Dissertação de Mestrado

SIMULAÇÃO DE TOMOGRAFIA COMPUTADORIZADA DE RAIOS X UTILIZANDO PROGRAMAÇÃO PARALELA EM SISTEMAS DE PROCESSAMENTO DE ALTO DESEMPENHO

Mauricio Quelhas Antolin

Orientadores: Marcelo Portes de Albuquerque (CBPF)

Luis Fernando de Oliveira (UERJ)

Mestrado em Instrumentação Científica

Centro Brasileiro de Pesquisas Físicas

Rio de Janeiro, 17 de Setembro de 2009

Agradecimentos

Aos meus orientadores Marcelo Portes de Albuquerque e Luís Fernando de Oliveira pelos seus ensinamentos e pela paciência.

As pessoas do CAT do CBPF pela receptividade e a ajuda na utilização dos *cluster* SsolarI e SsolarII, em especial ao Marcelo Giovani.

Ao CESUP/UFRGS nas pessoas de Denise Grüne Ewald, Luís Fernando Fernandez e Carlos H. Brandt por permitirem a utilzação do *cluster* gauss.

Ao meu pai, Manuel Angel Antolin Garcia por me incentivar a ingressar na carreira acadêmica e por todo o seu carinho.

À minha mãe, Iliana de Carvalho Quelhas pela paciência, amor, carinho e apóio em toda minha caminhada acadêmica e pessoal.

Aos meus irmãos Rafael, Leonardo e Paola e aos amigos Adilson Ferreia Gomes, Thiago, Luís Roberto, Herder, Luciana Iecker, Luciana Maria, Beto, Gabriel por me aturarem à tanto tempo.

À minha esposa Gisele pela ajuda e compreensão desde que estamos juntos. Obrigado meu amor!

Sumário

Lista de Ilustrações

Lista de Tabelas

Lista de abreviaturas e siglas

- UML - Linguagem de modelagem unificada
- TC - Tomografia computadorizada
- CESUP - Centro Nacional de Supercomputação - Universidade Federal do Rio Grande do Sul
- CARM++ - Biblioteca paralela para C++
- MC - Monte Carlo
- 3D - Três dimenções
- CCD - Sensor para captação de imagem (Dispositivo de Carga Acoplado)
- GEANT - Ferramenta de simulação para física de altas energias
- EGS4 - Simulador de transporte de elétrons e fótons para geometrias arbitrárias
- MCNP - Código Monte Carlo para transporte de nêutrons, elétrons ou fótons
- MPI - Message Passing Interface
- AMPI - Adaptive Message Passing Interface
- NIST - National Institute of Standards and Technology
- IAEA - International Atomic Energy Agency
- .raw - Arquivo de imagem em formato binário
- HPC - High Performance Computing
- QoS - Quality of Service

RESUMO

A tomografia computadorizada (TC) é uma técnica de ensaio não-invasivo que permite a visualização de estruturas internas das amostras. Em função disto, sua aplicação nas áreas de saúde (médica) e tecnológicas (industrial) tornou-se imprescindível. Com este intuito foi desenvolvido um simulador de tomografia computadorizada para a geração de dados que auxiliem no desenvolvimento de algoritmos de reconstrução de imagem e reconstrução volumétrica para as técnicas de TC baseadas em espalhamento coerente (Rayleigh) e em transmissão de raios X. O simulador proposto é baseado no método Monte Carlo (MC) por ser largamente utilizado na solução de problemas envolvendo processos estatísticos com características estocásticas da emissão de radiação, transporte e detecção de processos radioativos. Este método é extremamente útil em problemas complexos que não podem ser modelados por códigos computacionais que utilizam métodos determinísticos. Outra grande aplicação do método Monte Carlo é feita em experimentos onde os resultados são de difícil medição ou incluem grandes quantidades de radiação em um paciente. Para contornar o problema de alta carga computacional exigida pelo MC o simulador foi desenvolvido em CHARM++, por ser uma biblioteca voltada para a programação paralela. Este programa foi executado em Clusters Linux a fim de diminuir o tempo de simulação.

ABSTRACT

Computer Tomography (CT) is a non-destructive technique because it does not compromise directly the sample integrity. Therefore application in the medical and industrial environments became essential. This thesis presents the development of parallel algorithms for Computer Tomography simulation based on coherent scattering (Rayleigh) and X-ray transmission. The results of the simulations are visualized in images with planar reconstructions. The proposed simulator is based on the widely used Monte Carlo method (MC) to solve statistical problems with stochastic characteristics, e.g. the radiation emissions, transport and detection of radioactive processes. This method is extremely useful in complex problems that cannot be modeled by deterministic computational methods. Another important application of the simulator is to carry out experiments which the results are difficult to measure due to the large amounts of radiation delivered to patients. To avoid the problem of high computational load required by the MC, the simulator was developed in CHARM ++ language and the program was run on Linux clusters to improve the performance.

Capítulo 1

Introdução

Diferentes áreas da ciência e da tecnologia utilizam as técnicas de geração de imagem para inspeção, avaliação, estudo e diagnóstico. Cada aplicação requer um conjunto próprio de equipamentos e procedimentos a fim de produzir os resultados esperados. Nos sistemas de imagem que utilizam raios X, a escolha da fonte e do(s) detector(es), além dos parâmetros de operação (tensão, corrente, ganho, geometria, tempo de exposição), devem ser adequadas a cada aplicação.

Uma das áreas de interseção entre a ciência física e as ciências biológicas é a produção de dados e imagens para caracterização e análise de amostras de tecidos biológicos utilizando radiação ionizante. Estas imagens podem ser obtidas por diferentes métodos, através de técnicas experimentais e/ou de simulação em computador, que podem ser classificados em métodos de projeção de imagem e métodos tomográficos. Ambos os métodos se valem dos mecanismos de interação da radiação com a matéria para gerar a informação que caracterizará a amostra biológica e que será impressa em um dispositivo de saída. A imagem resultante está carregada de informação que precisa ser processada para tornar-se útil ao usuário final.

Para o desenvolvimento de novas técnicas de imagem ou o simples melhoramento de uma determinada técnica requer da equipe de profissionais um esforço de pesquisa que se

traduz em tempo, custos e muito trabalho, particularmente quando é necessário avaliar um número muito grande de parâmetros que podem influenciar a qualidade da imagem final.

Sob esta perspectiva, diferentes técnicas foram desenvolvidas para a geração de imagem, tais como: radiografia utilizando contraste de fase utilizada especialmente em mamografia, e tomografia computadorizada por espalhamento coerente de raios X, técnica esta com um grau de seletividade o que não ocorre na tomografia por transmissão de raios X.

Nas técnicas radiográficas, a imagem a ser investigada surge de forma direta como uma projeção da amostra sobre um sistema de detecção, podendo ser processada a posteriori para extração de novas informações. Nas técnicas tomográficas, as imagens geradas durante a aquisição dos dados não formam o produto final ainda, mas já trazem a essência da informação. Através das técnicas de reconstrução de imagens, estas imagens "brutas" são processadas para gerar um outro conjunto de imagens. Para a tomografia, este conjunto de imagens é a coleção de reconstruções das seções transversais da amostra estudada. Assim como a radiografia, estas imagens exibem as informações necessárias para a caracterização da amostra e posterior análise dos dados.

A simulação de imagens em computador se encontra nas mesmas circunstâncias das técnicas experimentais, implementando, em linguagem apropriada, os mecanismos de interação da radiação. As informações vitais para a execução da simulação são provenientes de experimentos já realizados ou de modelagens computacionais que são paulatinamente aperfeiçoados através dos resultados obtidos. As imagens simuladas são geradas reproduzindo-se os procedimentos experimentais em computador. Estas imagens, carregadas de informação, são processadas de forma a se extrair os dados de interesse que irão caracterizar a amostra e permitir uma análise do problema em foco (alvo de um estudo específico).

