UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ALEXSSANDRO FERREIRA CORDEIRO

AVALIAÇÃO DE DESEMPENHO NA RECONSTRUÇÃO DE IMAGENS 2D DE TOMOGRAFIA COMPUTADORIZADA UTILIZANDO PROGRAMAÇÃO MASSIVAMENTE PARALELA CUDA

TRABALHO DE CONCLUSÃO DE CURSO

MEDIANEIRA

2017

ALEXSSANDRO FERREIRA CORDEIRO

AVALIAÇÃO DE DESEMPENHO NA RECONSTRUÇÃO DE IMAGENS 2D DE TOMOGRAFIA COMPUTADORIZADA UTILIZANDO PROGRAMAÇÃO MASSIVAMENTE PARALELA CUDA

Trabalho de Conclusão de Curso apresentado ao Departamento Acadêmico de Computação da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de "Bacharel em Computação".

Orientador: Prof. Ms. Hamilton Pereira da Silva

MEDIANEIRA

2017

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Coordenação do Curso de Ciência da Computação

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

TERMO DE APROVAÇÃO

AVALIAÇÃO DE DESEMPENHO NA RECONSTRUÇÃO DE IMAGENS 2D DE TOMOGRAFIA COMPUTADORIZADA UTILIZANDO PROGRAMAÇÃO MASSIVAMENTE PARALELA CUDA

Por

ALEXSSANDRO FERREIRA CORDEIRO

Este Trabalho de Conclusão de Curso foi apresentado às 14:40h do dia 06 de Junho de 2017 como requisito parcial para a obtenção do título de Bacharel no Curso de Ciência da Computação, da Universidade Tecnológica Federal do Paraná, Câmpus Medianeira. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. Ms. Hamilton Pereira da Silva
UTFPR - Câmpus Medianeira

Prof. Dr. Evando Carlos Pessini
UTFPR - Câmpus Medianeira

Prof. Dr. Everton Coimbra de Araujo
UTFPR - Câmpus Medianeira

A folha de aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

CORDEIRO, Alexssandro Ferreira. AVALIAÇÃO DE DESEMPENHO NA RECONSTRUÇÃO DE IMAGENS 2D DE TOMOGRAFIA COMPUTADORIZADA UTILIZANDO PROGRAMAÇÃO MASSIVAMENTE PARALELA CUDA. 49 f. Trabalho de Conclusão de Curso – Curso de Ciência da Computação, Universidade Tecnológica Federal do Paraná. Medianeira, 2017.

O presente estudo apresenta uma análise e avaliação do desempenho na reconstrução de imagens 2D de tomografia computadorizada, utilizando uma abordagem massivamente paralela em uma GPU aplicando a tecnologia CUDA, comparando com uma abordagem paralela e sequencial em uma CPU convencional. Foi avaliada a qualidade das imagens geradas utilizando as métricas de avaliação relação sinal ruído de pico (PSNR), método utilizado para verificar as diferenças dos pixels por meio do erro quadrático médio (MSE); E pelo método do índice da semelhança estrutural (SSIM), utilizado para verificar a semelhança das imagens a partir da perda de luminância, correlação, distorção e distorção de contraste. Também foram analisadas as reconstruções com os tipos de dados *Float* 32 bits e *Double* 64 bits, a fim de validar o desempenho e a qualidade das imagens geradas com o aumento das casas decimais. As reconstruções das imagens de tomografia computadorizada 2D foram realizadas por meio do algoritmo *Filtered Back Projection* (FBP) ou Retroprojeção filtrada, utilizando a transformada de Radon e o filtro de convolução para retirada de ruídos da imagem. Os resultados apontam que a GPU no tipo de dados *Float* 32 bits , tem o melhor desempenho dentre as abordagens utilizadas. Em *Double* 64 bits a abordagem da CPU paralela obteve um melhor desempenho em comparação a GPU, porém a GPU manteve-se com valores próximos a abordagem da CPU paralela.

Palavras-chave: Computação de alto desempenho, Computação gráfica, Processamento de imagens

ABSTRACT

CORDEIRO, Alexssandro Ferreira. . 49 f. Trabalho de Conclusão de Curso – Curso de Ciência da Computação, Universidade Tecnológica Federal do Paraná. Medianeira, 2017.

The present study presents an analysis and evaluation of the performance in the reconstruction of 2D images of computed tomography, using a massively parallel approach in a GPU applying CUDA technology, comparing with a parallel and sequential approach in a conventional CPU. The quality of the generated images was evaluated using the Peak Noise Ratio (PSNR) evaluation metrics, method used to verify the pixel differences through the mean square error (MSE); And by the structural similarity index (SSIM) method, used to verify the similarity of the images from loss of luminance, correlation, distortion and contrast distortion. We also analyzed the reconstructions with the Float 32 bits and Double 64 bits data types to validate the performance and quality of the generated images with the increase of the decimal places. Reconstructions of 2D computed tomography images were performed using the Filtered Back Projection (FBP) algorithm, using the Radon transform and convolution filter to remove image noise. The results indicate that the GPU in the Float 32 bits data type has the best performance among the approaches used. In Double 64 bits the parallel CPU approach performed better in comparison to GPU, but the GPU remained close to parallel CPU approach.

Keywords: High performance computing, Computer graphics, Image processing

Aos meus familiares, professores e colegas que me apoiaram durante esta jornada.

AGRADECIMENTOS

Agradeço primeiramente Deus, pela vida, que torna tudo isso possível. Agradeço aos meus familiares pelo incentivo e dedicação. Agradeço aos professores que acreditaram e me apoiaram, principalmente meu orientador pela paciência, apoio e confiança em meu sucesso nesse trabalho. Agradeço aos meus colegas e a todas as pessoas que participaram dessa etapa tão importante da minha vida.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

CPU	Central Processor Unit
CUDA	Compute Unified Device Architecture
dB	Decibéis
DICOM	Digital Imaging and Communications in Medicene
GB	Gigabytes
GDDR	Graphics Double Data Rate
GHz	Giga-Hertz
GPU	Graphic Processor Unit
GUI	Graphical User Interface
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
MVC	Model View Controller
PSNR	Peak Signal Noise Ratio
SM	Multiprocessadores de Streaming
SP	Processadores de Streaming
SSIM	Structural Similarity Index

SUMÁRIO

1 INTRODUÇÃO

Com o aumento da demanda de processamento e a limitação do *clock* dos processadores, adotou-se a tecnologia de vários núcleos para cada processador. Desta maneira, o programador pode designar tarefas para vários processadores ao mesmo tempo, ganhando desempenho em suas aplicações.

A *Central Processor Unit* (CPU) vem ganhando bastante desempenho nas últimas décadas, porém o *clock* foi estagnado na faixa dos 4.0 *Giga-Hertz* (GHz), conforme pode ser observado nos últimos lançamentos de processadores como o Intel I7 extreme Edition. Adotando a tecnologia de múltiplos núcleos, conforme descrito, o processador não fica só dependente de seu *clock* mas também da quantidade de núcleos. Quanto maior o número de núcleos mais poder de processamento paralelo pode ser realizado simultaneamente na CPU, desta maneira gerando aumento no desempenho dos softwares.

Atualmente tem-se processadores como o Intel[1] I7 com 8 núcleos físicos, emulando mais 8 núcleos lógicos, podendo executar até 16 processos (*threads*) simultaneamente conforme o fabricante (INTEL, 2016), desta maneira possibilita o ganho de desempenho em softwares que se utilizam da programação paralela. Porém, dependendo do problema, como reconstruções de imagens 2D de tomografia computadorizada, essa quantidade de processamento ainda pode ser pouca, tornando o software lento.

Com a evolução dos jogos eletrônicos foi necessário ampliar o poder de processamento gráfico, pois esses jogos estão ficando cada vez mais realistas, exigindo muitos cálculos por segundo, fazendo com que as placas gráficas evoluíssem a um nível de processamento muito alto. Desta forma os pesquisadores verificaram que esse processamento poderia ser usado em paralelo com a CPU para agilizar as suas pesquisas, aumentando o desempenho dos aplicativos.

Hoje em dia uma placa de vídeo relativamente potente tem em média 1024 núcleos, como a GTX 960 da NVIDIA[2]. Eles não podem ser comparados aos núcleos da CPU, pois

[1] Site Intel: http://www.intel.com.br/content/www/br/pt/homepage.html
[2] Site Nvidia: http://www.nvidia.com.br/page/home.html

são para uso distintos, enquanto a CPU tem propósito geral em um computador, a *Graphic Processor Unit* (GPU), que é o processador das placas gráficas, são de propósito de cálculos numéricos, em concordância aos autores Kirk e Hwu (2011a), porem há outras fontes que discordam dos autores. O site da fabricante[3] das GPUs se refere a essa tecnologia como uma solução para computação de propósito geral, conforme NVIDIA (2016b). Neste trabalho a GPU será usada para fins de cálculos numéricos.

Com o avanço da tecnologia das placas de vídeo, surgiram as GPUs programáveis, que facilitaram o processo de programação para as placas de vídeo, desta forma os softwares podem executar de forma paralela com a GPU e CPU, originando-se a tecnologia *Compute Unified Device Architecture* (CUDA).

CUDA é uma tecnologia proprietária da NVIDIA. Com essa tecnologia os programadores podem fazer suas aplicações para trabalhar de forma paralela com a GPU e CPU, porém essa tecnologia tem suas limitações, os processadores CUDA são desenvolvidos especialmente para problemas de cálculos matemáticos e que sejam paralelizáveis.

Um desafio é paralelizar problemas, e para se ter ganhos de desempenho nessa tecnologia, o problema precisa ser dividido em várias partes pequenas e independentes, como cálculo de matrizes, onde cada posição da matriz é calculada de forma independente uma da outra.

