretanto, a construção da imagem de um pedaço do corpo humano utilizando ressonância magnética não é direta. O que se obtém inicialmente é um espectro da imagem desejada no domínio da transformada de Fourier (kx e ky), chamado de espaço-k. Para definir kx e ky, utiliza-se a codificação de freqüência e de fase, respectivamente (para isso, basta colocar um gradiente adequado no eixo x no momento da leitura do sinal; e semelhantemente pôr no eixo y entre a emissão do pulso RF e a leitura do sinal). Ainda, é preciso definir a altura no eixo z em que a imagem será feita. Para isso, utiliza-se um gradiente no eixo z no momento de emissão do pulso de RF. Com isso, para obter a imagem a ser processada, basta fazer a leitura do sinal para diversos gradientes de fase diferentes. Tal processo corresponderá à leitura de várias linhas distintas, com um tempo de separação entre elas. A imagem que desejamos (no domínio espacial) será o resultado do processamento do sinal obtido com essas leituras (no espaço-k).

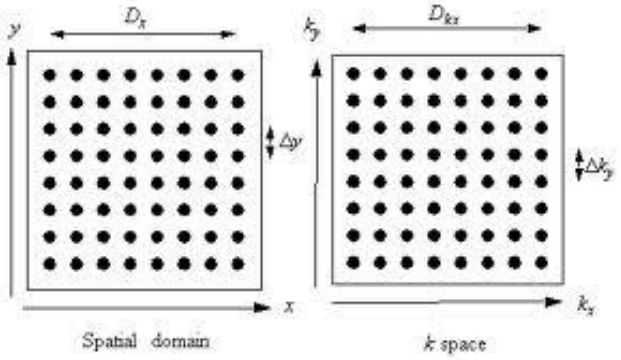

Figura 2: comparação da imagem obtida com a imagem desejada: (esquerda) corresponde à imagem desejada. O domínio espacial, ou *spatial domain*, tem parâmetros que podem ser encontrados pelas seguintes equações: $\Delta y = 1/(N\Delta k_y)$ e $\Delta x = 1/(N\Delta k_x)$; ainda: $Dy = 1/\Delta k_y$ e $Dx = 1/\Delta k_x$; (direita) corresponde à imagem do sinal obtido pelo aparelho de ressonância magnética (espaço-k ou *k-space*).

Para a obtenção de imagens de fluxo por ressonância magnética, o mesmo princípio de funcionamento da máquina é seguido. O que caracteriza os dados de fluxo essencialmente são a seqüência e os intervalos em que serão aplicados os

gradientes magnéticos. Num contexto de muitas doenças relacionadas ao comprometimento de vasos sanguíneos hoje, imagens de fluxo de ressonância são importantes para permitir a detecção e tratamento daquelas.

A codificação espiral FVE (codificação de velocidade por Fourier) é um recurso que permite a rápida aquisição de dados de fluxo, e apresenta uma amostragem não-uniforme. Alguns dados tratados na pesquisa foram do tipo espiral FVE, e técnicas específicas como *gridding* e *nufft* foram neles aplicados. Tais métodos serão discutidos a seguir.

B. Alguns Conceitos e Técnicas Relacionados ao Processamento de Sinais de RM

O conceito mais importante a ser utilizado é o da transformada de Fourier no domínio discreto, bem como suas formas computacionais, como a FFT (*fast Fourier transform*) e a FFTW (*fastest Fourier transform in the west*). Ela permite a transição de figuras e funções entre o domínio da freqüência e o domínio espacial (ou temporal). Ela possibilita a reconstrução de dados de ressonância magnética.

Outros conceitos importantes de serem apresentados são: UNFOLD, *view sharing* e *gridding*. Tais conceitos serão apresentados aqui de maneira prática, de maneira a cumprir o foco do trabalho. Para um maior aprofundamento, recomenda-se a leitura das referências [6], [7] e [8].

O *gridding* é um método de reconstrução de dados que foram amostrados de maneira não-uniforme (figura 3). Para isso, utiliza-se a interpolação dos dados com alguma função pré-determinada (ex: sinc, Gaussiana, Kaiser-Bessel, piramidal), também chamada de *kernel*.

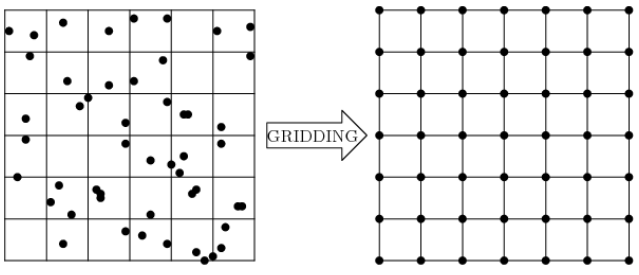

Figura 3: ilustração do que resulta da aplicação do método de *gridding*. No caso da RM, quando se utiliza a amostragem em espiral por exemplo, no lugar da amostragem cartesiana, a maior parte dos pontos fica concentrada no centro do espaço-k. O *gridding* pode então ser aplicado para uniformizar o espaçamento entre os pontos e permitir uma reconstrução direta para o domínio espacial.

Quando se utiliza o *gridding*, deve-se escolher um fator de *oversampling* (superamostragem) a fim de deixar o *field-of-view* (FOV) da imagem reconstruída com o tamanho adequado. Além disso, deve ser aplicada a deapodização, que corresponde à divisão da imagem reconstruída por uma função de ajuste. Essa função de ajuste é a transformada de Fourier do *kernel* utilizado, e pode ser calculada analiticamente.

A fim de comparar o desempenho do *gridding* seguido de deapodização, foi utilizada ainda a *nufft* (*non-uniform fast Fourier transform*, ou transformada rápida de Fourier não-uniforme).

O método de UNFOLD pode ser utilizado para permitir uma reconstrução eficiente e com baixo ruído de uma seqüência de imagens apesar da subamostragem. Ele baseia-se na idéia de que, ao subamostrar quadros no eixo ky de maneira alternada entre eles, um quadro estaria deslocado em relação ao outro, e no domínio espacial isso corresponde a uma diferença na fase. Se a subamostragem for de uma linha a cada duas, essa diferença vai fazer que as imagens se alternem entre positiva e negativa, por causa da diferença de fase. Analisando cada ponto da imagem ao longo do tempo e seu respectivo espectro, vê-se então que aparece um componente de freqüência para mais altas freqüências, indicando a alteração rápida entre positivo e negativo. Pode-se fazer uma filtragem passa-baixa capaz de amenizar significativamente esses artefatos. Caso o filtro a ser utilizado seja ideal (rect), isso corresponderia a fazer uma interpolação com a função sinc para cada ponto da imagem ao longo do tempo. Pode-se, entretanto, utilizar outros tipos de filtro, com amenização mais suave (como o gaussiano).