Por exemplo, a tomografia computadorizada (TC) pode ser utilizada para a realiza-

ção de estudos que não comprometam a integridade da amostra e é capaz de extrair informações sobre a estrutura da amostra e também os componentes constituintes da mesma. Em função disto, sua aplicação nas áreas de saúde (médica) e tecnológicas (industrial) tornou-se imprescindível. Entretanto, antes de disponibilizar uma nova técnica tomográfica, é necessário avaliá-la quanto à sua aplicabilidade, sendo que toda tomografia necessita de uma reconstrução de imagem dos dados adquiridos (e, eventualmente, uma visualização volumétrica no caso da tomografia 3D).

As técnicas de simulação computacional são especialmente úteis neste cenário onde diferentes parâmetros, combinados de diversas formas, produzem resultados distintos representando melhoras ou pioras na qualidade da imagem. Componentes e dispositivos podem ser avaliados quanto aos seus comportamentos através de simulação, dado que esta permite a concepção de um ambiente virtual de estudo a baixos custos[1].

O uso do método Monte Carlo (MC) para a simulação de transporte de radiação se mostrou o método mais acurado para a simulação do espectro de raio-X em geometrias complexas e permite incorporar facilmente dados de secção de choque[2]. Tipicamente, os simuladores baseados na técnica de MC funcionam através da geração de eventos e o acompanhamento da seqüência de interações destes mesmos eventos até que eles sejam eliminados do processo. Os simuladores MC para transporte de energia através de fótons de raios X funcionam neste modelo. Os fótons são gerados por uma fonte cujos parâmetros devem ser definidos (posição, direção, tamanho focal, abertura de feixe, espectro de emissão, fluência) e detectados por dispositivos sensíveis à radiação, também configurados adequadamente (composição, dimensão, direção, resolução espacial, resolução em energia, curva de eficiência).

A qualidade da imagem gerada nestes dispositivos reais depende diretamente do número de fótons detectados, isto é, dependem de uma estatística de detecção. Não é diferente na análise da qualidade das imagens geradas por simulação computacional. A

estatística é um fator determinante na qualidade do resultado. Por isso, as simulações baseadas nas técnicas de Monte Carlo consomem um esforço computacional muito grande dado o número elevado de eventos envolvido. Para mera ilustração, na modelagem de um detector tipo CCD com 1024x1024 pixels, se a resolução em bits for 8, isto é, uma contagem máxima de 256 eventos por pixel, seriam necessários $(256x1024)^2$ (=268.435.456) fótons para saturar a imagem. Se cada fóton gerado pela fonte tem um tempo médio de vida igual a 1 ms, seriam necessários 3,1 dias para que o detector saturasse (imaginando, obviamente, algum material entre a fonte e o detector, onde ocorrem interações). O tempo de simulação de uma tomografia computadorizada com 360 projeções (imagens por feixe emitido), utilizando estes parâmetros seria de 3,1x360 ($\cong$ 3 anos) de simulação.

Uma solução normalmente aplicada nestas situações é o processamento da simulação de forma paralela ou distribuída. O processo de simulação de aplicações com raios-X é fortemente paralelizável dado que o "'histórico de vida"' de um fóton não interfere no histórico dos demais fótons simulados. Isto facilita a modelagem dos simuladores deste tipo em sistemas de computação paralela ou distribuída.

A modelagem de sistema de tomografia computadorizada exige uma carga computacional ainda maior pois não se simula apenas uma imagem. Dependendo das características de qualidade da imagem reconstruída, a quantidade de projeções simuladas pode ser muito grande (na ordem de centenas ou milhares). Dependendo das características da técnica tomográfica simulada (tomografia por transmissão, ou por espalhamento coerente, ou por fluorescência de raios-X), a quantidade de fótons detectados pode ser muito pequena exigindo a geração de um número de eventos ainda maior.

Muitos pesquisadores desenvolvem simuladores de TC em programas de simulação como GEANT3, GEANT4, EGS4, MCNP [3, 4, 5, 6]. Neste trabalho optou-se por escrever o código MC com o objetivo de desenvolver o simulador de TC em um sistema de processamento distribuído devido à alta carga computacional exigida pelo método MC.

Para isso o simulador está sendo desenvolvido em Charm++ (Parallel Programming Laboratory, University of Illinois at Urbana-Champaign) por ser uma biblioteca de programação paralela para o C++. Esta biblioteca pode ser vista como uma 'fusão' entre o MPI (Message Passing Interface) e o C++, o que fornece uma maior flexibilidade quando estamos realizando programação distribuída, pois ela já incorpora algumas funções específicas para a comunicação entre computadores em um cluster e, também, permite a utilização de toda a potencialidade da programação orientada a objeto.

O desenvolvimento de um simulador de TC deve incorporar todo o sistema, ou seja, a geração de cada fóton, o acompanhamento deste da sua origem até o momento de detecção ou absorção pela amostra, a simulação da própria amostra (seus componentes e geometria) e os detectores.

Nesta primeira versão, o simulador de TC focou a movimentação do sistema fonte detector e da interação da radiação com a amostra, ou seja, foi dada maior ênfase ao funcionamento mecânico e físico do sistema.

Este trabalho apresenta os fundamentos teóricos (físicos e computacionais) para o desenvolvimento de um simulador de tomografia computadorizada por transmissão de raio-X. Em seguida é apresentada a metodologia da simulação dos fenômenos físicos envolvidos e como o simulador realiza o processo da tomografia computadorizada, incluindo os equipamentos necessários para realizarmos um processamento distribuído, e finalmente, são apresentados os resultados obtidos e alguns testes do simulador que avaliaram a fidelidade dos dados simulados e o tempo de simulação.

Capítulo 2

Fundamentos teóricos

Neste capítulo é descrito toda a base física das interações da radiação com a amostra relevantes para o simulador, como o efeito Compton, Rayleigh e transmissão de raio-X. Também são descritos alguns sistemas tomográficos assim como a implementação do MC.

2.1 Interação da radiação com a matéria

Quando um material é atravessado por uma radiação ionizante pode ocorrer interação da radiação com os átomos, com os elétrons ou com o núcleo do material. O processo de interação da radiação com a matéria pode ser descrito de maneira clássica como uma colisão, mesmo quando as interações envolvem forças de ação à distância.

As interações que ocorrem com núcleos individuais são observadas apenas para radiação incidente de altas energias, cujo comprimento de onda associado tem a mesma ordem de grandeza que o diâmetro do núcleo. Já as interações que ocorrem com o todo do átomo só ocorrem com radiações muito fracas, pois o comprimento de onda é da ordem de grandeza do átomo. Neste caso a interação pode ser incoerente (efeito Compton) ou coerente (Rayleigh).

A interação coerente ocorre simultaneamente com vários átomos o que produz efeitos

ondulatórios (superposição e padrões de difração). O efeito Compton é puramente um efeito de colisão de partículas que leva em consideração efeitos relativísticos.

2.1.1 Efeito fotoelétrico e radiação secundária

É uma interação em que ocorre total absorção do fóton incidente, podendo ocorrer a ejeção de um fotoelétron de uma das camadas do átomo. Se a radiação incidente for um raio gama com energia suficiente para tal, é muito provável que a origem do fotoelétron seja a camada "K" do átomo com uma energia dada por:

$$E_{e^-} = h\nu - E_b \tag{2.1}$$

onde E_b representa a energia de ligação do fotoelétron na sua camada original.

Para raios gama com energias maiores que algumas centenas de KeV, o fotoelétron fica com a maioria da energia do fóton.

A ionização criada pela ejeção de um fotoelétron pode ser rapidamente neutralizada pela captura de um elétron livre do meio em que o átomo se encontra ou pela redistribuição dos elétrons de outra camada do átomo.

Este efeito é a forma predominante para a interação de raios gama (ou raios-X) de energia relativamente baixa. Se o material absorvedor tiver um número atômico alto a probabilidade de ocorrer efeito fotoelétrico aumenta.

A absorção, na interação fotoelétrica, pode ser aproximada por:

$$\tau = K \times \frac{Z^n}{E^3(2.2)}$$

onde n varia entre 4 e 5 para a região do raio gama de interesse e K é uma constante.