Um problema paralelizável é a reconstrução de imagens obtidas a partir da tomografia computadorizada. Essas imagens são representadas por uma matriz, onde tem-se os valores das atenuações na qual o raio-x sofreu ao passar por um corpo em determinado angulo θ.

Este trabalho tem como objetivo reconstruir imagens obtidas por meio de tomografia computadorizada, tornando-a um problema paralelizável para se utilizar da tecnologia CUDA, visando comparar com o desempenho da execução paralela e sequencial em uma CPU convencional.

[3] http://www.nvidia.com.br/page/home.html

1.1 DELIMITAÇÃO DO TEMA

Em busca de um melhor desempenho para os softwares, a utilização de uma computação acelerada por uma GPU pode contribuir para alcançar esse objetivo.

No escopo do estudo proposto serão analisados dados de um tomógrafo, a fim de reconstruir imagens de tomografia computadorizada 2D, verificando se uma computação acelerada por GPU obtém melhor desempenho em comparação a uma computação sequencial ou paralela em uma CPU convencional.

1.2 PROBLEMAS E PREMISSAS

Os principais problemas encontrados que objetivaram o estudo foram:

- Aceleração no desempenho de softwares;
- Falta de informações com componentes atualizados;
- Análise do desempenho de uma computação acelerada por uma GPU em comparação a uma CPU convencional.

Em decorrência dos problemas expostos acima, faz-se necessário verificar a possibilidade de ganho no desempenho, com a utilização da computação acelerada por uma GPU, bem como quantificar esse ganho utilizando componentes atuais, usando a placa de vídeo GTX 960 da NVIDIA com 1024 núcleos e o processador AMD[4] FX-8350 com 8 núcleos e 4.0Ghz.

[4]http://www.amd.com/pt

1.3 OBJETIVOS

Neste tópico são apresentados os objetivos gerais e específicos que se pretende atingir durante a pesquisa.

1.3.1 Gerais

Efetuar uma análise e avaliação no desempenho das reconstruções de imagens 2D de tomografia computadorizada, utilizando uma abordagem massivamente paralela, por meio da tecnologia CUDA, comparando com uma abordagem paralela e sequencial em uma CPU convencional; Serão usados dados do tipo *float* 32 bits e *double* 64 bits, a fim de verificar o comportamento do experimento com o aumento da precisão, também será analisada a qualidade das imagens geradas em cada abordagem, desta maneira será possível verificar a integridade das imagens, pois são imagens de uso da medicina, onde a qualidade se faz necessário para detecção de anomalias.

1.3.2 Específicos

- Estudar o funcionamento da tomografia computadorizada;
- Estudar a utilização da tecnologia CUDA com a linguagem C#, por meio da IDE do Visual Studio;
- Utilização de processamento paralelo com GPU da NVIDIA para reconstrução de imagens 2D de tomografia computadorizada, com algoritmo otimizado;
- Implementar a reconstrução de imagens 2D obtidas por meio de tomografias computadorizadas, utilizando a linguagem C#;
- Desenvolver um aplicativo que utilize computação paralela para aplicações em placas de

vídeo NVIDIA com múltiplos núcleos, para reconstrução de imagens 2D de tomografia computadorizada.

1.4 JUSTIFICATIVA

Com a crescente demanda de processamento utilizada pelos *softwares* e a diminuição dos avanços dos processadores devido aos problemas de aquecimento e consumo de energia, que limitou a frequência do *clock* dos processadores, conforme Kirk e Hwu (2011b), as indústrias de semicondutores estabeleceram duas trajetórias, a de múltiplos núcleos (*multicore*), e a baseada em muitos núcleos (*many-core*).

A trajetória *multicore* visa aumentar os núcleos das CPUs, começando com 2 núcleos e dobrando a cada geração. Por outro lado a trajetória *many-core* surgiu com vários núcleos menores para uso de computação numérica e dobrando a cada geração, chegando ao ponto de milhares de núcleos em uma GPU, contra alguns de uma CPU, conforme NVIDIA (2016c).

Como solução para computação de alto desempenho, entre outras, existe a computação massivamente paralela com os milhares de núcleos da GPU, desta forma é possível retirar da CPU uma carga significativa de processamento redirecionando para a GPU.

Com este trabalho é possível constatar a importância de uma programação paralela com a GPU para reconstrução de imagens e outros problemas paralelos para obter-se um melhor desempenho, sem a necessidade de distribuir o processamento entre outros computadores na rede, como o uso de *cluster*.

1.5 ORGANIZAÇÃO DO DOCUMENTO

A estrutura deste trabalho está organizada da seguinte forma. O capítulo Fundamentação Teórica apresenta conceitos sobre CUDA, bem como sua forma de controle

das *threads*, histórico da tomografia computadorizada e definições do raio-x para obtenção da imagem. O capítulo Materiais e Métodos apresenta como o estudo foi desenvolvido, bem como as ferramentas utilizadas. O capítulo Desenvolvimento demonstra como foi desenvolvido o aplicativo para reconstrução das imagens 2D. O capítulo Resultados apresenta os resultados obtidos por meio de sucessivas reconstruções das imagens 2D. O capítulo Conclusão apresenta a análise dos resultados obtidos.

2 FUNDAMENTAÇÃO TEÓRICA

A seguir são apresentados os conceitos importantes para o desenvolvimento da pesquisa, como a tomografia computadorizada, com suas etapas de obtenção da imagem e a linguagem CUDA, com sua arquitetura e funcionamento das *threads*.

2.1 CUDA

CUDA é uma extensão da linguagem C, tecnologia proprietária da NVIDIA, que possibilita a utilização da GPU das placas gráficas para trabalharem em conjunto com a CPU em uma computação massivamente paralela, conforme NVIDIA (2016c).

Em CUDA é possível programar em várias linguagens, como C, C++, C#, Fortran, Java e Python, entre outras, por meio de bibliotecas disponibilizadas pela NVIDIA, conforme NVIDIA (2016a);

Tendo em vista a compatibilidade com outras linguagens e os milhares de *cores* da GPU em comparação a uma CPU, conforme ilustra a Figura 1, CUDA torna-se atraente por apresentar um poder computacional muito elevado e compatibilidade com várias linguagens mundialmente utilizadas.

A ideia dessa tecnologia é transferir para a placa gráfica problemas que possam ser paralelizáveis, de forma a utilizar o maior número de processadores simultaneamente das GPUs, desta maneira, obtendo um melhor desempenho em suas aplicações, conforme Kirk e Hwu (2011c).

Para transferir processamento da CPU para a GPU em CUDA, é necessário fazer chamadas de *kernel* utilizando a *TAG _global_* para que esse código seja executado na GPU.

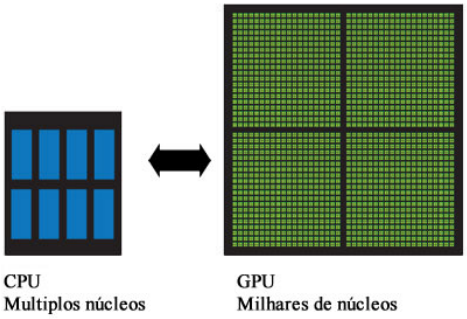

Figura 1 – Comparação de uma CPU vs GPU em quantidade de núcleos.
Fonte: http://www.nvidia.com/object/what-is-gpu-computing.html (Adaptado)

2.1.1 Arquitetura CUDA

A arquitetura CUDA, conforme ilustrada na Figura 2, organiza seus processadores por meio de blocos chamados Multiprocessadores de *Streaming* (SM) altamente encadeados, os SM são compostos por Processadores de *Streaming* (SP) e a quantidade de SP pode alterar em cada SM de acordo com a geração da GPU.

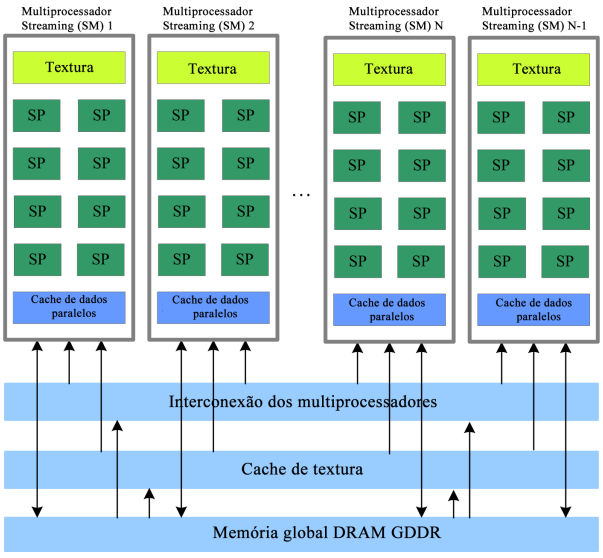

Figura 2 – Ilustração da arquitetura de uma GPU CUDA.
Fonte: (LIU et al., 2009) (Adaptado)

Atualmente encontra-se GPUs com até 8 *Gigabytes* (GB) de memória DRAM (GDDR) (Graphics Double Data Rate), conforme ilustrado na Figura 2, definida como Memória global. Elas diferem da DRAM da placa mãe, pois são para propósitos gráficos, mais especificamente

memória de *frame buffer*, mas em CUDA ela funciona como uma memória com largura de banda muito grande, porém com uma latência maior que a DRAM da placa mãe.

Cada SP tem o conceito de *threaded*, ou seja, pode executar milhares de *threads* por aplicação; Um exemplo é a placa gráfica G80, que possui 128 SP (16 SM, com 8 SP cada) podendo executar até 768 *threads* por SM, chegando a um total de 12288 *threads* para a placa G80 conforme Kirk e Hwu (2011c).