O método de *view sharing* é muito semelhante ao do UNFOLD, e se baseia no aproveitamento de pontos anteriormente amostrados para permitir uma subamostragem de maneira alternada entre uma seqüência de quadros de imagens. A grande diferença é que para o *view sharing* utiliza-se uma função linear como interpoladora, e apenas os quadros mais imediatos em que o ponto foi

amostrado devem ser levados em consideração. Trata-se, portanto, de um método mais simples do que o UNFOLD.

Alguns outros conceitos de processamento de imagens serão explicados no início da seção III.

II. METODOLOGIA

O software utilizado no trabalho foi o MATLAB, por ter funções gráficas e matemáticas que facilitassem o trabalho desenvolvido.

Primeiramente, tendo a teoria já conhecida, foram realizadas atividades que fixassem o conteúdo apresentado. Em seguida foi desenvolvido um algoritmo para a reconstrução paralela de dados de uma imagem amostrada de maneira não-uniforme. Testou-se então o algoritmo para um grande volume de dados, e foram feitas as alterações necessárias. Em seguida, colocou-se um vídeo de um *phantom* em situação de UNFOLD e, nele, aplicou-se a reconstrução paralela. Para a medição do tempo de processamento, foram utilizados para teste processadores *dual core* (dois núcleos) e *quad core* (quatro núcleos).

III. PESQUISA E RESULTADOS OBTIDOS

A. Prática de algumas técnicas de processamento de imagens aplicadas à reconstrução de uma imagem

A primeira atividade de processamento de imagem de RM realizado no MATLAB consistiu na introdução de *zero-padding* (introdução de linhas e/ou colunas zeradas intercaladas) e subamostragem, observando os resultados de cada técnica.

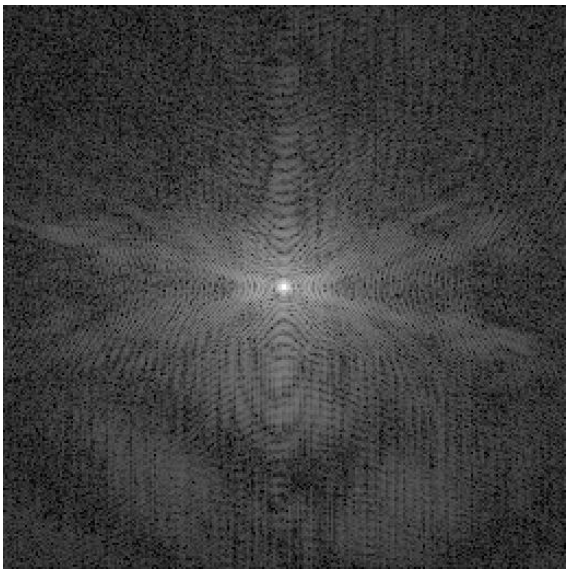

Figura 4: Espaço-k da imagem a ser reconstruída, tamanho 256 pixels x 256 pixels. Foi utilizada a função logarítmica para facilitar a visualização.

O *zero-padding* tem a função de gerar uma imagem com maior tamanho, pela inserção de zeros. A figura 4 e a figura 5 ilustram o espaço-k com e sem o *zero-padding*. Foi utilizada a função logarítmica para facilitar a visualização.

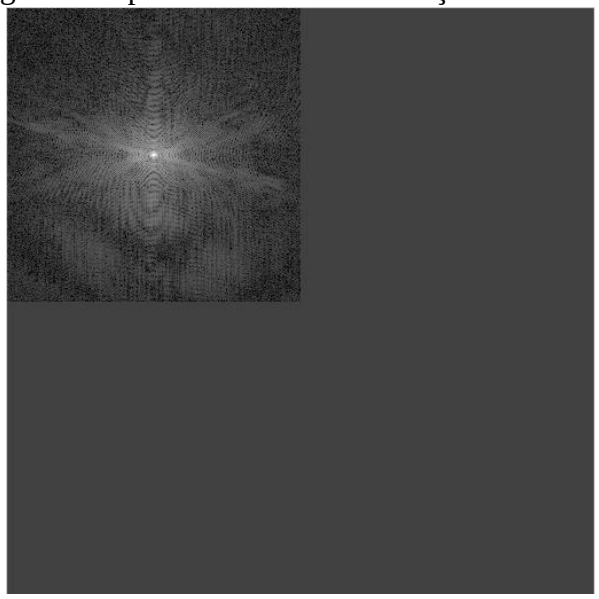

Figura 5: Espaço-k da imagem a ser reconstruída com zero-padding. Tamanho 512 pixels x 512 pixels. O tamanho da imagem reconstruída aumenta, mas não há ganho de resolução. Foi utilizada a função logarítmica para facilitar a visualização.

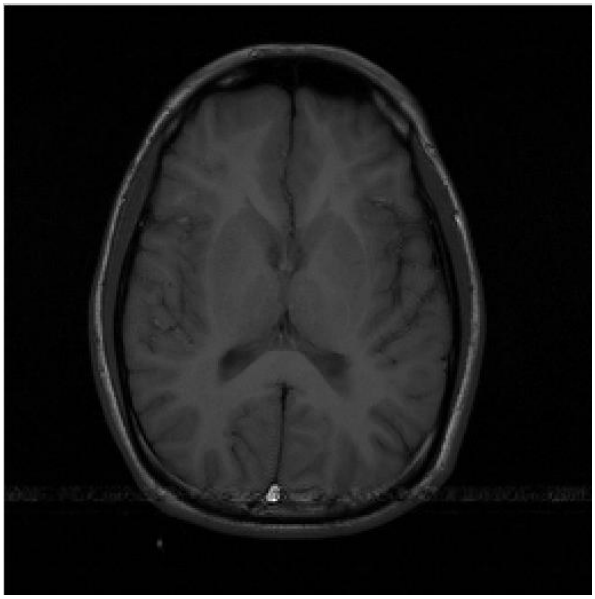

Figura 6: Imagem obtida a partir do espaço-k apresentado na figura 4. Tamanho 256 pixels x 256 pixels. A imagem obtida a partir do espaço-k apresentado na figura 5 é semelhante, mas com tamanho 512x512. Há uma ampliação da imagem, mas não ganho de resolução.