2.1.2 Espalhamento incoerente: Compton

É um dos mecanismos mais freqüêntes de interação entre fótons emitidos por radioisótopos e elétrons livres. O efeito Compton é uma colisão entre o fóton incidente e um elétron do alvo onde o fóton incidente é espalhado em um ângulo θ em relação a sua direção original. Por ser uma colissão inelástica a energia cinética não se conserva, o que ocorre é uma transferência de parte da energia do fóton incidente para o elétron (que é chamado elétron espalhado). Devido ao ângulo de espalhamento ser aleatório a energia transferida para o elétron varia de zero até a energia do fóton. Assim, a energia do sistema antes e depois da colisão é, respectivamente, dada por

$$E = h\nu + m_0c^2 \tag{2.3}$$

$$E^{'} = h\nu^{'} + mc^2 \tag{2.4}$$

onde m_0 é a massa de repouso do elétron e ν é a freqüência do fóton.

Considerando que o elétron espalhado está em repouso antes da colisão, podemos escrever a conservação do momento linear como

$$h\nu = h\nu^{'}cos\phi + mvcos\theta \tag{2.5}$$

$$0 = h\nu^{'}sen\phi + mvsen\theta \tag{2.6}$$

onde θ e ϕ são os ângulos de espalhamento (ver figura 2.1).

Resolvendo o sistema acima, podemos mostrar que

$$h\nu^{'} = \frac{h\nu}{1 + \frac{h\nu}{m_0c^2}(1 - cos\theta)} \tag{2.7}$$

onde m_0c^2 é a energia de repouso do elétron (cerca de $0,511MeV$) [7]

A probabilidade de ocorrência do espalhamento Compton por átomo do material alvo depende do número de elétrons disponíveis no alvo, e cresce linearmente como número atômico Z. A distribuição angular para fótons espalhados é modelada pela a equação de *Klein-Nishina* para uma seção de choque diferencial:

$$\frac{d\sigma}{d\Omega} = r_0^2 \left(\frac{1+cos^2\theta}{2}\right) \left[\frac{1}{1+\alpha(1-cos\theta)}\right]^3 \left[1 + \frac{\alpha^2(1-cos\theta)^2}{(1+cos^2\theta)[1+\alpha(1-cos\theta)]}\right] \quad (2.8)$$

onde $\alpha \equiv \frac{h\nu}{m_0c^2}$ e r_0 é o raio clássico do elétron[13] .

Com pequenas manipulações algébricas podemos escrever a equação 2.8 como

$$\frac{d\sigma}{d\Omega} = \frac{r_0^2}{2} \left(\frac{E_c}{E}\right)^2 \left(\frac{E_c}{E} + \frac{E}{E_c} - sen^2\theta\right) \quad (2.9)$$

onde $E_c \equiv h\nu'$.

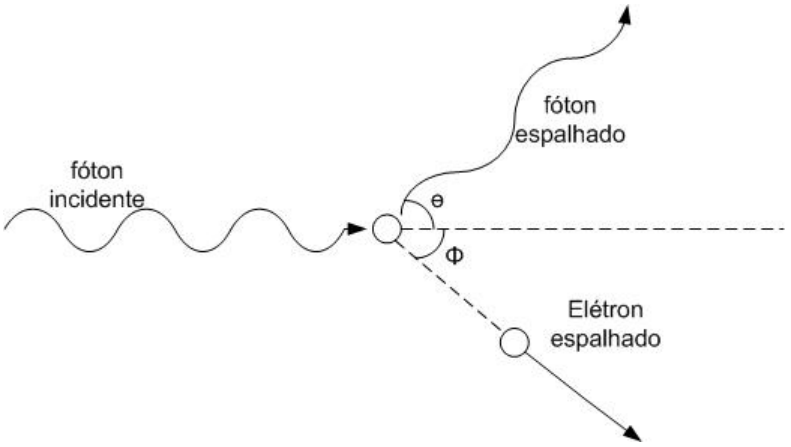


Figura 2.1: Representação do efeito *Compton*

2.1.3 Espalhamento coerente: Rayleigh

As interações dos raios X com a matéria podem ser descritas em termos da seção de choque diferencial e total. A seção de choque diferencial descreve a distribuição angular do espalhamento em termos dos ângulos polar (θ) e azimutal (ϕ). Já a seção de choque

total mede o espalhamento total do feixe pela a amostra e é obtida pela a integração da seção de choque diferencial.

Se o raio espalhado tem o mesmo comprimento de onda e freqüencia que o raio X incidente, dizemos que o raio espalhado é corente com o incidente ou que o espalhamento é coerente. Como os raios-X podem ser espalhados em todas as direções ao interagir com um elétron ligado ao átomo a, intensidade do espalhamento depende do ângulo do raio-X espalhado. De acordo com Thomson a intensidade do feixe espalhado (I) por um elétron é dada por [9]

$$I = I_0 \frac{e^4}{r^2 m^2 c^4} \left(\frac{1 + cos^2 2\theta}{2} \right) \tag{2.10}$$

onde I_0 é a intensidade inicial do feixe e c é a velociadade da luz.

O fenômeno da difração ocorre quando um feixe de raios X incide com um ângulo θ sobre um conjunto de planos cristalinos de distância interplanar d. Quando a diferença de caminho óptico for igual a um multiplo inteiro do comprimento de onda do raio incidente, este raio será refletido por dois planos subsequentes. Devido a esta reflexão pode ocorrer superpoção construtiva e o fenômeno da difração será observado; caso contrário, haverá superposição destrutiva, i.e. não se observará qualquer sinal de raios X. Este fenômeno é descrito pela lei de *Bragg*, representado na figura 2.2.

$$n\lambda = 2dsin(\theta) \tag{2.11}$$

2.2 Sistemas tomográficos

A tomografia computadorizada (TC) é uma ferramenta importante no estudo de amostras, pois fornece informações que não estão disponíveis através da inspeção visual direta. Ela é classificada como uma técnica de ensaio não-invasivo que permite a

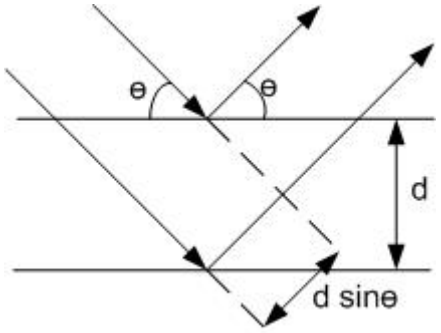


Figura 2.2: Representação da lei de *Bragg*

visualização de estruturas internas das amostras. Em função disto, sua aplicação nas áreas de saúde (médica) e tecnológicas (industrial) tornou-se imprescindível [3].

A TC utiliza um aparelho de raios-X que gira em torno da amostra, fazendo radiografias transversais da mesma. Estas radiografias são convertidas por um computador nos chamados cortes tomográficos. Isto quer dizer que a TC constrói imagens internas das estruturas da amostra através de cortes transversais, de uma série de seções fatiadas que são posteriormente montadas pelo computador para formar um quadro completo.

Se uma propriedade de um objeto é descrita por uma função $f(x, y, z)$ e se todas as projeções desta função através do objeto no plano z são conhecidas, é possível calcular o valor de uma função $f(x, y)$ em cada ponto do plano z. Para isso, a única exigência é que cada linha da projeção deve ser uma função linear do objeto a ser caracterizado.

Assim as técnicas de TC por transmissão são classificadas em gerações pela movimentação do sistema fonte-detector e forma do feixe [8]. Abaixo temos algumas classificações de sistemas tomográficos.

2.2.1 TC de Primeira Geração

Um tomógrafo de primeira geração é caracterizado pelo uso de apenas um detector. Assim, para realizar todo o processo tomográfico, o detector é transladado em conjunto com a fonte de raios-X de forma a adquirir uma projeção. Ao percorrer toda a extensão da

amostra o sistema (fonte-detector) gira de um pequeno incremento angular e o processo de translação é repetido. Outra característica da TC de primeira geração é a forma do feixe da fonte de raio-X, que nesta configuração é de feixes paralelos e bem colimados (figura 2.3). As tomografias de 1ª geração também são chamadas de tomografias de transmissão. A intensidade do feixe que percorre a amostra é dado por:

$$I = I_0 e^{-\mu x} \tag{2.12}$$

onde I_0 é a intensidade inicial do feixe, x é a espessura do material e μ é o coeficiente de atenuação (constante para cada material).