2.1.2 Threads em CUDA

A Figura 3 ilustra o funcionamento de *threads* em CUDA. Em uma GPU CUDA as *threads* são organizadas em *GRID* (grades) e *BLOCK* (blocos), sendo possível distinguir cada *thread*, pois elas têm coordenadas exclusivas.

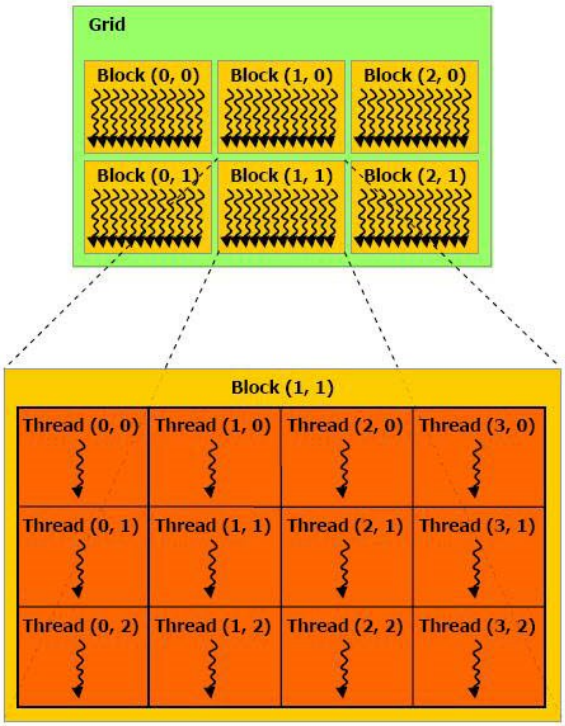

Figura 3 – Ilustração de *threads* em uma GPU CUDA.
Fonte: http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz4M2HixnHR

As variáveis de referências sobre as coordenadas de uma *thread* são *blockIdx* e

threadIdx. As dimensões dos blocos e grades são *gridDim* e *blockDim*.

Vale ressaltar que *threads* de mesmo bloco compartilham dados por meio de uma memória compartilhada e com pouca latência, porém *threads* de blocos diferentes não cooperaram entre si.

2.2 TOMOGRAFIA COMPUTADORIZADA

Com a descoberta do raio-x e a possibilidade de efetuar exames não invasivos nos pacientes, surgiu a tomografia computadorizada. Pode-se verificar na história da tomografia computadorizada, que será apresentada a seguir, que o avanço da tomografia não é mérito somente de um pesquisador, foram anos de pesquisas e avanços até se chegar nos dias de hoje, como uma tecnologia que permite ao usuário menos exposição ao raio-x e maior precisão para o médico ao efetuar sua avaliação.

Os tópicos a seguir estão em conformidade com (NERSISSIAN, 2016; NACIF; SANTOS, 2009; KAK; SLANEY, 1987).

2.2.1 Histórico

Com a descoberta do raio x pelo doutor Wilhelm Conrad Roentgen em 1895, ocorreram muitos estudos sobre formas de obter imagens a partir de radiação, não sendo necessário exames invasivos nos pacientes, surgindo as radiografias.

Em 1917 Johann Radon desenvolveu um algoritmo matemático capaz de reconstruir imagens de seções ou fatias de um corpo por meio de medições das atenuações do raio-x.

Em 1960 a tomografia computadorizada começa a andar de forma lenta, pois havia falta de apoio matemático, necessitando de uma extensa análise para obtenção da imagem.

Em 1962 o físico e matemático Allan Cormack fez uma contribuição fundamental para

a reconstrução de imagens, construindo um algoritmo que estuda a atenuação do raio-x em seu trajeto pro meio de um segmento de corpo.

Em 1972 o engenheiro elétrico Godfrey Hounsfield fez a primeira imagem de diagnóstico, pois Hounsfield acreditava que os raio-x poderiam conter mais informações do que as quais era possível capturar com o filme; desta forma, pensou que um computador pudesse ajudá-lo a obtê-las. Com isso, o primeiro diagnóstico de Hounsfield foi em um paciente selecionado pelo Dr. James Ambrose, com suspeita de tumor no cérebro. O diagnóstico foi animador, pois a imagem reconstruída mostrava a área lesionada. Após os anos de 1972 Hounsfield e Cormack chamaram esse novo método de obtenção de imagens com raio-x de tomografia computadorizada.

O primeiro protótipo de Hounsfield foi utilizando raios gama a partir de amerício-241, demorando 9 dias para obter a imagem de um crânio e mais 150 horas para o computador reconstruir. Após o explicitado anteriormente Hounsfield efetuou outro teste, agora com raio-x reduzindo o tempo de obtenção da imagem para 9 horas.

2.2.2 Gerações

A primeira geração de tomografia computadorizada se baseia no aparelho que Hounsfield desenvolveu em 1972, onde se tem uma ou duas fontes de raio-x e um ou dois detectores conforme mostra a Figura 4, na qual efetuam uma translação de 180 graus no paciente para obter um escaneamento completo do corpo. Esta geração de tomografia tem um feixe muito colimado, ou seja, direcional em forma de “lápis fino”, onde vai direto para um detector , que recebe o feixe de raio x e analisa a atenuação, conforme (NACIF; SANTOS, 2009).

O procedimento de escaneamento é feito da seguinte forma:

- A fonte de raio-x emite um feixe que será recebido pelo detector;
- A fonte de raio-x e o detector efetuam uma rotação de 1 grau;
- A fonte de raio-x e o detector efetuam translação para coletar dados de outra área;
- Esses procedimentos se repetem até completar os 180 graus de translação.

A Figura 5 representa um sistema de segunda geração, onde o número de detectores aumentou entre 5 a 50 unidades. Desta forma, a fonte de raio-x agora utiliza um feixe divergente, em forma de leque, onde para cada fonte de raio-x tem-se vários detectores.

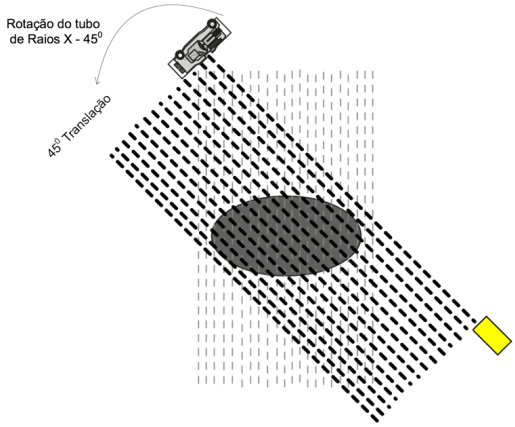

Figura 4 – Tomografia computadorizada de primeira geração.
Fonte: (NERSISSIAN, 2016)

Considerando isso, o tempo da aquisição das projeções ficou em torno de 20 segundos em média.

A translação continuou com 180 graus em torno do paciente, porém a rotação das fontes e detectores aumentou para 6 graus a cada passo, assim alterando o algoritmo de reconstrução da imagem, que se baseia no espaçamento dos ângulos entre os feixes.

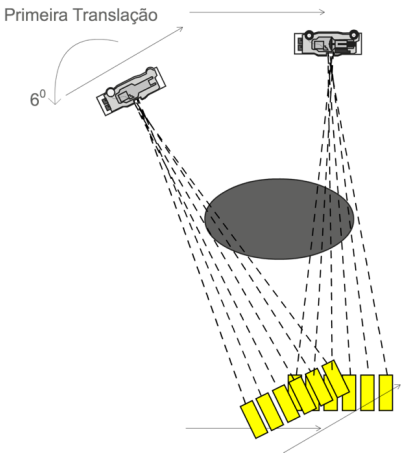

Figura 5 – Tomografia computadorizada de segunda geração.
Fonte: (NERSISSIAN, 2016)

A Figura 6 ilustra o funcionamento da terceira geração. Observa-se aumento no número de detectores, podendo-se utilizar até 520, conforme (NACIF; SANTOS, 2009).

Nessa geração o tempo da aquisição das projeções é em média, menor que 1 segundo.

A rotação conjunta da fonte de raio-x com os detectores é de 360 graus em torno do

paciente.

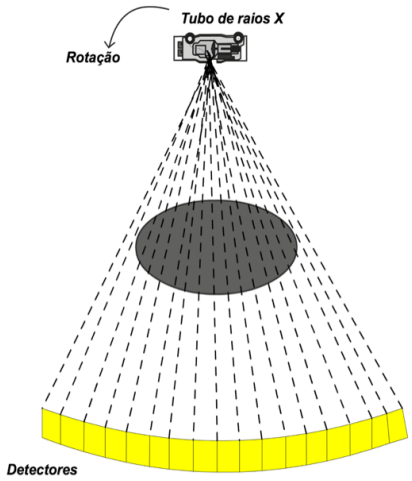

Figura 6 – Tomografia computadorizada de terceira geração.
Fonte: (NERSISSIAN, 2016)

A Figura 7 ilustra o funcionamento da quarta geração. Observa-se novamente aumento no número de detectores, contendo em média 4000 unidades que se mantém fixos. Deste modo a fonte de raio-x translada o paciente e os detectores fixos recebem os raios-x após passar pelo paciente.

A rotação da fonte de raio x é de 360 graus em torno do paciente.