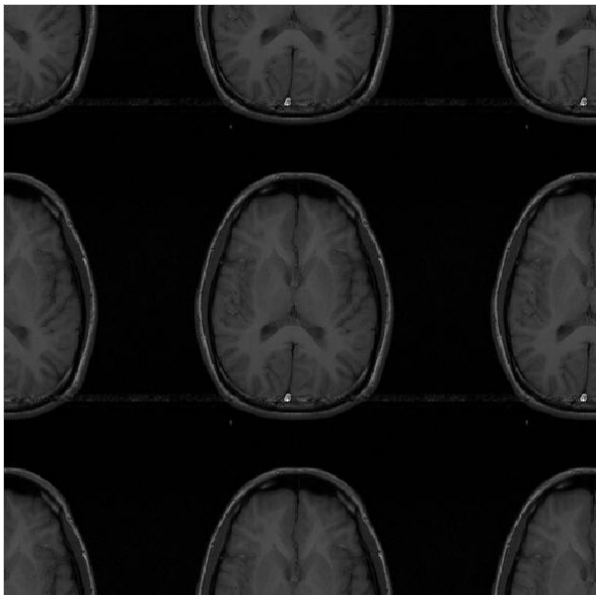

Figura 7: Imagem obtida após a inserção de linhas e colunas com elementos nulos intercaladamente no espaço-k. A imagem reconstruída tem o mesmo número de pixels da transformada após a inserção (512 pixels x 512 pixels).

A imagem reconstruída da figura 5 tem o mesmo número de pontos de sua FFT (ou seja, no caso, suas dimensões são o dobro do que eram antes (antes eram 256 pixels x 256 pixels, agora são 512 x 512).

No processo de inserção de linhas ou colunas com valor nulo intercaladamente no espaço-k, também não se tem perda nem ganho de informação. Nesse caso, o FOV da imagem reconstruída aumenta, mas aparecem réplicas da imagem original nos cantos da figura. Para entender esse resultado pode-se pensar no espaço-k como a multiplicação entre um trem de impulsos e o sinal analógico obtido pelo sensor da máquina de RM. Sabe-se que, quando um trem de impulsos tem o espaçamento entre os impulsos aumentado, sua transformada terá esse espaçamento diminuído. No domínio espacial, portanto, ocorre uma convolução da imagem reconstruída com um trem de impulsos de distância menor entre eles. Quanto mais linhas ou colunas intercaladas forem inseridas, mais réplicas aparecerão. A figura 7 ilustra o resultado desse processo.

Quando se zeram linhas ou colunas, ou seja, se subamostra no espaço-k, acontecerá aliasing no domínio espacial. Nesse processo, há perda de informação. Isso ocorre porque a freqüência mínima de amostragem definida pela taxa de Nyquist deixa de ser cumprida, e as imagens passam a se sobrepor. Geralmente, a freqüência de Nyquist é avaliada no domínio da freqüência, mas no caso da subamostragem no espaço-k, o aliasing ocorre no domínio espacial. O teorema de Nyquist define que a taxa de amostragem deve ser no mínimo o dobro da largura de banda ocupada pela imagem. As figuras 8, 9 e 10 a seguir ilustram as imagens reconstruídas da subamostragem no espaço-k.

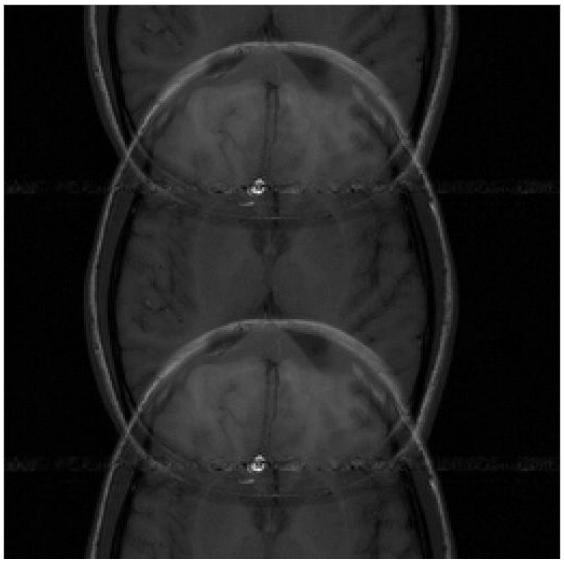

Figura 8: Imagem obtida após descarte das linhas pares (k_y subamostrado) no espaço-k. Tamanho da imagem: 256 pixels x 256 pixels

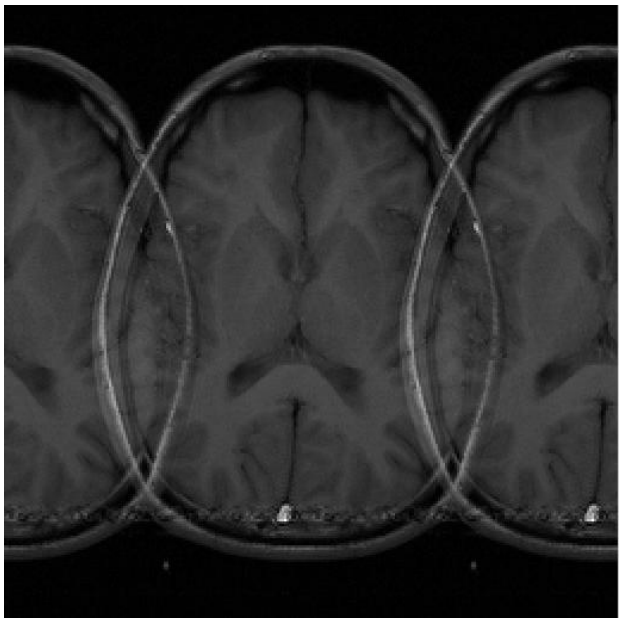

Figura 9: Imagem obtida após descarte das colunas pares (k_x subamostrado) no espaço-k. Tamanho da imagem: 256 pixels x 256 pixels

O caso da figura 8 é mais típico para imagens de RM do que o da figura 9. Isso porque o processo de obtenção de uma linha inteira é consideravelmente rápido. Mas apenas uma é obtida por vez, e há um tempo de recuperação entre elas. Para que o paciente não fique muito tempo dentro da máquina sem se mexer, deve-se procurar uma solução de compromisso, e um número demasiadamente grande de linhas pode não ser uma boa escolha.

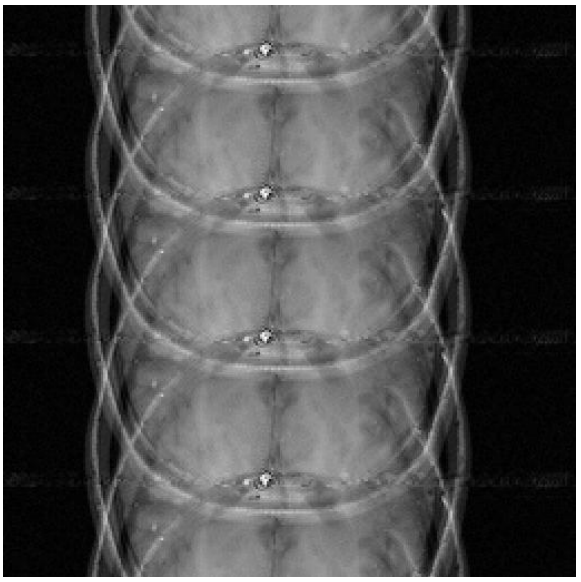

Figura 10: transformada inversa após a subamostragem de zerar 3 em cada 4 linhas no espaço-k. Tamanho da imagem: 256 pixels x 256 pixels

Por fim, passou-se um filtro ideal passa-baixas 64 pixels x 64 pixels no espaço-k mostrado na figura 4. O resultado obtido, conforme esperado, foi o borramento das bordas e dos detalhes, que são os elementos correspondentes às altas freqüências. Tal processo corresponde à interpolação com uma sinc no domínio espacial, e está ilustrado na figura 11.