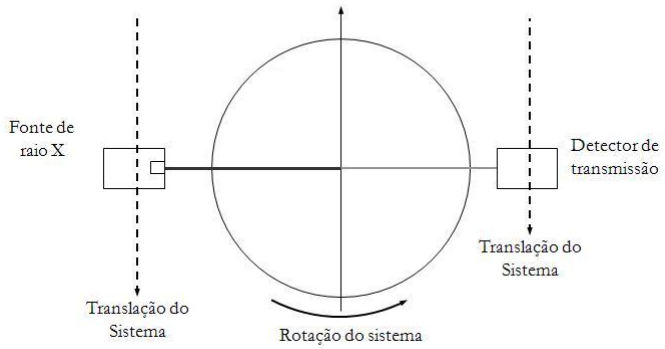


Figura 2.3: Representação de um sistema tomográfico de primeira geração

2.2.2 TC de Segunda Geração

A principal diferença da tomografia de primeira geração para a tomografia de segunda geração é a utilização de uma fonte de raio x de feixes divergentes no plano e a substituição do detector por um vetor de detectores. Desta forma a translação do sistema fonte-detector não é mais nescessária. Porém para inspecionar a amotra como um todo o sistema de TC de 2ª geração mantém a rotação em torno da amostra (figura 2.4).

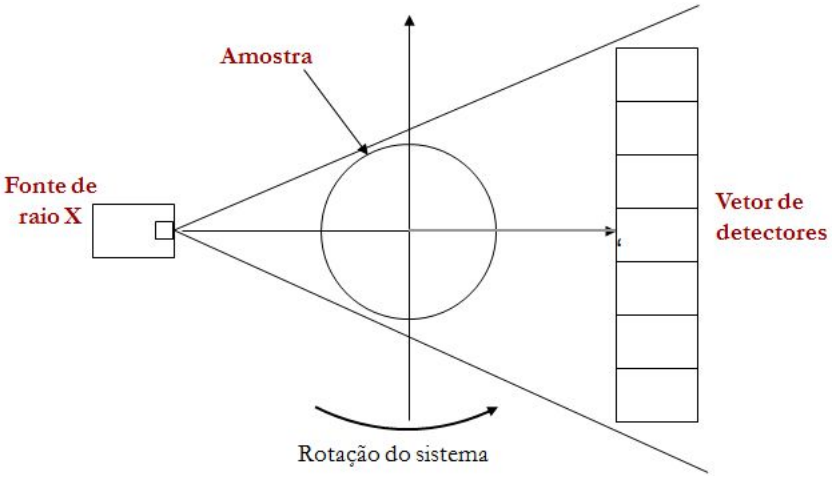


Figura 2.4: Representação do sistema tomográfico de 2^a geração

2.2.3 TC de Terceira Geração

A tomografia de terceira geração (figura 2.5)utiliza um feixe *fan-beam*, que é capaz de iluminar toda a amostra. Da mesma maneira que a tomografia de segunda geração, ela também pode usar um feixe cônico. A maior diferença deste sistema para o de segunda geração é a utilização de detectores bidimencionais ou seja uma matriz de detectores.

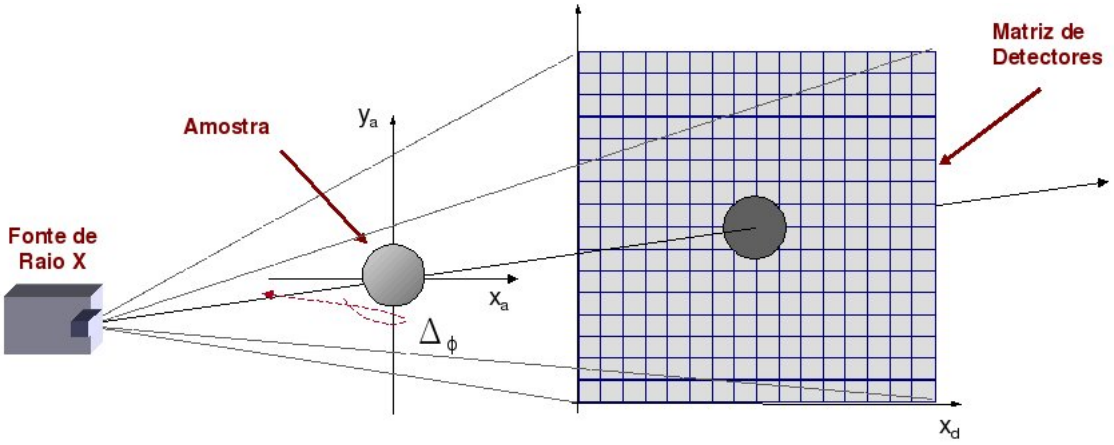


Figura 2.5: Representação do sistema tomográfico de 3^a geração

2.2.4 Fontes de raios-X

Os raios-X são produzidos quando elétrons liberados por uma fonte são acelerados em direção a um alvo metálico. Os tubos de raio-X podem ser divididos em dois tipos básicos conforme a produção de elétrons, são eles: tubos de gás, que produzem elétrons por ionização e tubos de filamentos, em que a fonte de elétron é um filamento aquecido.

No tubo de filamento (figura 2.6), os elétrons são liberados quando o anodo (filamento) é aquecido. Os elétrons emitidos, também chamados de termoelétrons, são ejetados quando a temperatura do anodo se torna muito alta. Esses elétrons são acelerados em direção ao alvo por um campo elétrico gerado por alta tensão. Ao interagir com os átomos do alvo os elétrons perdem parte de sua energia cinética. Esse tipo de radiação também é chamada de radiação de *bremsstrahlung* ou radiação de freiamento.

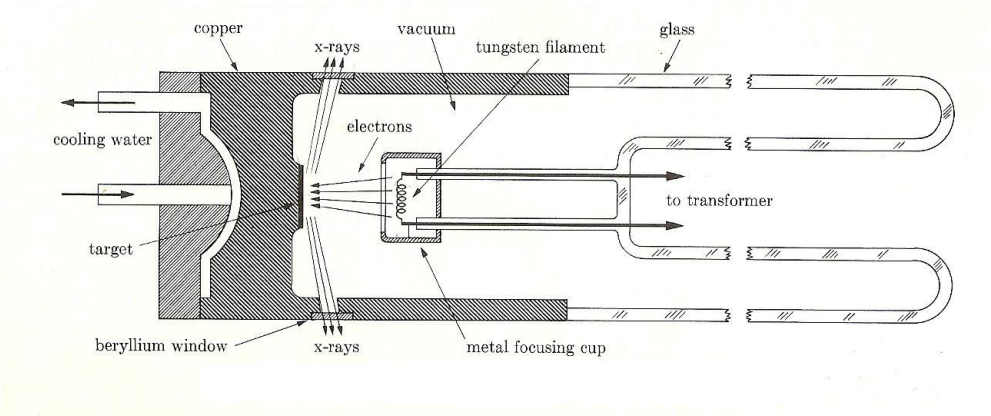


Figura 2.6: Esquema de um tubo de filamento de raio X. [9]

2.2.5 Luz síncrotron

Luz síncrotron é uma intensa radiação eletromagnética produzida por elétrons de alta energia num acelerador de partículas. A luz síncrotron abrange uma ampla faixa do espectro eletromagnético como raios X, luz ultravioleta e infravermelha, além da luz visível.

A luz síncrotron pode ser gerada pela aceleração de elétrons a velocidade próximas a da luz. A existência de campos magnéticos colocam os elétrons em caminhos circulares. Quando os elétrons são desviados de suas trajetórias eles liberam fótons. As radiações infravermelhas e de raios X são enviadas para as linhas de trabalho (chamados linha de luz) para as áreas onde os experimentos serão realizados. Os componentes necessários para a geração da luz síncrotron são: uma fonte de elétrons, um acelerador linear, um anel circular, dois anéis de restauração do feixe e linhas de luz.