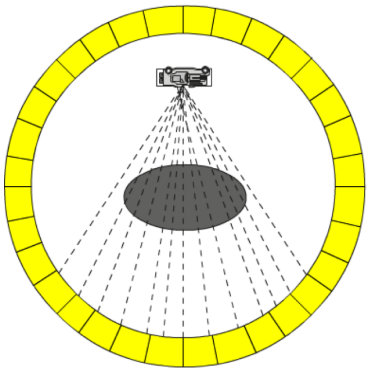

Figura 7 – Tomografia computadorizada de quarta geração.
Fonte: (NERSISSIAN, 2016)

A Figura 8 ilustra o funcionamento dos sistemas helicoidal ou espiral. Verifica-se características semelhantes aos da quarta geração, porém, agora, a mesa do paciente translada, enquanto a fonte de raio-x e os detectores o circulam.

A rotação da fonte de raio-x é de 360 graus em torno do paciente.

Com esse sistema foi possível obter imagens volumétricas.

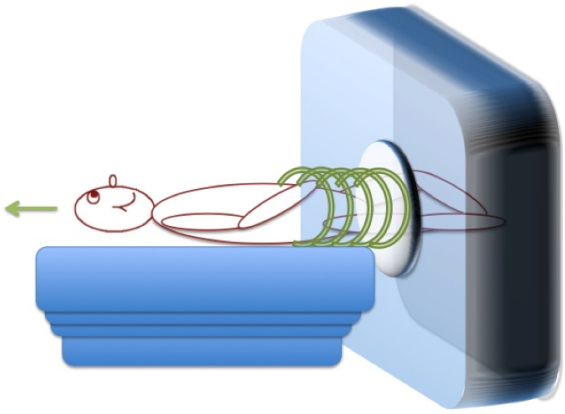

Figura 8 – Tomografia computadorizada helicoidal.

Fonte: (NERSISSIAN, 2016)

Nota-se um avanço nos sistemas de tomografia, que proporciona ao paciente exames não invasivos e menores doses de raio-x, com imagens tridimensionais, devido à forma como o sistema helicoidal adquire os dados, em forma de espiral.

O princípio da tomografia referente a aquisição dos dados continua o mesmo, e serão discutidos na seção de fundamentos da tomografia.

2.2.3 Componentes de um equipamento de tomografia computadorizada

Nesta seção é apresentado de forma, sucinta, um aparelho de tomografia computadorizada da geração helicoidal, pois são os mais utilizados atualmente. A Figura 9 representa um tomógrafo helicoidal.

Um aparelho de tomografia helicoidal é composto por *Gantry*, mesa e computador conforme pode ser observado em (NERSISSIAN, 2016; ANVISA, 2016).

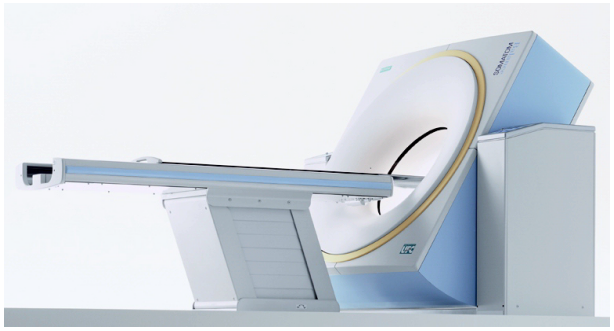

Figura 9 – Equipamento de tomografia computadorizada Siemens SOMATOM Balance.
Fonte: (ANVISA, 2016)

O *Gantry* pode ser ilustrado conforme a Figura10 . Na Figura 11 pode-se perceber um melhor detalhamento do interior do equipamento, onde encontra-se a fonte de raio-x juntamente com os detectores de atenuação e o colimador; Sua função é lançar feixes de raio-x por meio da fonte, onde se tem os colimadores logo a frente, para regular as doses de raio-x no paciente e melhorar a qualidade da imagem, conforme (NERSISSIAN, 2016), e os detectores na outra extremidade para coletar as atenuações que o raio-x sofreu ao atravessar o corpo.

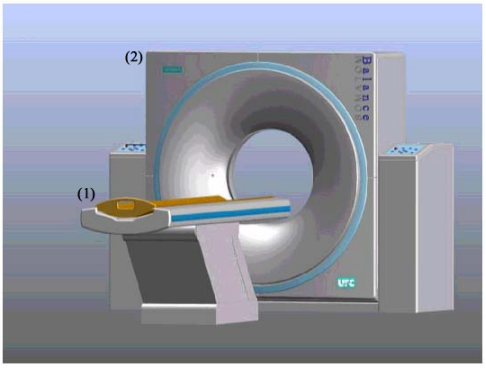

Figura 10 – Componentes de uma tomografo. (1)Mesa (2)Gantry
Fonte: (ANVISA, 2016)

A mesa do paciente é ilustrada na Figura 10. Ela tem como função acomodar o paciente e transladá-lo internamente no *Gantry*, para que os feixes de raio-x façam a varredura no corpo.

O computador tem como objetivo receber os dados do tomógrafo e analisá-los a fim de reconstruir a imagem.

Esses são os principais componentes de um equipamento de tomografia computadorizada helicoidal.

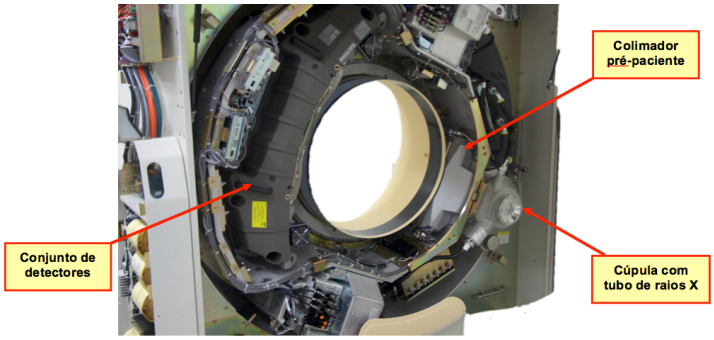

Figura 11 – Demonstração interna do Gantry.

Fonte: (NERSISSIAN, 2016)

2.2.4 Fundamentos da tomografia e transformada de Radon

Conforme visto anteriormente, o sistema de tomografia pode ser realizado por meio de amostras dos feixes de raio-x, ou gama, em diferentes ângulos. A ideia principal deste sistema é medir a atenuação que feixes sofrem ao atravessar um corpo, conforme mostra a Figura 12, e para isso é utilizada a transformada de Radon, que são denominadas Sinogramas.

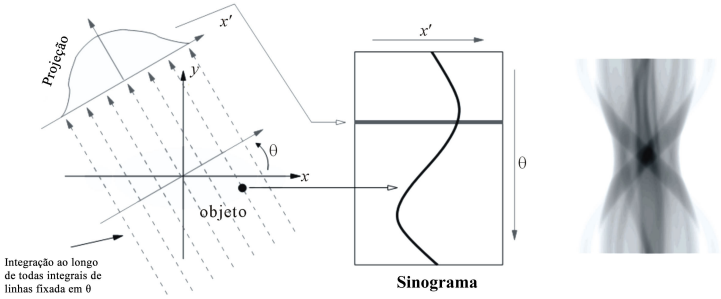

Figura 12 – Ilustração de projeção e sinograma.

Fonte: (ASL; SADREMOMTAZ, 2013) (Adaptado)

A Figura 13 mostra algumas referências universais para exames clínicos.

Figura 13 – Ilustração de radio densidade.

Fonte: http://www.ufrgs.br/semiologiaortopedica/Modulo_21.pdf

Também é possível ver exemplos de atenuações universais para exames clínicos, pela Tabela 1 de atenuação de Hounsfield (HU).

Tecido	Unidades em (HU)
Osso denso	1000
Músculo	50
Matéria branca	45
Matéria cinzenta	40
Sangue	20
Fluido Cerebroespinhal	15
Água	0
Gordura	-100
Pulmão	-200
Ar	-1000

Tabela 1 – Valores em Hounsfield (HU) para substâncias comuns em exames clínicos.

A lei que estuda a interação de um feixe colimado de raio-x, ou gama, com a matéria, é a lei de LAMBERT-BEER conforme a equação 1 :

$$I = I_0 \cdot e^{-\mu \cdot d} \tag{1}$$

Onde I é a intensidade do feixe após passar pelo corpo; I_0 é a intensidade inicial do feixe; e $\mu \cdot$ d, o produto do coeficiente de atenuação com a espessura do corpo incidido pelo feixe, conforme ilustra a Figura 14.

Com isso, ao efetuar uma tomografia, tem-se uma matriz com os valores das atenuações dos feixes que passaram pelo corpo e foram recebidos pelos detectores.

Nesse caso, as imagens são representadas em um espaço bidimensional $f(x,y)$, representando cada fatia ou seção transversal de um objeto.

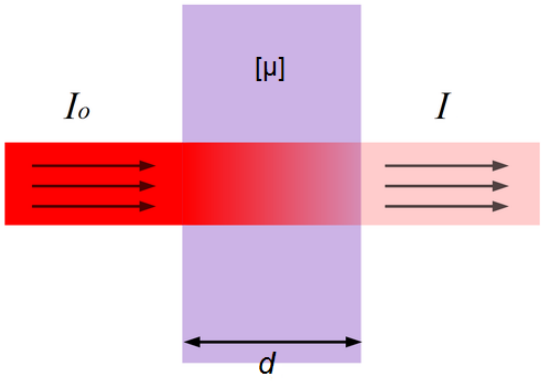

Figura 14 – Ilustração de lei de Lambert-Beer.
Fonte: (CALIFORNIA, 2016) (Adaptado)

No R^3, um corpo tridimensional pode ser obtido por meio de uma pilha com seções transversais bidimensionais, conforme Kak e Slaney (1987).