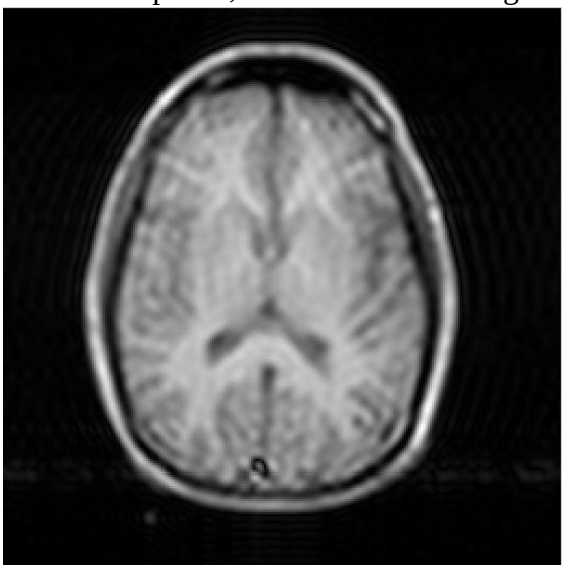

Figura 11: Imagem obtida após a filtragem do espaço-k (subamostrado) com passa-baixas ideal com ¼ do tamanho da imagem. Tamanho total: 256 pixels x 256 pixels.

B. Implementação das técnicas de gridding e deapodização

A próxima atividade foi a implementação de um código adequado para *gridding* e deapodização de uma função com rects. Esse código seria

posteriormente utilizado no processo de reconstrução paralela.

Para teste da função de *gridding*, utilizou-se a seguinte função inicial, ilustrada na figura 12.

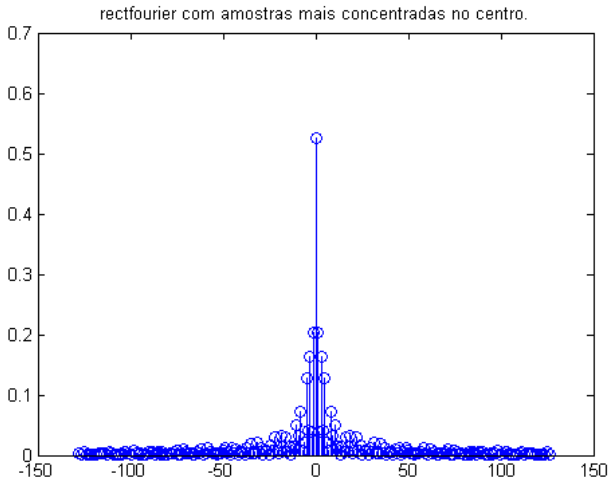

Figura 12: Espaço-k amostrado (com amostras concentradas nas baixas freqüências) da função que se quer reconstruir (três rects)

O processo de *gridding* foi feito de duas formas e comparado: primeiramente com um interpolador ideal. Neste caso, a função de deapodização é uma rect com largura no mínimo igual à largura da função, portanto não precisou ser aplicada. A outra forma foi por meio de uma Gaussiana. A função de deapodização foi, portanto, outra gaussiana. Para os dois casos, foi utilizado um fator de superamostragem igual a 2. A figura 13 contém o resultado da transformada inversa após o *gridding* com a Gaussiana, antes do processo de deapodização.

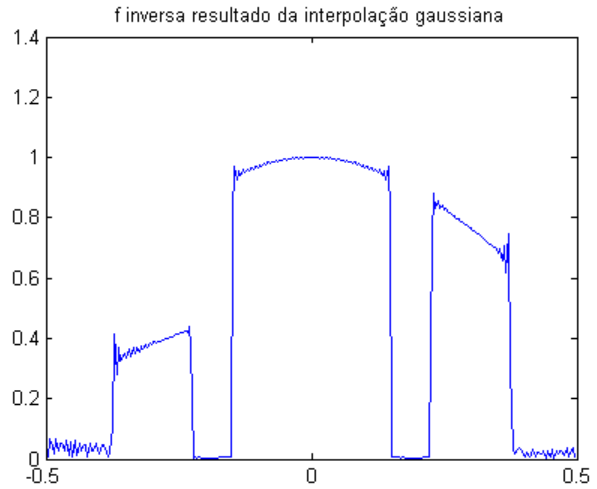

Figura 13: Transformada inversa após a interpolação Gaussiana, ainda sem deapodização.

Foi utilizada também uma função de reconstrução exata de dados amostrados não-uniformemente implementada pelo orientador, a *idrft*. Após o devido ajuste das formas de reconstrução, o seguinte gráfico (figura 14) ilustra o resultado obtido.

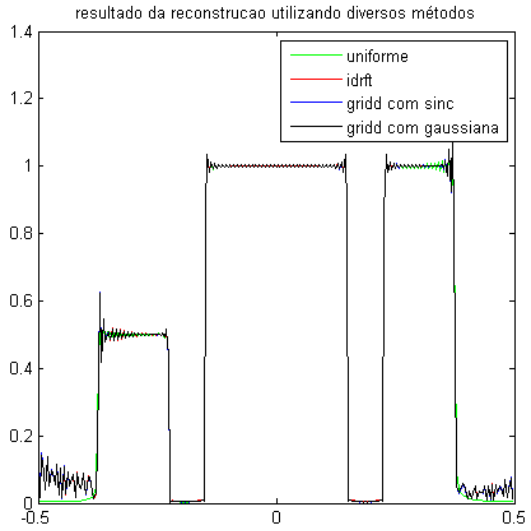

Figura 14: Resultado da reconstrução dos dados adequadamente utilizando diversos métodos. O caso uniforme mencionado na legenda se refere à reconstrução caso os dados tenham sido amostrados de maneira uniforme. Após a reconstrução, todos estão uniformes.

Os resultados se mostraram bem semelhantes e próximos ao resultado teórico esperado. Para uma análise mais precisa do erro, fez-se uma análise do erro obtido para cada reconstrução, e obteve-se o seguinte gráfico.