Quando apontamos esta luz para uma pequena amostra podemos formar uma imagem que nos dá algumas propriedades da sua estrutura no detector. É, a partir desta imagem que se realiza a análise estrutural da amostra.

A caracterísctica mais importante da luz síncrotron é que o feixe de radiação emitido é monocromático, ou seja a intensidade do feixe é constante durante toda a sua utilização.

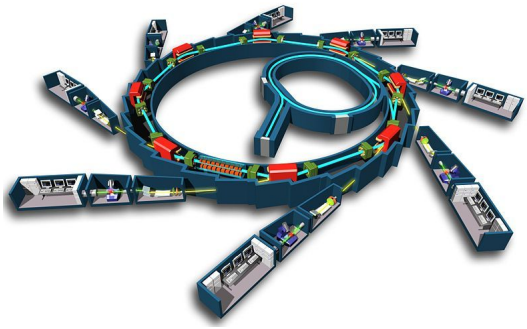


Figura 2.7: Esquema de geração de luz síncroton

Um esquema de geração de luz sincrotron pode ser visto na figura 2.7.

2.3 O método Monte Carlo

O método Monte Carlo é largamente utilizado na solução de problemas envolvendo processos estatísticos. É empregado na física médica devido à característica estocástica da emissão de radiação, transporte e detecção de processos radioativos. Este método é extrememente útil em problemas complexos que não podem ser modelados por códigos

computacionais que utilizam metodos determinísticos[10]. Outra grande aplicação do método Monte Carlo é feita experimentos onde os resultados são de difícil medição ou incluem grandes quantidades de radiação em um paciente.

A carga computacional do método Monte Carlo continua sendo o 'gargalo' para as simulações de modelagem de sistemas de imagens radiológicas, particularmente a TC de raio X [11], devido ao alto tempo de computação e a alta demanda de memória, já que para a solução do problema de transporte de energia é nescessário monitorar as trajetórias de vários fótons e seu respectivo histórico até sua detecção.

Mesmo assim o uso do método Monte Carlo (MC) para a simulação do transporte de radiação tornou-se bastante popular pois a alta demanda de processamento e de memória pode ser contornada se utilizarmos um sistema de processamento distribuído ou uma "grid", possibilitando diminuir o tempo de simulação mesmo se estivermos modelando geometrias complexas.

A simulação Monte Carlo é baseda em geração de números aleatórios de uma distribuição uniforme gerando variáveis aleatórias para estabelecer outras funções de distribição. Assim os números aleatórios ou pseudo-aleatórios são a base do método Monte Carlo, pois este método realiza todo o controle de decisão quando um evento tem uma certa possibilidade de ocorrer, tornando essencial a existência de uma subrotina de geração de número aleatórios para qualquer simulador baseado no método Monte Carlo. As subrotinas de geração de números aleatórios se baseiam em algoritmos matemáticos específicos para garantir que os números sorteados são realmente aletórios, pois estes números devem, em teoria, ser igualmente prováveis em um sorteio dentro de um intervalo definido.

2.3.1 Geração de números aleatórios

De maneira geral os algorítmos de geração de números aleatótios se baseiam em gerar números aleatórios (ϵ) uniformemente distribuídos entre 0 e 1. Existem vários algorítmos que realizam a geração desses números, os mais usados para o MC são a congruência multiplicativa e a congruência linear.

Os algorítmos que utilizam a congruência linear (equação 2.13) geram números que não são realmente aleatórios, ou sejam, não têm uma probabilidade de sorteio iguais para todos os números no intervalo. Esses algoítmos tem uma periodicidade de 10^{11} eventos [12]. Uma solução para melhorar esta periodicidade é calcular o logarítmo da congruência linear, porém esta operação acarreta em um maior tempo de sorteio do número.

$$X_{n+1} = mod(aX_n + c, 2^{32}) \tag{2.13}$$

Um outro método utilizado para a geração de números aleatórios é a congruência multiplicativa que gera números randômicos com uma periodiciade de 2^{30}.

$$X_n = 7^5 X_{n-1}(mod 2^{31} - 1), \qquad \epsilon_n = X_n/(2^{31} - 1) \tag{2.14}$$

Na verdade a congruência multiplicativa (equação 2.14) não gera números realmente aleatórios, o mais correto é chamarmos de números pseudo-aleatórios, pois este método utiliza um algoritmo determinístico para a determinação do número. Mesmo assim é muito improvável que a relação entre os valores da seqüência gerada influenciem nos resultados da simulação.

O gráfico da figura 2.8 mostra números sorteados utilizando a congruência linear 2.13, nele podemos ver que os números têm a tendência de ser sorteados em planos. Já o gráfico da figura 2.9 também mostra o sorteio de números utilizando a congruência, 2.14 multiplicativa. Comparamdo-se os gráficos destas duas figuras podemos ver que a distribiçcão de números sorteados utilizando a congruência multiplicativa é mais uniforme

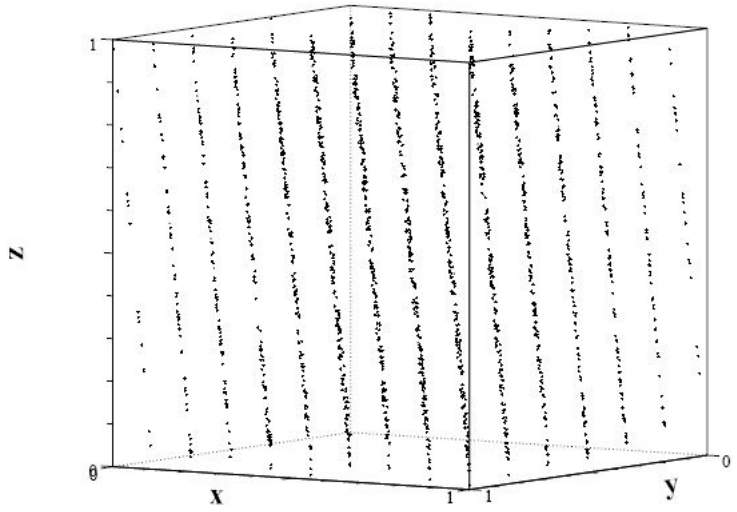


Figura 2.8: Representação em 3D de números aleatórios utilizando a congruência linear [13].

que a da congruência linear.

2.3.2 Método direto ou inverso

Seja uma distribuição de probabilidade típica (por exemplo a distribuição gaussiana) definida em um intervalo conhecido, digamos entre [a, b], ela deve ser integrável e não negativa. Assim, podemos definir a função de probabilidade acumulada desta distribuição como:

$$P(x) = \int_{x_{min}}^{x} p(x)dx \tag{2.15}$$

Criando uma nova variável aleatória através da transformação $\epsilon = P(x)$ e assumindo que a distribuição de probabilidade acumulada admite uma inversa, $P^{-1}(\epsilon)$, definida no intarvalo de (0,1), temos:

$$p_\epsilon(\epsilon) = p(x)\left(\frac{d\epsilon}{dx}\right)^{-1} = p(x)\left(\frac{dP(x)}{dx}\right)^{-1} \tag{2.16}$$

Assim fica claro que ϵ é um número aleatório, a variável x que é definida por $x =$

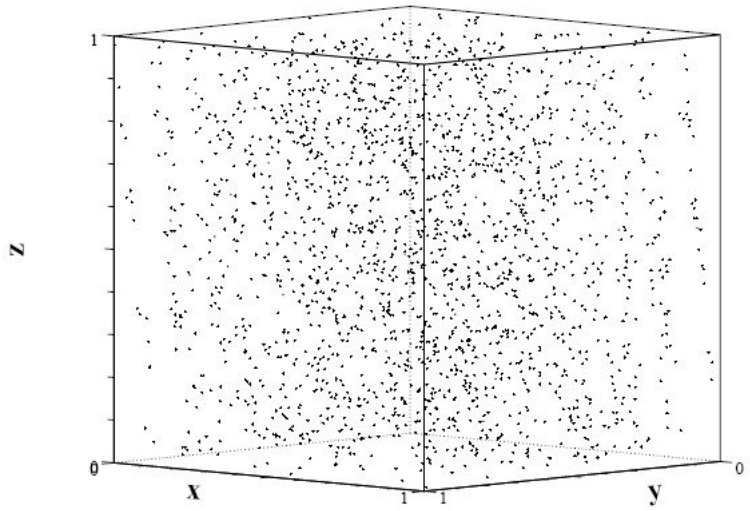


Figura 2.9: Representação em 3D de números aleatórios gerados utilizando a congruência multiplicativa [13].