Como o raio-x viaja em linha reta, uma forma de medir sua atenuação é por meio da integral de linha. Será utilizado o sistema de coordenadas definido na Figura 15 para descrever as integrais de linhas e projeções.

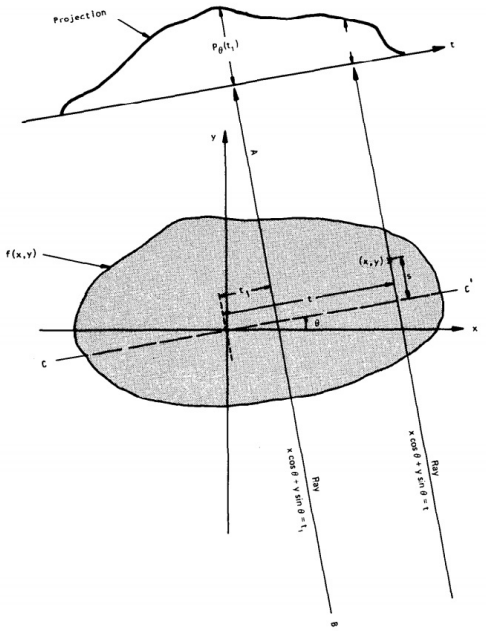

Figura 15 – Projeção Unidimensional.
Fonte: (KAK; SLANEY, 1987)

Observe que cada objeto é representado por uma função f(x,y) e cada linha integrada

pelos parâmetros (θ, t).

Também observa-se que a equação de cada linha ou feixe de raio x é $(xcos\theta + ycos\theta = t)$.

Assim pode-se usar essa relação para definir a integral de linha Pθ(t) como:

$$p\theta(t) = \int_{linha(\theta,t)} f(x,y)ds \tag{2}$$

Utilizando a função delta de Dirac conforme Solomon e Breckon (2013), será garantido a singularidade e propriedade de amostragem, desta forma, os pontos tomados sobre a linha Pθ(t) são diferentes de zero, enquanto os demais são zero.

$$p\theta(t) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\delta(xcos\theta + ycos\theta - t)dxdy \tag{3}$$

onde Pθ(t) é chamada de transformada de Radon de f(x,y).

2.2.5 Retroprojeção filtrada

A retroprojeção filtrada tem como objetivo retirar imperfeições nas amostras, como ruídos e Efeito estrela mostrado na Figura 16.

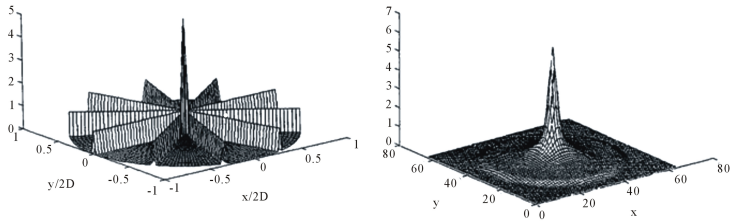

Figura 16 – Projetando uma superfície de imagem utilizando retroprojeção filtrada.
Fonte: (ASL; SADREMOMTAZ, 2013)

A transformada inversa de Radon é dada conforme 4:

$$f(x,y) = \frac{1}{2\pi}\int_0^{\pi}\int_{-\infty}^{\infty} \frac{1}{X\cdot\theta - l}\frac{\partial P_\theta(l)}{\partial l}\partial l\partial\theta \tag{4}$$

Segundo (HERMAN, 2009), existe singularidade em $(X \cdot \theta - l)$, pois este denominador pode chegar a zero. Usando o valor principal de Cauchy e assumindo que a equação 5 existe e é continua:

$$\frac{\partial P_\theta(l)}{\partial l} \tag{5}$$

Pode-se reescrever a equação 4 da seguinte forma:

$$f(x,y) = \frac{1}{2\pi} \int_0^\pi \lim_{\xi \to 0} \int_{-\infty}^{\infty} P_\theta(l) H\xi(X \cdot \theta - l) dl d\theta \tag{6}$$

Assim contornando a singularidade. Porém, ainda tem-se o problema das atenuações lineares altas, fora das regiões de fronteiras. Ocasionando efeito estrelado nas reconstruções de imagens conforme (REBELO, 1984). Para contornar essa situação será utilizado o filtro de convolução conforme as equações 7 e 8 (HERMAN, 2009; PARENTE, 1996; RAO; KRIZ, 2016).

$$H_\xi(l) = \frac{1}{\xi^2} \ \ para\ l < \xi \tag{7}$$

$$H_\xi(l) = \frac{-1}{l^2} \ \ para\ l >= \xi \tag{8}$$

O filtro de convolução conforme Solomon e Breckon (2013), é descrito como uma sequência de lentes em um sistema óptico de imagens, ou seja, uma sequência de processamento que atue sobre a entrada de dados lineares e invariantes sob translação é descrito por uma sequência de convoluções da entrada com as respectivas funções dos elementos.

A partir deste ponto, em geral, um caso real utiliza-se para a reconstrução da imagem a Equação 9.

$$f(x,y) \ = \ \sum_i^k Q\ \theta_i\ (l)[\ x\ cos(\theta_i)\ +\ y\ cos(\theta_i)] \tag{9}$$

A Equação 9 será utilizada no algoritmo desenvolvido em C# para a reconstrução da imagem.

3 MATERIAIS E MÉTODOS

Neste capítulo são apresentadas as ferramentas e os métodos para efetuar o comparativo "*BENCHMARK*" referente a reconstrução de imagens 2D de tomografia computadorizada, utilizando uma abordagem massivamente paralela com a utilização de uma GPU e a paralela em uma CPU.

3.1 LINGUAGEM DE PROGRAMAÇÃO

A linguagem adotada para o desenvolvimento do estudo foi a C#.

C# é uma linguagem de alto nível, com suporte a orientação a objetos, fortemente tipada e com suporte a *threads*.

Outro fator que ajudou na decisão da linguagem foi a disponibilidade de bibliotecas CUDA para a linguagem C#, assim possibilitando o *benchmark*.

3.2 VISUAL STUDIO 2013 COMMUNITY

Não foi utilizada a versão 2015, pois ocorreu incompatibilidade com o CUDA Toolkit 8. Por esse motivo foi utilizado a versão 2013 do Visual *Studio*.

O Visual *Studio* 2013 *community*[1] é um *Integrated Development Environment* (IDE) ou ambiente de desenvolvimento integrado, que visa facilitar, organizar e agilizar o processo de desenvolvimento de softwares.

Essa ferramenta tem suporte para desenvolvimento na linguagem C#, desta maneira, possibilitando seu uso no estudo. A ideia de utilizar essa ferramenta é justamente facilitar o processo de desenvolvimento, sendo possível dar maior ênfase na solução do problema.

3.3 HARDWARE E SISTEMA OPERACIONAL

Foi utilizado um computador com processador AMD FX-8350 com 8 núcleos e 4.0 Ghz, 8GB de memória RAM DDR3 kingston[2] HyperX Fury com 1600Mhz, um HD SSD kingston UV400 240GB e uma placa de vídeo NVIDIA GTX 960 com 1024 núcleos CUDA e 4GB de memória GDDR5.

O sistema Operacional utilizado foi o *Windows* 10 64 bits.

3.4 CUDA TOOLKIT 8

Foi utilizada a ferramenta CUDA *Toolkit* 8[3] disponibilizada pela NVIDIA para integração com o Visual Studio 2013.

Ao instalar o CUDA *Toolkit* 8, foi instalado o NVIDIA NSight[4] no Visual *Studio* 2013. Essa ferramenta é a plataforma de computação heterogênea para implementação, depuração e otimização da programação CUDA no Visual *Studio* 2013. Desta maneira o programador poderá obter conhecimento de como está ocorrendo a implementação do código na GPU.

[1] https://www.visualstudio.com/en-us/news/releasenotes/vs2013-community-vs
[2] https://www.kingston.com/br
[3] https://developer.nvidia.com/cuda-toolkit
[4] http://www.nvidia.com/object/nsight.html

3.5 CUDAFY

A biblioteca utilizada para a programação com a GPU foi o CUDAFY[5], uma biblioteca que visa facilitar a programação de alto desempenho GPGPU com a linguagem C#.

O papel da biblioteca CUDAFY é possibilitar a programação para a GPU utilizando a linguagem C#, desta maneira, a conversão do código C# para a linguagem C fica a encargo da biblioteca, para ser compilado pelo CUDA.

3.6 OBTENÇÃO DOS DADOS PARA RECONSTRUÇÃO DAS IMAGENS

A base de dados com as atenuações para a reconstrução das imagens foi obtida no site da Universidade Técnica de Denmark [6], esses dados fazem parte do projeto *The Visible Human* [7] da biblioteca nacional de medicina dos Estados Unidos.

O dataset é composto por 13 arquivos que contém dados de atenuações de raio x para reconstrução da imagens 2D de tomografia computadorizada, possibilitando a reconstrução de 13 imagens 2D.

3.7 IMAGEM DICOM

Para a reconstrução das imagens 2D foi utilizado o padrão de imagens *Digital Imaging and Communications in Medicene* (DICOM[8]). Esse padrão é um conjunto de normas adotado

[5] https://cudafy.codeplex.com/
[6] http://bme.elektro.dtu.dk/31545/?ct_data/ct_data.html
[7] https://www.nlm.nih.gov/research/visible/visible_human.html
[8] http://dicom.nema.org/

na área da medicina a fim de padronizar as imagens médicas e a forma de comunicação entre os equipamentos.

Os arquivos DICOM tem extensão ".DCM". É um conjunto de dados que vai além da imagem, como dados do paciente, médicos envolvidos e equipamentos utilizados.