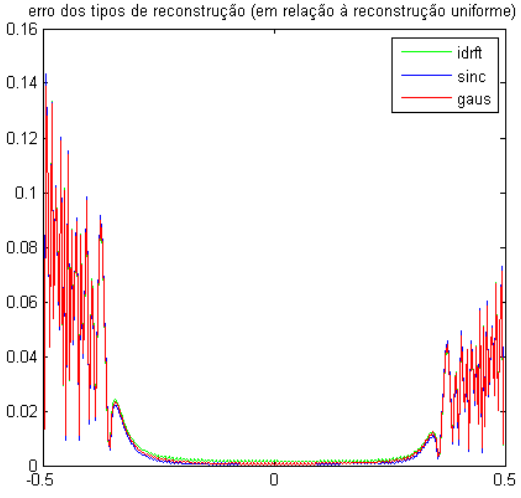

Figura 15: erro obtido da reconstrução dos dados por diversos métodos. Nas extremidades, o erro é maior.

Deve-se notar que o *gridding* com a função sinc tem o resultado final com menor erro dentre as três, apesar da diferença ser pequena. Em seguida, o *gridding* com função Gaussiana. Os erros foram calculados pelo MATLAB utilizando o valor RMS dos erros aumentados por uma escala de 1000. Foram obtidos os seguintes erros: 1,0105 para a *idrft*, 0,9846 para o *gridding* com a sinc e 1,0102 para o *gridding* com a gaussiana.

C. Criação de uma função bidimensional de gridding e deapodização

Viu-se então que o algoritmo para o *gridding* funcionou bem, e apresentou baixo erro. Em seguida, o algoritmo foi desenvolvido para um caso genérico, e em forma 2D. Para isso, utilizou-se como base um algoritmo de *gridding* simples e rápido. O *gridding* simples utiliza um *kernel* piramidal de uma forma engenhosa, mas não leva em consideração a densidade dos dados, não faz sobreamostragem, nem deapodização. Adicionando essas características ao algoritmo, a reconstrução de uma imagem que fora amostrada de maneira não uniforme mostra-se não só possível, mas muito eficiente. A seguir, figuras que mostram a reconstrução da imagem com o *gridding* simples e em seguida com o *gridding* aprimorado.

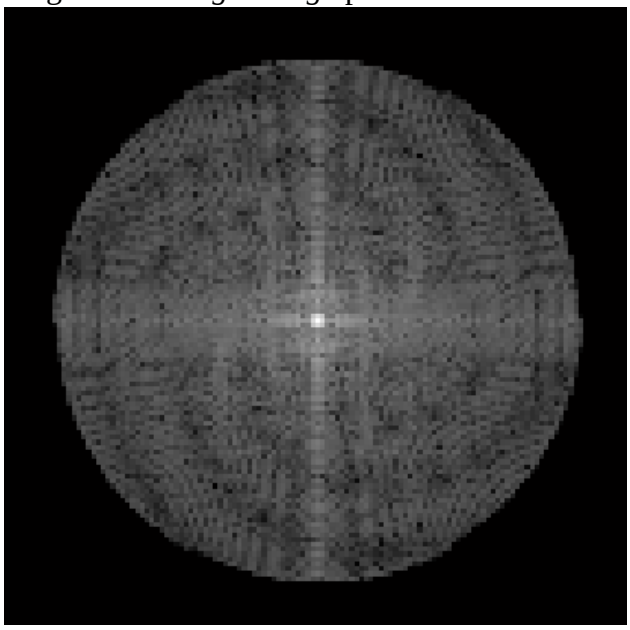

Figura 16: Resultado do *gridding* simples de transformada não-uniformemente amostrada dos dados a serem reconstruídos. Tamanho da imagem: 128 pixels x 128 pixels.

A seguir, a imagem resultante transformada inversa de Fourier do espaço-k apresentado na figura 16.

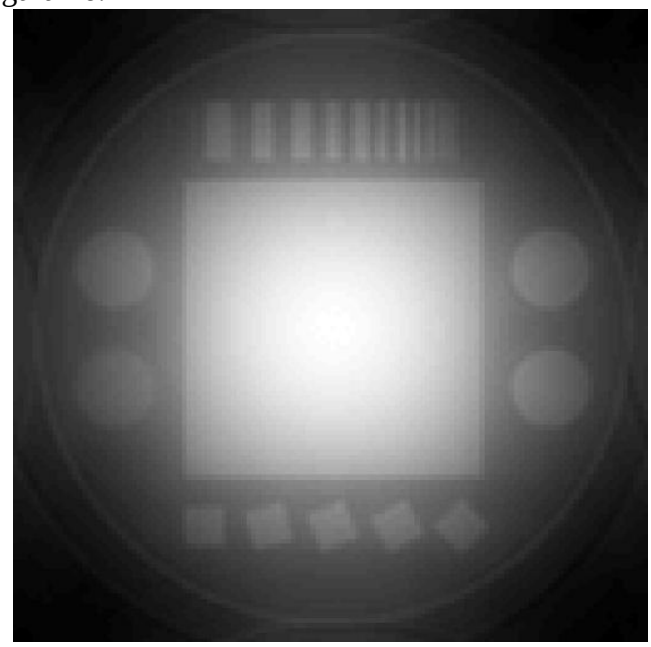

Figura 17: Imagem obtida a partir do espaço-k apresentado na figura 16. Como foi utilizado o *gridding* simples para a uniformização dos dados, o FOV tem a aparência de estar cortado, indicativo de que não houve o fator de sobreamostragem no domínio do espaço-k. Além disso, no centro da imagem os pontos têm um valor significativamente maior, indicando a falta de deapodização. Tamanho da imagem: 128 pixels x 128 pixels.

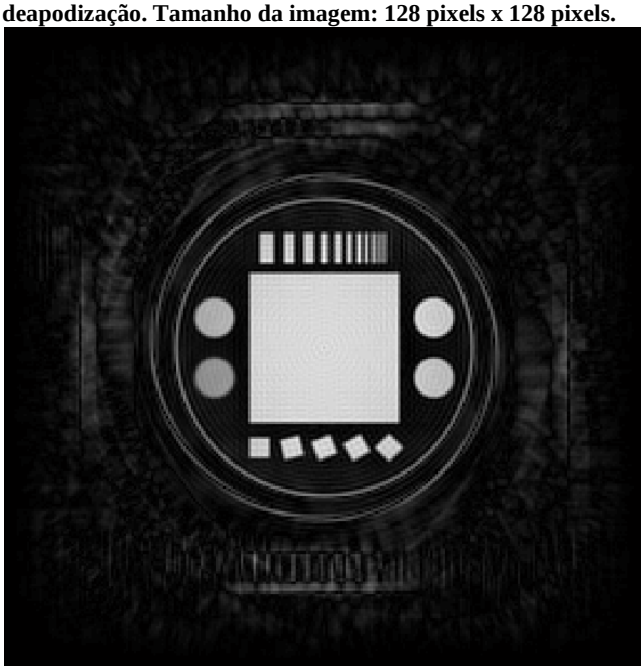

Figura 18: Resultado da função desenvolvida de *gridding* aprimorado no MATLAB (grid1weightsOVS2deapod.m, que será utilizada mais à frente). Os dados chamados dentro dessa função passaram por essencialmente três etapas: *gridding* com consideração de pesos e sobreamostragem de fator 2, transformada inversa de Fourier, e por fim a deapodização. Tamanho da imagem: 256 pixels x 256 pixels.