$P^{-1}(\epsilon)$ é randomicamente distribuído no intervalo (x_{min}, x_{max}). Note que x é raiz única da equação

$$\epsilon = \int_{x_{min}}^{x} p(x)dx \tag{2.17}$$

A implementação do método direto de Monte Carlo pode ser feita através do seguinte algoritmo (figura 2.10):

1. Gerar um número aleatório ϵ;

2. verificar que x corresponde ao ϵ sorteado;

3. verificar a que probabilidade correspondente o x.

O simulador de TC utiliza este método para definir qual das interações irá ocorrer e também se a interação sorteada é o efeito *Rayleigh*, já que para simularmos este efeito temos uma tabela de dados para cada elemento químico que compõe a amostra.

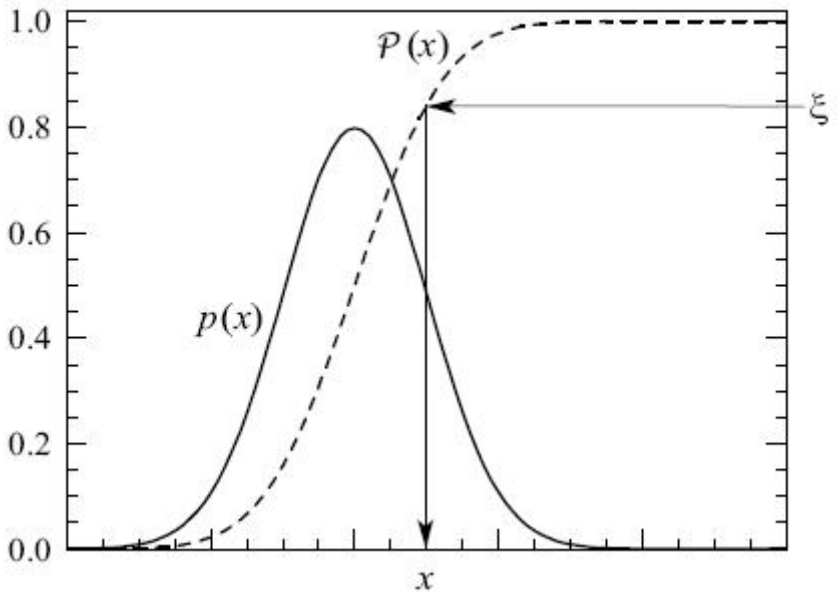


Figura 2.10: Sorteio de uma variável aleatória da districuição $p(x)$ usando o método inverso ou direto.

2.3.3 Método da rejeição

Este método é utilizado quando é muito complicado realizar o cálculo da função inversa da distribuição. Uma aplicação que ilustra bem este método é o cálculo do número π através de sorteios de números aleatórios. Para isso suponha um círculo inscrito dentro de um quadrado de lado unitário. Se sorteamos N eventos e contarmos quantos destes eventos estão dentro do círculo podemos calcular o valor de π.

Sabemos que a integral entre os intervalos a e b é igual a um quarto da área do circulo (ver figura 2.11). Desta maneira podemos dizer que o número de pontos sorteados dentro da área total do círculo dividido pelo número total de sorteios é igual a área do circulo sobre a área do quadrado.

$$\frac{N_c}{N_t} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4} \tag{2.18}$$

onde N_c é o número de sorteios dentro do cículo e N_t é o total de números sorteados. Obviamente quanto maior for o total de número sorteados mais preciso será o resultado.

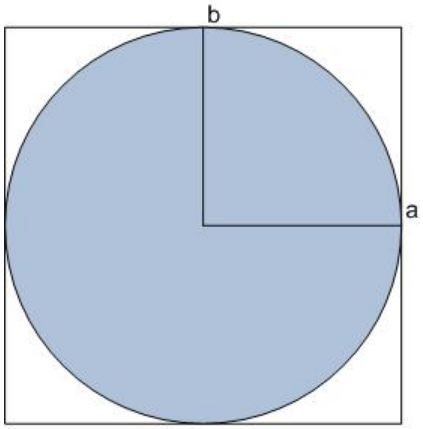


Figura 2.11: Exemplo do calculo do número π através do método direto de Monte Carlo

Em geral, o método da rejeição é utilizado para a estimativa de eventos que são regidos por distribuições contínuas, ou seja, aqueles eventos que obedecem uma curva de probabilidade como o cálculo da energia e a direção no espalhamento *Compton*.

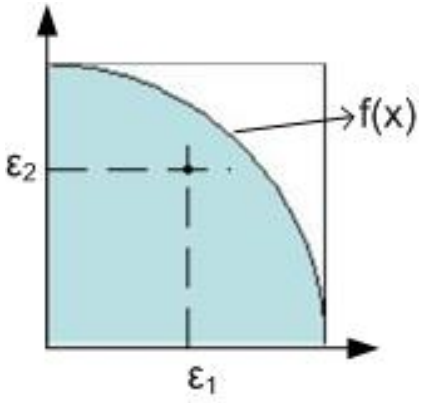


Figura 2.12: Processo de calculo do número π

É importante ressaltar que no método direto é necessário o sorteio de dois números aletórios. O primeiro (ϵ_1) para definir o valor da abcissa (x) e um segundo (ϵ_2) para definir o valor nos eixos das ordenadas (ver figura 2.12)

Se ϵ_2 for menor que $f(\epsilon_1)$ registramos ϵ_1 como sendo um valor válido. Caso contrário ϵ_1 é rejeitado e ocorre o sorteio de outro número aleatório.

O algoritmo do método da rejeição pode ser descrito como:

1. Sortear o primeiro número aleatório;

2. Verificar qual é o valor da função para esse número aleatório;

3. Sortear o segundo número aleatório;

4. Se $\epsilon_2 < f(\epsilon_1)$ registar ϵ_1 como válido;

5. Senão, rejeitar ϵ_1 e voltar ao passo 1.

Este método pode ser utilizado para a simulação do efeito *Compton*, pois os números sorteados obedecem uma função conhecida que rege o comportamento os fótons é chamada de equação de *Klein-Nishina* (equação 2.8).

Capítulo 3

Materiais e Métodos

3.1 Simulação dos fenômenos físicos

A simulação de um sistema de TC deve incorporar a simulação das interações da radiação com a matéria. Como citado anteriormente, os mecanismos de interação implementados são: efeito fotoelétrico, espalhamento *Compton* e espalhamento *Rayleigh*. A simulação do espalhamento *Rayleigh*, que obedece à lei de *Bragg* (equção 2.11) deve levar em consideração a translação do fóton dentro da amostra, o cálculo do ângulo de espalhamento e o cálculo da nova direção de propagação.

O algorítmo para realizar essa interação está exemplificado abaixo:

1. sortear um número aleatório
2. calcular o momento transferido
3. testar em qual quadrante se encontra o ângulo sorteado
4. calcular o ângulo de espalhamento
5. atualizar o vetor direção do fóton
6. transladar o fóton
7. voltar ao passo 1 (próximo fóton)

No caso da simulação do efeito *Compton* devemos simular a translação do fóton, calcular a seção de choque, calcular a direção e ângulo de espalhamento do fóton e também verificar o ângulo e a energia do elétron espalhado. Porém o elétron espalhado não está sendo simulado no estágio de desenvolvimento atual. Para simular o efeito *Compton* o algoritmo que está sendo utilizado é:

1. Sortear um número aleatório (ângulo de espalhamento)
2. calcular a energia de espalhamento
3. calcular a seção de choque em função do ângulo espalhado
4. calcular a nova energia do fóton
5. atualizar a direção do fóton
6. transladar o fóton
7. voltar ao passo 1 (próximo fóton)

Na simulação do efeito fotoelétrico não foi implementada a possível geração do elétron característico. Assim a simulação deste efeito está sendo, na verdade, um teste para verificar se o fóton é transmitdo ou se ele é absorvido pela amostra.