Como o estudo se utiliza de imagens de tomografia computadorizada, e o padrão DICOM rege essa categoria de aparelho, as imagens do estudo foram reproduzidas em formato DICOM.

Para criar imagens Dicom, foi utilizada a biblioteca Clear Canvas[9] disponibilizada para a linguagem C#.

3.8 BENCHMARK

O *BENCHMARK* foi efetuado por meio do tempo de reconstrução que cada abordagem consume. foi colocado um *TIMER* nas abordagens da GPU e CPU, com a finalidade de analisar o tempo que cada procedimento consumiu para terminar o processo.

Foram analisadas reconstruções com os tipos de dados de precisão simples, *float* 32 bits, e com precisão dupla, *double* 64 bits. Dados do tipo decimal 128 bits não foram avaliados, visto que a GPU não tem suporte para esse tipo de dados conforme Whitehead e Fit-florea (2011).

Cada bateria de teste é composta por 13 imagens reconstruídas 10 vezes, somando um total de 130 imagens reconstruídas por abordagem, foram utilizadas três abordagens diferentes no estudo, a abordagem massivamente paralela em uma GPU, paralela e sequencial em uma CPU convencional, desta maneira obtendo 390 imagens por teste. Foram efetuadas 5 baterias de testes, somando um total de 1.950 imagens reconstruídas. Esse teste foi executado para cada tipo de dados, *float* e *double*.

Para cada bateria de teste foi calculado a média e o desvio padrão, verificando a dispersão dos dados em relação a média, com a finalidade de analisar se existem pontos discrepantes.

[9] https://www.clearcanvas.ca/

Para garantir que o *benchmark* seja executado sem interferência, foi desativada a desfragmentação automática do *Windows*, o antivírus *Windows Defender* e a rede.

Vale ressaltar que só foi considerado o tempo para construir a imagem a partir do arquivo de projeção. O tempo para transformar a imagem em padrão DICOM não foi considerado no estudo.

Com base nos resultados, foram gerados gráficos com o objetivo de verificar qual a abordagem é mais vantajosa para solucionar o problema de reconstrução de imagens 2D de tomografia computadorizada.

3.9 QUALIDADE DAS IMAGENS

As imagens geradas nas duas abordagens CPU e GPU, foram avaliadas utilizando os algorítimos relação sinal ruído de pico PSNR e índice de estrutura similar SSIM.

Para a métrica de qualidade SSIM foi utilizado o software ImageJ[10], porém, foi necessário instalar o *plugin* para análise SSIM disponibilizado no *site* do ImageJ, na seção *plugins*.

Em concordância a (AL-NAJJAR; SOONG, 2012), PSNR é uma estimativa da imagem reconstruída com a original, com base nas diferenças de *pixels* entre as duas imagens, possibilitando a comparação das imagens entre as abordagens. O Algorítimo PSNR é dado pela equação 11.

A equação 10 verifica as diferenças entre cada pixel nas imagens L e K, efetuando uma divisão pelo produto do total de linhas por colunas da matriz que representa essas imagens.

$$MSE = \frac{1}{m*n}\sum_{i}^{m}\sum_{j}^{n}(L(i,j) - k(i,j)) \tag{10}$$

$$PSNR = 20*\log_{10}(\frac{MAX_i}{\sqrt{MSE}}) \tag{11}$$

Para o PSNR, quanto maior for o resultado do cálculo da equação, dado em decibéis

[10]https://imagej.nih.gov/ij/

dB, mais próximas são as imagens.

Para o cálculo do SSIM, o desenvolvedor do *plugin* descreve que o algorítimo foi realizado conforme (WANG et al., 2004), seguindo a equação 12.

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{12}$$

Cada termo da equação 12, refere-se á informações sobre luminância, contraste e coeficiente de correlação da imagem e segundo (HORE; ZIOU, 2010), o método SSIM é utilizado para verificar a semelhança das imagens a partir da perda de luminância, correlação, distorção e distorção de contraste.

Para o SSIM, se o resultado for 1, as imagens são iguais, caso seja próximo a 1, as imagens são semelhantes.

3.10 APRESENTAÇÃO DAS IMAGENS E DADOS

Para apresentação das imagens e dados relativos ao *BENCHMARK*, foram utilizadas telas *Graphical User Interface* (GUI), que foram implementadas conforme sugere o padrão de projeto *Model View Controller* (MVC).

O padrão MVC ilustrado na Figura 17 separa a interface do usuário, do resto do sistema em 3 camadas, conforme (GALLOWAY BRAD WILSON, 2014; BURBECK, 1992);

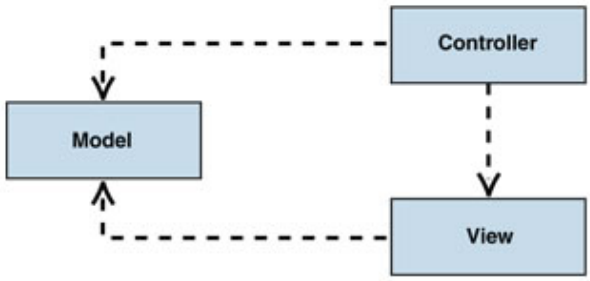

Figura 17 – Ilustração do funcionamento de um projeto MVC.
Fonte: https://msdn.microsoft.com/en-us/library/ff649643.aspx

- Model: Conjunto de classes que descreve os dados que estão sendo manipulados, bem como as regras para manipular tais dados;
- View: Controla como os dados serão exibidos para o usuário;
- Controller: Capta os eventos do usuário e informa a camada de Model e/ou View para se alterar conforme as ações do usuário.

4 DESENVOLVIMENTO

A implementação do aplicativo para reconstrução de imagens de tomografia computadorizada 2D, iniciou-se pelo desenvolvimento do algorítimo que executa o filtro de convolução, para tratar os dados de entrada, retirando ruídos da imagem.

A seguir segue o algorítimo do filtro de convolução aplicado nos dados de entrada.

```
1
2     for i <- 0 to qtdProjecoes do
3     |   for j <- 0 to tamanhoImg do
4     |    |     for k <- 0 to tamanhoImg do
5     |    |     |
6     |    |     |    funcConvolucao[k] = (-1.0 /
7     |    |     |        ((4.0 * Math.Abs(j - k) *
8     |    |     |             Math.Abs(j - k)) - 1.0));
9     |    |     end
10    |    |
11    |    |     for l <- 0 to tamanhoImg do
12    |    |      |    mult[l] = funcConvolucao[l] *
13    |    |      |                dadosEntrada[i, l];
14    |    |      |    temp1 += mult[l];
15    |    |      end
16    |    |
17    |    |      //Linha de dados convoluidos
18    |    |      Ldc[i, j] = temp1;
19    |    |
20    |    end
21    end
```

O algorítimo acima, demonstra a obtenção e aplicação do filtro de convolução nos dados de entrada das atenuações de raio-x.

Antes de iniciar o algorítimo de *backprojection*, é efetuado o calculo do seno e cosseno para cada ângulo θ, de acordo com o valor da quantidade de projeções informado, desta maneira, diminuindo os cálculos efetuados pelo aplicativo.

```
1
2       for i <- 0 to qtdProjecoes do
3       |   theta =    i * Math.PI / proj;
4       |   sin_t[i] = Math.Sin(theta);
5       |   cos_t[i] = Math.Cos(theta);
6       end
```

Com os dados normalizados pelo filtro de convolução, os senos e cossenos calculados, inicia-se o algorítimo de *backprojection* para criar a imagem 2D.

```
1
2           for i <- 0 to qtdProjecoes do
3           |   ro = cos_t[k] * kxk + sin_t[k] * kyk;
4           |   mb = ((ro * mp * tw) + mp);
5           |   lb = Math.Floor(mb);
6           |   hb = Math.Ceiling(mb);
7           |   frac = mb - lb;
8           |
9           |   //Ldc[ , ] = linha de dados convoluidos,
10          |   //matriz resultante apos aplicacao
11          |   //do filtro de covolucao.
12          |   soma += ((Ldc[k, lb]) * (1.0f - frac) +
13          |                       frac * (Ldc[k, hb]));
14          end
15
16
17          //normalizando a soma para assumir somente
18          //valores positivo
19          if (soma < 0) then
20          |
21          |  soma = 0;
22          |
23          end if
24
25          Imag[x, y] = (soma * (Math.PI / qtdProjecoes));
```

O código acima representa o algorítimo de *backprojection*. Após a execução deste código obtêm-se uma imagem de tamanho [x,y], onde x e y é o tamanho da imagem informado pelo usuário.

Para a execução da abordagem da CPU com *threads*, foi alterado os comandos *for()* por *Parallel.For()*, pois em C#, o método *Parallel.For()* é implementado por meio de *threads* conforme descrito no site[1] da *Microsoft*.

Para a execução da abordagem da GPU, foi utilizado a biblioteca CUDAFY, na qual é necessário criar *kernels* para serem executados na GPU, cada *kernel* executa equivalente a um *for()* das outras abordagens, desta maneira ficando similar o conteúdo dos *kernels* com o conteúdo dos *for()* das outras abordagens.

Para a execução da GPU, foi necessário previamente, alocar as variáveis na GPU antes da execução dos *kernels*, e após a execução, recuperar os dados processados da GPU, com os comandos *gpu.CopyToDevice()* e *gpu.CopyFromDevice()*.

[1] https://msdn.microsoft.com/pt-br/library/system.threading.tasks.parallel.for(v=vs.110).aspx

5 RESULTADOS

Nesse capítulo serão apresentadas as imagens reconstruídas e o desempenho que cada abordagem CPU e GPU atingiu nas reconstruções das imagens 2D. Também serão apresentados os dados referentes a qualidade das imagens geradas.