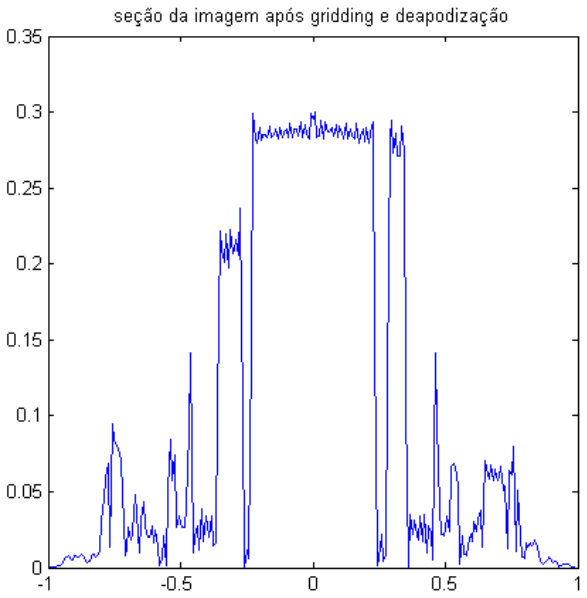

Figura 19: Gráfico ilustrando uma seção do meio da imagem mostrada na figura 18 (pontos da coluna 128). Observa-se o efeito da deapodização, uniformizando os pontos do centro e das extremidades da imagem. Observa-se também que o ruído nas regiões das extremidades são maiores.

D. Aplicação da função gridding na reconstrução paralela (e comparação com outras funções de reconstrução)

A seguir, começou-se o desenvolvimento de um código para a reconstrução paralela dos dados em multiprocessadores. A organização do código de forma a executar ações importantes em menos tempo passou a ser considerada com maior relevância. Pesquisou-se então sobre os recursos apresentados pelo MATLAB para processamento paralelo. Algumas ferramentas presentes no *MATLAB Parallel Computing Toolbox*: a função *spmd*, o uso de *codistributed arrays*, e o *parfor*. Além deles, há ferramentas desenvolvidas por usuários do MATLAB, como o código desenvolvido por Markus Buehren para multiprocessadores. Várias funções específicas para trabalhar diretamente com um ou outro núcleo também foram vistas (como o labSend, labReceive, labProb, entre outros). Em meio a tantas ferramentas, para a abordagem da pesquisa realizada, foi escolhida a função *parfor* (algumas das outras ferramentas não estavam disponíveis para a versão do MATLAB utilizada, e a função *parfor* demonstrou ser prática e capaz de suprir às necessidades do problema). Para realizar os testes, primeiramente utilizaram-se os mesmos dados da etapa anterior, a saber, rt_spiral.mat.

O primeiro código desenvolvido tinha o objetivo de reconstruir os dados de apenas uma imagem. Para isso, fez-se um algoritmo que separasse em N blocos os dados recebidos (onde N é o número de núcleos de processamento disponíveis) e executasse o processamento de cada bloco simultaneamente. Para a execução simultânea, utilizou-se o *parfor* (abreviação de *parallel for*), que cria um loop da mesma forma que o *for*, mas separando as repetições por bloco. Há, portanto, algumas restrições para o uso dessa ferramenta, que foram estudadas.

Há seis tipos de variáveis que podem estar dentro de um loop *parfor*. Se o MATLAB não reconhecer alguma das variáveis escritas como um dos tipos pré-determinados, o programa não é capaz de executar o código. Uma ilustração da *MathWorks* é mostrada a seguir ilustrando cada tipo dessas variáveis.

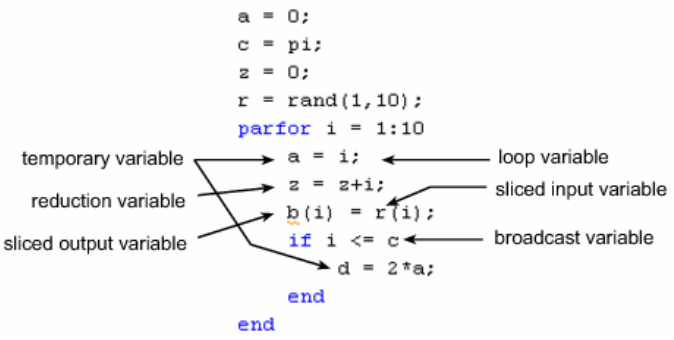

Figura 20: Ilustração da Mathworks com demonstração de código com os tipos de variáveis que o MATLAB é capaz de reconhecer dentro de um loop *parfor*.

A variável de loop, ou *loop variable*, é a que cresce dentro do loop, e serve de índice para os vetores. A *sliced input variable* é a que tem os dados acessados conforme o índice indicado pelo *parfor*, ou seja, varia a cada iteração. A *sliced output variable* tem seus elementos definidos conforme o índice indicado pelo *parfor*, a cada iteração. A variável *broadcast* está definida fora do loop e pode ser acessada dentro dele, mas sem alterar o valor nela armazenado. A variável *reduction* vai incrementando seu valor a cada iteração, e independe da ordem das iterações. A variável *temporary* é criada dentro do loop *parfor*, mas não pode ser acessada fora dele.

No desenvolvimento do algoritmo, tomou-se sempre o cuidado de trocar construções que mantinham variáveis que não se encaixam em nenhum desses tipos por outras com execução permitida pelo *software*. Obteve-se então o código:

```
%Rosana R Lima 2011
%Universidade de Brasília

clear all, close all;
load rt_spiral.mat
Nim=128;
d=d(:);
k=k(:);
w=w(:);
N=length(d);
Ncores=feature('NumCores');
NporLoop=ceil(N/Ncores);
m=zeros(2*Nim,2*Nim);
%---------------------------------------
%PARA A OPÇÃO INUFFT, DESCOMENTAR:
%         cd fessler
%         setup
%         cd ..
%         om(:,1) = 2*pi*real(k(:)); %kspace trajectory
%         om(:,2) = 2*pi*imag(k(:));
%         Nd=[Nim, Nim];
%         ovg=4; %OVG = overgrid factor
%---------------------------------------
matlabpool open 2
disp('-------------------------------------------------');
disp('Tempo de execução utilizando o parfor:');
j=(1):(NporLoop);
pause(.1)
tic
parfor i=0:Ncores-1
%         %-------------------------------------------------
%         %PARA A OPÇÃO IDRFT:
            dx=d(j+i*NporLoop);
            wx=w(j+i*NporLoop);
            kx=k(j+i*NporLoop);
            m=m+idrft2d(dx,wx,kx,2*Nim);
%         %-------------------------------------------------
%         %PARA A OPÇÃO INUFFT:
%             at=j+i*NporLoop;
%             m=m+calcporLoop2(om,Nd,ovg,at,d,w);
%         %-------------------------------------------------
%         %PARA A OPÇÃO DO KERNEL TRIANGULAR
%             dx=d(j+i*NporLoop);
%             wx=w(j+i*NporLoop);
%             kx=k(j+i*NporLoop);
%             m=m+grid1weightsOVS2deapod(dx,kx,Nim,wx);
end
toc
pause(.1)
matlabpool close
```

Figura 21: Código desenvolvido para a reconstrução paralela com multiprocessadores de rt_spiral.mat, por três formas diferentes: utilizando a função *idrft*, a *nufft*, ou a de *gridding* com *kernel* piramidal, desenvolvida anteriormente.