3.2 Metodologia

3.2.1 Arquivos de configuração

Todos os parâmetros de configuração do simulador como, posição da fonte e dos detectores, energia dos fótons, forma geométrica da amostra, composição da amostra, tamanho da matriz do detector e da amostra, número de fótons a serem emitidos, entre outros são passados através de arquivos de dados, que são arquivos texto em formato ASCII e em sua grande maioria são escritos pelo programador. Somente os arquivos

que contêm as secções de choque dos elementos utilizados são arquivos gerados pelos programas do NIST [14] e da IAEA [15].

A tabela 3.1, mostra um dos arquivos de configuração da amostra (*amostra.txt*). Este arquivo contém as informações de número de *voxels* da amostra (1ª linha), comprimento da matriz da amostra em centímetros (2ª linha), nome do arquivo que contém a forma geométrica da amostra (3ª linha) e nome do arquivo que contém a lista de elementos que compõem a amostra (4ª linha, tabela 3.1). O comprimento da matriz é informado em centímetros, pois as informações de seção de choque nos arquivos gerados pelo NIST ou pela a IAEA são em centímetros.

Tabela 3.1: Arquivo de configuração da amostra (*amostra.txt*).

501	501	501
5	5	5
Dados.txt		
Composicao.txt		

Este arquivo informa as características gerais da amostra para o simulador, sendo o arquivo *Dados.txt* responsável pela organização da forma geométrica da amostra conforme a tabela 3.2.

Tabela 3.2: Arquivo *Dados.txt*.

cx	1	250	250	200	200	0	450
#cilindro oco							
p	2	100	150	100	150	150	150
e	3	250	250	250	200	200	200

Na primeira linha os caracteres 'cx' informam ao simulador que a figura a ser formada é um cilindro com eixo de simetria na direção 'x'. Nesta linha podemos trocar o 'x' por 'y' ou 'z', dependendo de onde queira se colocar o eixo de simetria do cilindro da amostra

a ser criada. Ainda na 1ª linha da tabela 3.2, o número 1 representa o índice do elemento correspondente, caso este índice seja 0 (zero), o simulador interpreta esse valor como sendo vazio. Os demais números são: centro no eixo x, centro no eixo y, semi-eixo em x, semi-eixo em y, voxel inicial e voxel final, respectivamente.

A 2ª linha da tabela 3.2 é um exemplo de linha de comentário, esta linha será ignorada pelo o simulador, servindo apenas de orientação ao usuário. Já a 3ª linha é a configuração de um paralelepípedo, (note que a linha começa com o caractere 'p') que deve conter as seguintes informações após o segundo caractere, ponto inicial nos três eixos (x,y,z) e o comprimento de cada lado.

Na última linha temos a configuração de uma esfera (a linha começando com o caractere 'e') que é feita de seguinte forma: centro (x,y,z) e os diâmetros nas três direções. Isto permite criar-los com diferentes raios, formando um cilindro de base elíptica ou, no caso de uma esfera, um elipsóide

Note que as configurações de todas as figuras geométricas são feitas utilizando oito parâmetros. Caso alguma linha válida para a configuração tenha menos ou mais de oito parâmetros o simulador acusa esse erro e a simulação é abortada.

O segundo arquivo que aparece na configuração da amostra é o arquivo que contém a composição da amostra. Ele informa ao simulador que elemento corresponde cada índice do arquivo *Dados.txt* e é organizado como mostra a tabela 3.3[1]

Tabela 3.3: Arquivo *Composicao.txt*.

s	1	2.7	cAluminio.xcom	mAluminio.txt
s	2	1.85	cHAp.xcom	mHAp.txt
s	3	1.85	cGordura.xmu	mGordura.txt
m	1	2	0.40	0.60

[1]A substância gordura é considerada um elemento simples para efeito de simulação baseado nos dados apresentados pelo programa XmuDAT (IAEA).

O primeiro caractere, que deve ser 's' ou 'm', informa se o voxel conterá um elemento (substância simples - s) ou mais (subtância composta - m). No caso do *voxel* conter uma substância simples o próximo caracter deve ser o índice que irá corresponder à substância. O terceiro caractere é a densidade do elemento em g/cm^3. Os últimos dois parâmetros são os arquivos que contém as seções de choque e o momento transferido do elemento.

Se o voxel for um elemento composto (ou seja, contiver mais de um elemento em um único *voxel*) o segundo e o terceiro caracteres (no exemplo da tabela 3.3, os números 1 e 2, respectivamente) indicam que o voxel contém tanto alumínio como hidroxiapatita em uma concentração de 40% de alumínio e 60% de hidroxiapatita (os dois últimos caracteres da última linha). A diferença de extensão na configuração da gordura e da hidroxiapatita ocorre pois são gerados por diferentes programas.

Ainda em relação aos arquivos de configuração da amostra, temos o arquivo de configuração do detector (tabela 3.4). Este arquivo possiu os dados sobre a posição do detector em relação ao centro da amostra (1ª linha), direção de detecção (2ª linha), definição de tamanho do detector em y e z (3ª e 4ª linha) e finalmente o tamanho da matriz de detecção. Nesta última linha é a responsável pelo o tamanho da imagem (resolução) de cada projeção.

Tabela 3.4: Arquivo de configuração do detector.

50.0	0.0	0.0
-1.0	0.0	0.0
0.0	0.0	1.0
0.1	30.0	30.0
128	128	

A configuração da fonte é feita através do arquivo *fonte.txt*. Este arquivo define a posição da fonte (1ª linha), a direção de emissão dos fótons (2ª linha), a abertura de emissão e o foco (3ª e 4ª linha). Na tabela 3.5 temos a apresentação de um arquivo de

configuração da fonte. A última linha é o arquivo que contém a energia dos fótons que serão emitidos e a probabiblidade de emissão com esta energia.

Tabela 3.5: Arquivo de configuração da fonte.

-50.0	0.0	0.0
1.0	0.0	0.0
0.05		
0.420		
Spe1.txt		

Todas as definições de posição são feitas em relação ao centro da matriz da amostra. É por este motivo que aparece o sinal negativo no primeiro número da primeira linha do arquivo *fonte.txt*.

Por último temos o arquivo *Sistema.txt*, representado na tabela 3.6. Nele defiminos o número de projeções (passos angulares) na 1ª linha, número de raio-soma na 2ª linha e número de eventos por raio-soma na 3ª linha.

Tabela 3.6: Arquivo de configuração do sistema.

360
1
1e8

3.2.2 Simulador de TC

No atual estágio de desenvolvimento o simulador encontra-se funcional, ou seja, ele é capaz de simular amostras de diferentes formatos geométricos, com diferentes substâncias e com os mecanismos de interação da radiação com a matéria mais relevantes para a TC, porém foi dada uma maior ênfase a mecânica do simulador em um sistema de programação paralela. Assim os detectores do simulador, como as fontes, não possuem uma curva de

eficiência e a confecção da amostra ainda é muito básica. No caso específico do detector, não estamos simulando a interação da radiação com o mesmo, sendo que a contagem no detector é realizada simplesmente verificando se o fóton incide na superfície de detecção.

Para um melhor desenvolvimento computacional o simulador foi desmembrado em 3 grandes partes: fonte e detectores, amostra e interação da radiação com a matéria. Com o objetivo de deixar o código mais simples, o simulador foi escrito em arquivos diferentes. O arquivo principal (*simulacao.cpp*), que controla o funcionamento geral do sistema e um arquivo que é o reponsável pela configuração e definições do simulador (*simulacao.cls.h*) que contém os vários métodos necessários para a computação da simulação.

Os passos gerais do simulador estão representados na figura 3.1 que também representa a distribuição do simulador pelos os processadores de um *cluster*.

Após a configuração inicial do cluster o simulador cria e envia os '*chares*' para o número de processadores escravos pedidos para o processamento da tomografia e o último passo é o armazenamento em disco dos resultados da projeção. Depois de armazenar a projeção o simulador repete o *loop* de distribuição dos '*chares*' realizando o processamento e armazenamento. Este *loop* acontece até o número de projeções seja igual ao definido no início do processo de simulação.