5.1 DESEMPENHO

Os resultados utilizando o tipo de dados de simples precisão *float* 32 bits, demonstram que a GPU atingiu um melhor desempenho em comparação com as demais abordagens, conforme os gráficos apresentados na Figura 18.

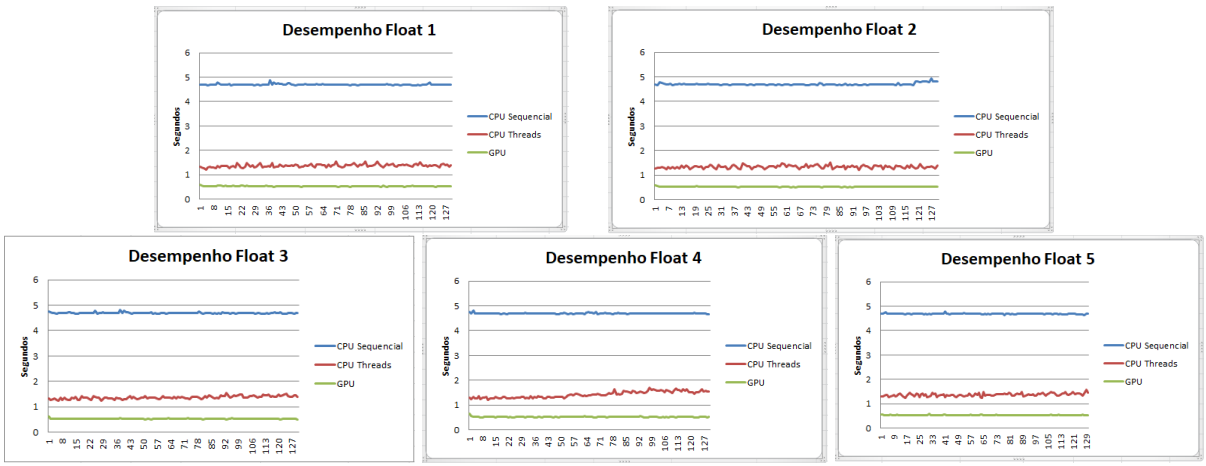

Figura 18 – Gráficos com o desempenho das reconstruções das imagens 2D utilizando o tipo de dados *float* 32 bits.

Fonte: Autoria própria

O tempo médio da GPU na reconstrução de uma imagem 2D ficou em 0,52 segundos, a reconstrução na CPU com *threads* ficou em 1,37 segundos e a CPU sequencial com 4,69 segundos. Vale ressaltar que a abordagem de CPU com *threads* teve o maior índice de desvio padrão, ou seja, é a abordagem que mais oscila em tempo de reconstrução.

Para o tipo de dados *float*, a GPU demonstrou-se eficaz e estável para a solução do problema de reconstrução de imagens 2D de tomografia computadorizada.

Referente aos resultados das reconstruções utilizando o tipo de dados *double* 64 bits, a abordagem da CPU com *threads* atingiu um melhor desempenho quanto as demais abordagens, conforme os gráficos apresentados na Figura 19.

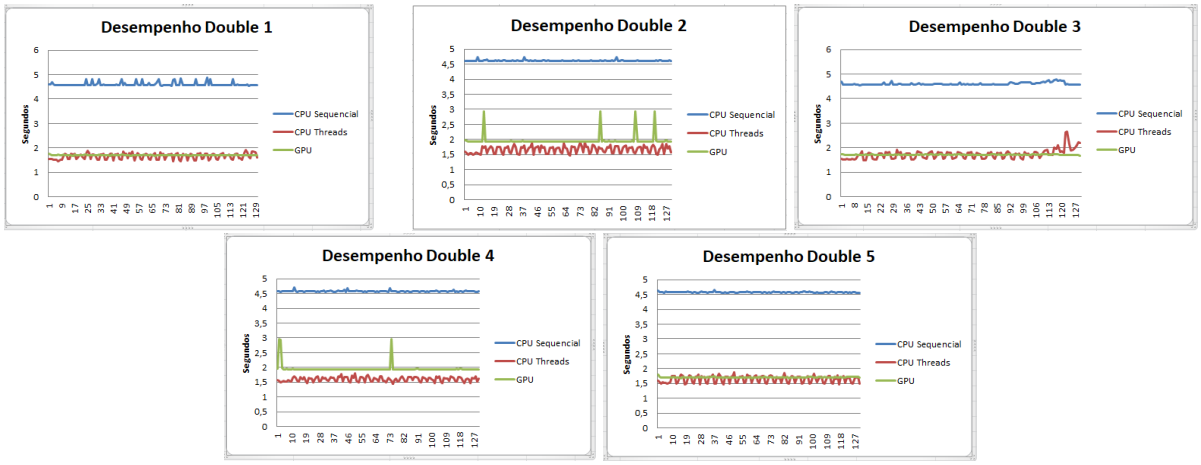

Figura 19 – Gráficos com o desempenho das reconstruções das imagens 2D utilizando o tipo de dados *float* 32 bits.

Fonte: Autoria própria

O tempo médio da CPU com *threads* nessa abordagem ficou em 1,66 segundos, a GPU obteve 1,93 segundos e a CPU sequencial com média de 4,57 segundos atingindo um melhor desempenho comparado com a versão *float*. A abordagem da CPU com *threads* ainda mantém o maior índice de desvio padrão, porém, os valores são melhores que a abordagem da GPU.

Referente o desempenho da GPU com o tipo de dados *double*, segundo Itu et al. (2011), a largura de banda, o aumento na quantidade de bytes para leitura de 4 para 8 e a arquitetura dos processadores CUDA, influenciam no desempenho da GPU, desta maneira, obtendo valores maiores que a CPU com Thread.

As imagens demonstradas na Figura 20, compõem o *dataset* das imagens utilizadas nas reconstruções.

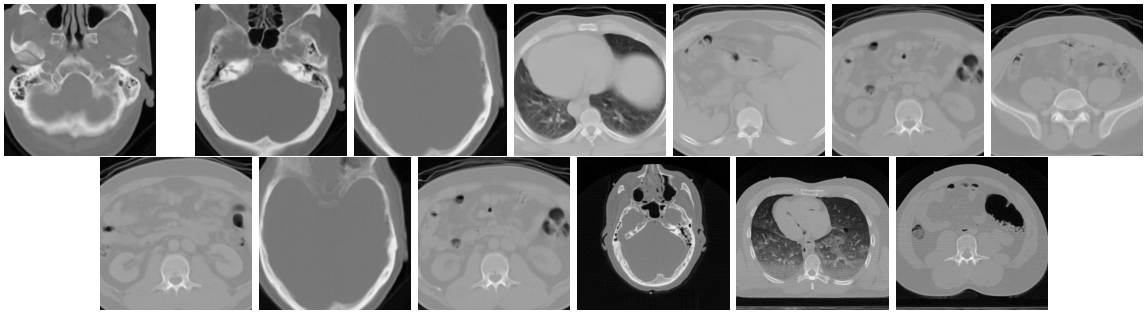

Figura 20 – Imagens 2D reconstruídas pelo aplicativo.

Fonte: Autoria própria

5.2 QUALIDADE DAS IMAGENS

A qualidade das imagens reconstruídas foram analisadas constatando-se a similaridade entre as imagens. As imagens foram analisadas por meio dos algorítimos PSNR e SSIM, conforme disposto nas tabelas 2,3,4,5.

A tabela 2, demonstra a similaridade das imagens geradas entre as abordagens, utilizando o tipo de dados *float*, por meio do algorítimo de análise de qualidade de imagem PSNR.

CPU - CPU Thread	CPU - GPU	CPU Thread - GPU
49,9449	84,8574	54,0245
55,3524	45,4621	51,5397
∞	50,8809	53,711
∞	48,0042	43,5866
36,4884	85,8053	36,7921
48,0352	85,8053	36,7921
40,1446	87,7834	46,2455
62,37	85,401	85,401
∞	53,2306	53,711
48,0352	48,0342	84,1872
36,582	38,8628	51,1872
40,5224	49,7154	52,1694
50,5089	50,5096	50,6431

Tabela 2 – Comparação entre imagens 2D, reconstruídas com o tipo de dados *float* ,utilizando o algorítimo PSNR.

A tabela 3, demonstra a similaridade das imagens geradas entre as abordagens, utilizando o tipo de dados *float*, por meio do algorítimo de análise de qualidade de imagem SSIM.

CPU - CPU Thread	CPU - GPU	CPU Thread - GPU
0,998344	0,999999	0,998599
0,998802	0,998586	0,99804
1	0,997344	0,997926
1	0,999255	0,998736
0,997545	0,999998	0,997214
0,999549	0,999998	0,999549
0,997301	0,999999	0,998282
0,999593	0,999998	0,999998
1	0,997772	0,997926
0,999549	0,99548	0,999999
0,996765	0,999548	0,999999
0,998777	0,999306	0,998677
0,994856	0,998455	0,998398

Tabela 3 – Comparação entre imagens 2D, reconstruídas com o tipo de dados *float* ,utilizando o algorítimo SSIM.

A tabela 4, demonstra a similaridade das imagens geradas entre as abordagens, utilizando o tipo de dados *double*, por meio do algorítimo de análise de qualidade de imagem PSNR.