Sabe-se, entretanto, que essa configuração para uma quantidade muito pequena de dados (com tempo de execução muito curto) não é vantajosa, pois a entrada do laço *parfor* e o *matlabpool open* têm um tempo considerável de execução. Assim, para um grande volume de dados a melhoria no tempo de execução fica perceptível. Levando isso em consideração, o resultado foi conforme o esperado, como observado pela *idrft*, que é, dentre as três, a forma de reconstrução mais demorada: com o *parfor*, o tempo de execução foi de cerca de 51,3 segundos em um *dual core*, e cerca de 20,0 segundos em um *quad core* (para utilizar os 4 núcleos do *quad core*, é necessário alterar o código onde está *matlabpool open 2* e colocar *matlabpool open 4*). Sem a preocupação com o uso paralelo dos multiprocessadores, o tempo de execução do código chega a ser de 100,0 segundos no *dual core* e de 76,5 segundos no *quad core*. O código utilizado mostrado a seguir, na figura 22.

```
%Rosana R Lima 2011
clear all, close all;
load rt_spiral.mat
Nim=128;
d=d(:);
k=k(:);
w=w(:);
N=length(d);
disp('-------------------------------------------------');
disp('Tempo de execução sem o parfor:');
pause(0.1);
tic
m=idrft2d(d,w,k,2*Nim);
toc
pause(.1)
```

Figura 22: Código desenvolvido para a reconstrução dos dados por *idrft* sem a preocupação com o uso paralelo de multiprocessadores.

Portanto, no caso da *idrft*, para esta reconstrução, observou-se uma redução de 48,7% do tempo para o *dual core* e de 73,8% para o *quad core* quando houve reconstrução com processamento paralelo entre os núcleos. Foi, portanto, uma redução significativa.

Para analisar a *nufft*, descomentaram-se os trechos referentes à *nufft* e comentou-se o trecho da *idrft*. A função calcporLoop2 é dada por:

```
function x=calcporLoop2(om,Nd,OVG,at,d,w)

dx=d(at);
wx=w(at);
omx(:,1)=om((at),1);
omx(:,2)=om((at),2);
st = nufft_init(omx, 2*Nd, [5 5], OVG*Nd, Nd);
weighteddata = dx.*wx;
x = nufft_adj(weighteddata,st)/sqrt(Nd(1)*Nd(2));

end
```

Figura 23: Função utilizada para reconstrução por meio da *nufft*. A criação de calcporLoop2 foi importante devido às restrições inerentes à função *parfor* (o código não pôde ser diretamente implementado na função principal).

Utilizando processamento paralelo no *quad core*, foi observado um tempo de 1,02 segundos para a *nufft*. Utilizando o processamento paralelo no *dual core*, o tempo de execução da *nufft* foi de 1,61 segundos. Sem o processamento paralelo, o tempo observado no computador com *dual core* foi de 0,84 segundo. O tempo observado no computador com *quad core* sem o processamento paralelo foi de 0,68 segundo. Esse resultado, aparentemente contraditório, é justificado pelo fato de que o *parfor* exige um tempo mínimo maior do que o *for* para sua inicialização, como já foi mencionado antes. Portanto, para que o uso do processamento paralelo compense, para a *nufft* deve ser utilizado um volume maior de dados. A seguir, na figura 24, o código que foi utilizado para o processamento da *nufft* de forma direta, sem a preocupação com o uso paralelo de multiprocessadores.

```
clear all, close all;
load rt_spiral.mat
Nim=128;
d=d(:);
k=k(:);
w=w(:);
N=length(d);
%---------------------------------------
%PARA A OPÇÃO INUFFT, DESCOMENTAR:
        cd fessler
        setup
        cd ..
        om(:,1) = 2*pi*real(k(:)); %kspace trajectory
        om(:,2) = 2*pi*imag(k(:));
        Nd=[Nim, Nim];
        ovg=4; %OVG = overgrid factor
%---------------------------------------

disp('-------------------------------------------------');
disp('Tempo de execução sem o parfor:');
pause(.1)
tic
st = nufft_init(om, 2*Nd, [5 5], ovg*Nd, Nd);
weighteddata = d.*w;
m = nufft_adj(weighteddata,st)/sqrt(Nd(1)*Nd(2));
toc
pause(.1)
```

Figura 24: Código desenvolvido para a reconstrução dos dados por *nufft* sem a preocupação com o uso paralelo de multiprocessadores.

Para processamento paralelo utilizando *gridding* com *kernel* piramidal, o tempo de execução é de 0,47 segundo com o *dual core*, e de 0,38 segundo com o *quad core*. Sem o processamento paralelo, o tempo é de 0,25 segundo com a máquina de *dual core* e 0,17 segundo com a de *quad core*. A justificativa é a mesma do caso da *nufft* citado acima, ou seja, o tempo é curto demais e o processamento paralelo com *parfor* nesse caso não compensa.

```
%Rosana R Lima 2011
%Universidade de Brasília

clear all, close all;
load rt_spiral.mat
Nim=128;
d=d(:);
k=k(:);
w=w(:);
N=length(d);
disp('-------------------------------------------------');
disp('Tempo de execução sem o parfor:');
pause(.1)
tic
m=grid1weightsOVS2deapod(d,k,Nim,w);
toc
pause(.1)
```

Figura 25: Código desenvolvido para a reconstrução dos dados por *gridding* com deapodização sem a preocupação com o uso paralelo de multiprocessadores.

Deve-se salientar a importância dos testes serem realizados evitando sempre que possível a execução de outros programas pesados (que consumam muito do processador) simultaneamente.

A seguir imagens ilustrando os núcleos do processador *quad core* em operação paralelizada ou não-paralelizada.