CPU - CPU Thread	CPU - GPU	CPU Thread - GPU
49,9439	85,5985	49,9435
∞	42,8479	86,3701
53,712	53,7102	53,7102
48,0048	48,0037	48,0038
39,7314	86,1353	30,6757
48,0346	87,3092	49,2355
40,1449	86,4925	40,1446
48,2527	86,4925	86,4925
∞	87,0195	53,7102
42,9966	87,3092	42,9965
36,5819	37,4668	53,5696
40,5226	84,3744	43,5129
50,509	85,9125	44,7184

Tabela 4 – Comparação entre imagens 2D, reconstruídas com o tipo de dados *double* ,utilizando o algorítimo PSNR.

A tabela 5, demonstra a similaridade das imagens geradas entre as abordagens, utilizando o tipo de dados *double*, por meio do algorítimo de análise de qualidade de imagem SSIM.

CPU - CPU Thread	CPU - GPU	CPU Thread - GPU
0,998345	0,999999	0,998345
1	0,997993	0,999999
0,997927	0,997926	0,997929
0,999256	0,999255	0,999256
0,997	0,999998	0,996507
0,99955	0,999998	0,998029
0,997301	0,999998	0,997301
0,999152	0,999999	0,999999
1	0,999999	0,997926
0,997928	0,999998	0,997928
0,996765	0,997236	0,9990001
0,998777	0,999999	0,999202
0,998457	0,999998	0,998029

Tabela 5 – Comparação entre imagens 2D, reconstruídas com o tipo de dados *double* ,utilizando o algorítimo SSIM.

Cada linha da tabela representa uma imagem diferente, assim formando o dataset com total de 13 imagens.

Os valores PSNR ∞, significam que a imagem é igual, pois não houve diferença entre os *pixels*, obtendo 0 no denominador da equação 10. Percebe-se que para as imagens em que o PSNR recebeu ∞, o SSIM também informou que as imagens são iguais retornando o valor 1, que representa similaridade máxima entre as imagens.

Para o algorítimo PSNR , valores típicos para imagens de 8 bits são entre 30dB a 50dB. Analisando as tabelas, verifica-se que as imagens geradas são similares, pois estão no intervalo desejado.

Referente o algorítimo SSIM, nota-se que as imagens analisadas tem valores muito próximo de 1 garantindo a similaridade das imagens.

Vale ressaltar que a cada nova reconstrução, pode-se obter um valor diferente de PSNR e SSIM, devido ao uso de dados do tipo ponto flutuante. Conforme Goldberg (1991), esses tipos de dados não são exatos, existe uma aproximação sistemática devido a limitação no armazenamento de bits, desta maneira, ocorrem truncamentos e arredondamentos conforme as regras da (IEEE) *Institute of Electrical and Electronics Engineers*, por meio das normas 754 e 854.

6 CONCLUSÃO

Na utilização de dados *float* 32 bits é recomendável a utilização da GPU para a reconstrução de imagens 2D de tomografia computadorizada, pois além de se ter ganho em desempenho, as imagens geradas são semelhantes as demais abordagens.

Referente a abordagem *double* 64 bits, vale ressaltar que é necessário verificar a arquitetura da CPU e GPU, pois pode ocorrer diferença no desempenho dessas abordagens. Nesse estudo, constatou-se melhor performance na abordagem CPU com *threads*, também é a abordagem com maior desvio padrão, sendo mais instável.

Desta maneira conclui-se que todas as abordagens são confiáveis referente a qualidade das imagens geradas, conforme as métricas PSNR e SSIM, porém, vale ressaltar que é necessário uma análise prévia quanto ao tipo de dados a ser utilizado, pois, constatou-se diferença no desempenho que cada abordagem atingiu, conforme o tipo de dados utilizado.

7 TRABALHOS FUTUROS

Desenvolver um aplicativo para reconstrução e análise de imagens de tomografia 3D, utilizando programação CUDA.

REFERÊNCIAS

AL-NAJJAR, Y. A. Y.; SOONG, D. C. Comparison of Image Quality Assessment :. v. 3, n. 8, p. 1–5, 2012. Disponível em: <https://www.ijser.org/researchpaper/Comparison-of-Image-Quality-Assessment-PSNR-HVS-SSIM-UIQI.pdf>.

ANVISA, A. N. de V. S. **Manual de instrução de uso do aparelho de tomografia.** 2016. Disponível em: <http://www4.anvisa.gov.br/base/visadoc/REL/REL[13309-1-2].PDF>.

ASL, M. N.; SADREMOMTAZ, A. Analytical image reconstruction methods in emission tomography. **Journal of Biomedical Science and Engineering**, v. 06, n. 01, p. 100–107, 2013. ISSN 1937-6871. Disponível em: <https://file.scirp.org/pdf/JBiSE_2013013114261034.pdf>.

BURBECK, S. Applications Programming in Smalltalk-80 (TM): How to use Model-View-Controller (MVC). **Smalltalk-80 v2**, v. 80, n. Mvc, p. 1–11, 1992. Disponível em: <http://www.dgp.toronto.edu/~dwigdor/teaching/csc2524/2012_F/papers/mvc.pdf>.

CALIFORNIA, U. of. Spectrophotometry. 2016. Disponível em: <http://chem.libretexts.org/Core/Physical_and_Theoretical_Chemistry/Kinetics/Reaction_Rates/Experimental_Determination_of_Kinetcs/Spectrophotometry>.

GALLOWAY BRAD WILSON, K. S. A. D. M. J. **Professional ASP.NET MVC 5**. [S.l.]: Wrox Press, 2014. ISBN 978-1-118-79475-3.

GOLDBERG, D. What every computer scientist should know about floating-point arithmetic. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 23, n. 1, p. 5–48, mar. 1991. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/103162.103163>.

HERMAN, G. T. **The Fundamentals of Computerized Tomography, Image reconstruction from projections**. Second edition. [S.l.]: Springer, 2009. ISSN 1617-7916.

HORE, A.; ZIOU, D. Image quality metrics: Psnr vs. ssim. In: **2010 20th International Conference on Pattern Recognition**. [S.l.: s.n.], 2010. p. 2366–2369. ISSN 1051-4651.

INTEL. **7ª geração do processador Intel® Core™ i7**. 2016. Disponível em: <https://www-ssl.intel.com/content/www/br/pt/processors/core/core-i7-processor.html>.

ITU, L.; SUCIU, C.; MOLDOVEANU, F. Comparison of single and double floating point precision performance for Tesla architecture GPUs. **Bulletin of the Transilvania University of Brasov**, v. 4, 2011.

KAK, A.; SLANEY, M. **Principles of computerized tomographic imaging**. [S.l.]: IEEE Press, 1987. ISBN 0-87942-198-3.

KIRK, D. B.; HWU, W.-M. W. **Programando para processadores paralelos**. [S.l.]: Elsevier, 2011. 5 p. ISBN 978-85-352-4188-4.

KIRK, D. B.; HWU, W.-M. W. **Programando para processadores paralelos**. [S.l.]: Elsevier, 2011. 2 p. ISBN 978-85-352-4188-4.

KIRK, D. B.; HWU, W.-M. W. **Programando para processadores paralelos**. [S.l.]: Elsevier, 2011. ISBN 978-85-352-4188-4.

LIU, Y.; MASKELL, D. L.; SCHMIDT, B. CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. **BMC research notes**, v. 2, p. 73, 2009. ISSN 1756-0500. Disponível em: <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2694204>.

NACIF, M.; SANTOS, E. **MANUAL DE TECNICAS EM TOMOGRAFIA COMPUTADORIZADA**. [S.l.]: RUBIO, 2009. ISBN 9788577710058.

NERSISSIAN, D. Y. Tomografia computadorizada, tecnologia e funcionamento dos equipamentos. 2016. Disponível em: <http://rle.dainf.ct.utfpr.edu.br/hipermidia/images/documentos/Tomografia_computadorizada_tecnologia_e_funcionamento_equipamentos.pdf>.

NVIDIA. **About CUDA**. 2016. Disponível em: <https://developer.nvidia.com/about-cuda>.

NVIDIA. **CUDA programação paralela facilitada**. 2016. Disponível em: <http://www.nvidia.com.br/object/cuda_home_new_br.html>.

NVIDIA. **o que é computação com aceleração de GPU?** 2016. Disponível em: <http://www.nvidia.com.br/object/what-is-gpu-computing-br.html>.

PARENTE, R. E. tomografia de susceptibilidade magnética com magnetômetro supercondutor squid. **Tese (Doutorado em engenharia elétrica)**, 1996.

RAO, R.; KRIZ, R. Parallel Implementation of the Filtered Back Projection Algorithm for Tomographic Imaging. **Review Literature And Arts Of The Americas**, p. 1–18, 2016.

REBELO, A. M. de O. RECONSTRUÇÃO DE IMAGEM EM TOMOGRAFIA COMPUTADORIZADA USANDO O MÉTODO DE CONVOLUÇÃO. **Tese(MESTRE EM CIÊNCIAS (M. Sc.) EM ENGENHARIA NUCLEAR)**, 1984.

SOLOMON, C.; BRECKON, T. **Fundamentos de processamento digital de imagens: uma abordagem pratica com exemplos em Matlab**. [S.l.]: LTC, 2013. ISBN 9788521623472.

WANG, Z.; BOVIK, A. C.; SHEIKH, H. R.; SIMONCELLI, E. P. Image quality assessment: From error visibility to structural similarity. **IEEE Transactions on Image Processing**, v. 13, n. 4, p. 600–612, 2004. ISSN 10577149.

WHITEHEAD, N.; FIT-FLOREA, A. Precision & Performance : Floating Point and IEEE 754 Compliance for NVIDIA GPUs. **NVIDIA white paper**, v. 21, n. 10, p. 767–75, 2011. ISSN 03331024. Disponível em: <http://developer.nvidia.com/content/precision-performance-floating-point-and-ieee-754-compliance-nvidia-gpus>.