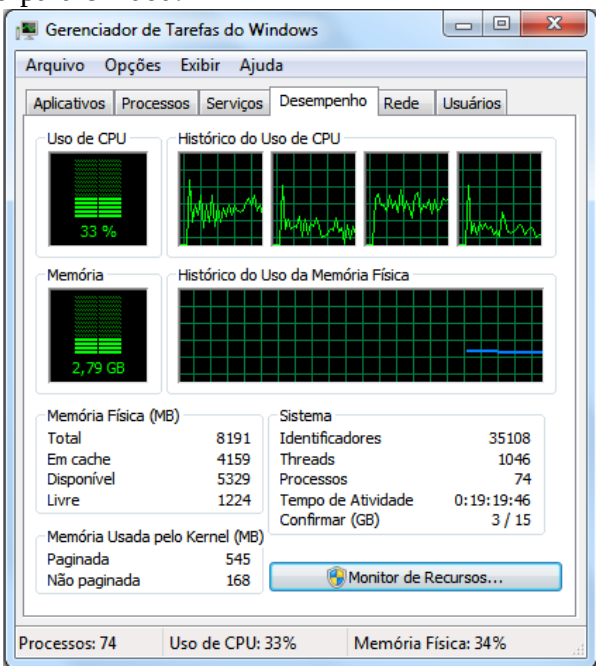

Figura 26: Desempenho do processador *quad core* utilizado de maneira não-paralelizada para reconstrução (nota-se o uso da CPU em 33%).

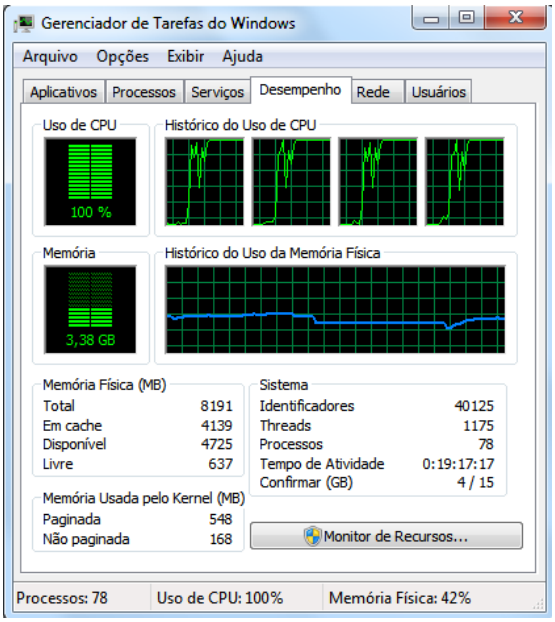

Figura 27: Algum tempo depois, desempenho do mesmo processador *quad core* utilizado de maneira paralelizada para a reconstrução (nota-se o uso da CPU máximo, em 100%).

E. Aplicação da reconstrução com processadores multi-núcleo em um grande volume de dados

A próxima etapa da pesquisa foi aplicar a reconstrução paralela a um volume grande de dados. A pesquisa foi realizada (e ainda está em andamento, à procura de melhorias nos resultados alcançados e soluções para os casos que ainda não foram resolvidos) para diversos casos: a reconstrução de um *voxel*; a de uma imagem obtida em uma bobina; de uma fatia do corpo (combinação do canal de várias imagens de diferentes bobinas); e de todo o volume.

Aqui, será comentado um dos casos com a utilização do *quad core*, para a reconstrução de dados de uma única fatia em uma posição z determinada (contendo, para cada *voxel*, um gráfico da velocidade ao longo de um tempo). A função de reconstrução utilizada foi a *nufft*. O tempo obtido sem o uso do processamento paralelo foi de cerca de 93,2 segundos, e o tempo com os múltiplos núcleos paralelamente ativos foi de 51,9 segundos e de 47,3 segundos, para duas formas ligeiramente diferentes de código. Houve uma redução de aproximadamente 49% no tempo de processamento, portanto, que antes não tinha aparecido para a função *nufft*, portanto um resultado significativo. Já com uma máquina de processador *dual core*, o tempo de processamento sem utilizar o paralelismo foi de 170,1 segundos; utilizando o multiprocessamento, foi de 111,1, para a mesma atividade. A redução foi, portanto, de 35%. Para otimizar este resultado, pode-se, por exemplo, buscar outras formas de processamento paralelo além do *parfor* (que apesar de ser muito eficiente, só funciona dentro do laço, e sob muitas condições).

F. Aplicação da reconstrução com processadores multi-núcleo utilizando a técnica de UNFOLD

Outra atividade relevante realizada foi a reconstrução de dados de vídeo de um *phantom* numérico sob o método de UNFOLD e de *view sharing*. Utilizou-se o processamento paralelo para o primeiro caso, e viu-se que tal abordagem é possível. O resultado, entretanto, não foi um tempo menor para o caso do processamento paralelo, porque o volume de dados era pequeno e o tempo de execução do código normal já era pouco. Como o *parfor* demanda um tempo para sua inicialização, espera-se que, caso o volume de dados fosse maior, o uso de reconstrução paralelizada seria mais vantajoso.

IV. Conclusão

Muitas técnicas de processamento tanto de imagens genéricas quanto de imagens de ressonância magnética foram aprendidas e desenvolvidas ao longo da pesquisa. Métodos relacionados à reconstrução de sinais de RM foram estudados, e adquiriu-se maior familiaridade com o software MATLAB ao longo do processo. O foco do trabalho, que é o processamento paralelo com multiprocessadores, mostrou resultados bastante satisfatórios, apesar das limitações vistas. Com a manutenção da pesquisa na área, alternativas de implementação devem surgir, e a redução de tempo deve ser ainda mais significativa. Os recursos cada vez mais avançados do software associado ao estudo mais aprofundado de novas técnicas devem ser responsáveis por significativas melhorias concernentes ao tema deste trabalho.

V. Referências

[1] R. H. Hashemi, W. G. Bradley, Jr and C. J. Lisanti, *MRI The Basics*, Lippincott Williams & Wilkins, 2004.

[2] M. A. Bernstein, K. F. King and X. J. Zhou, *Handbook of MRI Pulse Sequences*, Elsevier Academic Press, 2004.

[3] A. V. Oppenhein, R. W. Schafer, *Discrete-Time Signal Processing*, Prentice Hall 2nd edition, 1999

[4] R. C. Gonzalez, R. E. Woods and S. L. Eddins, *Digital Image Processing using MATLAB*, Gatesmark Publishing 2nd Edition, 2009

[5] J. L. A. Carvalho, *Velocity-encoded magnetic resonance imaging: acquisition, reconstruction and applications*, Ph. D. Thesis, 2008

[6] J. Pauly, *Non-Cartesian Reconstruction*, 2007

[7] B. Madore, G. H. Glover and N. J. Pelc, *Unaliasing by Fourier-Encoding the Overlaps Using the Temporal Dimension (UNFOLD), Applied to Cardiac Imaging and fMRI*, Magnetic Resonance in Medicine 42:813-828 (1999)

[8] J. Tsao, *On the UNFOLD Method*, Magnetic Resonance in Medicine 47:202-207 (2002)

[9] S. H. Nawab, A. V. Oppenheim and A. S. Willsky, *Sinais e Sistemas*, Pearson 2a Edição, 2010