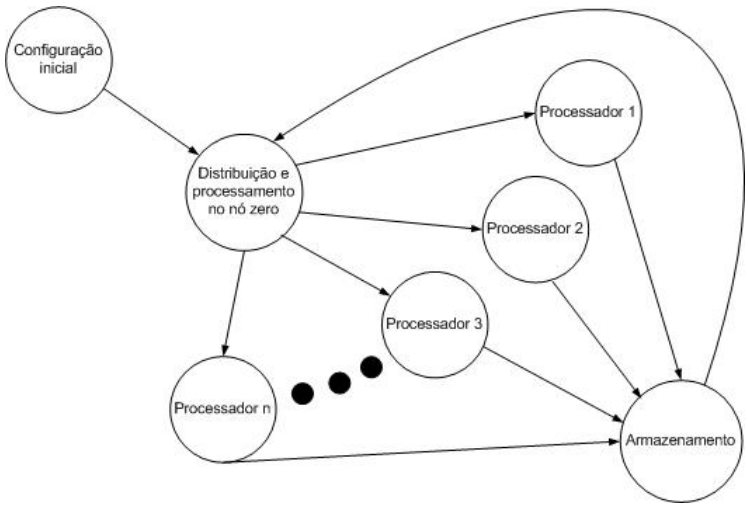


Figura 3.1: "Máquina"de estado do simulador de TC

As trocas de mensagens entre as instâncias do simulador são mostradas na figura 3.2. Nela podemos ver que após a configuração inicial (passo 1 da figura 3.1) o simulador chama a instância *'projecao'* para dar início à distribuição e ao processamento da simulação (passo 2 da figura 3.1). Depois de realizar essa chamada, é enviada uma mensagem para a instância *interacao*. Essa mensagem é enviada para cada voxel da amostra e para todos os fótons configurados (passo 3 da figura 3.1). Caso seja o último fóton, a instância *'deteccao'* chama a instância *'reducao'* que é a responsável pelo o armazenamento dos dados simulados (passo 4 da figura 3.1).

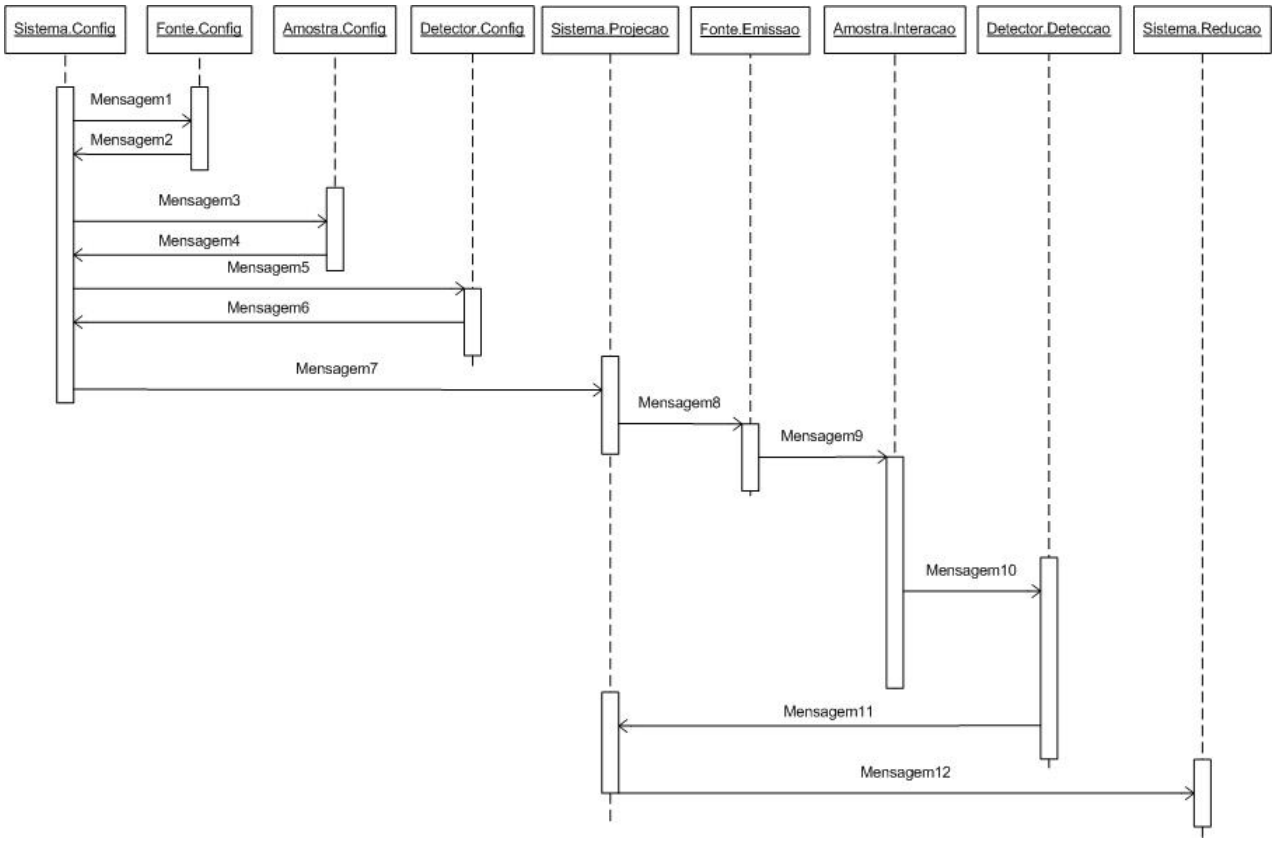


Figura 3.2: Trocas de mensagens do simulador

A Figura 3.3 representa o fluxograma do arquivo principal do simulador, ou seja ela dá uma visão geral do funcionamento do simulador. O primeiro passo é a geração e emissão de um fóton. Após a emissão deste fóton, é realizado um teste para verificar se ele colide ou não com a amostra. Em caso de colisão o simulador lê o índice do elemento da amostra

e pergunta se o elemento é vazio ou não. Sendo elemento vazio atualizamos a posição o fóton e testamos se ele saiu da amostra. Se o elemento não for vazio o simulador procura os dados da substância correspondente ao índice e calcula as probabilidades de interação.

Já com as probabilidades na memória o simulador sorteia a probabilidade de acontecer o efeito fotoelétrico, efeito Compton e o efeito Rayleigh. É importante ressaltar que se a interação da radiação com a matéria for, por exemplo, efeito fotoelétrico os cálculos das probabilidades de interação de Compton e Rayleigh não serão realizados.

Caso ocorra efeito fotoelétrico, o fóton é absorvido e acontece a emissão de um novo fóton, ou seja, o fóton atual é descartado fazendo com que o simulador de TC sorteie um novo fóton. Se a interação sorteda for efeito Compton, devemos calcular a nova energia e a nova direção de propagação do fóton de absorção, sendo efeito Rayleigh só devemos calcular a nova direção de propagação do fóton.

O algoritmo de emissão do fóton é capaz de calcular, a partir do foco definido no arquivo de configuração (tabela 3.5), uma direção aleatória de emissão dentro deste foco. Para isso sorteamos um ponto dentro do foco estabelecido, definimos uma direção de emissão e rotacionamos este vetor direção de um ângulo aleatório multiplo de 2π. Este algorítmo também imprime na tela as configurações do fóton que são a direção, posição inicial e energia.

Já o procedimento de inicialização da fonte é feito através da leitura do arquivo de configuração correspondente. Este procedimento aloca o vetor de energia e probabilidade e calcula a direção de emissão. Feito isso o fóton está pronto para ser emitido.

O código que trata do detector (ver figura 3.5), além de posicioná-lo conforme o arquivo de configuração, monitora o fóton quando este sai da amostra verificando se o fóton colide com a face de detecção do mesmo. Caso o fóton colida com qualquer outra superfície do detector ele não será contado. Porém, o fóton não é transladado até atingir o detector, o simulador verifica se a direção de propagação do fóton irá interceptar a face

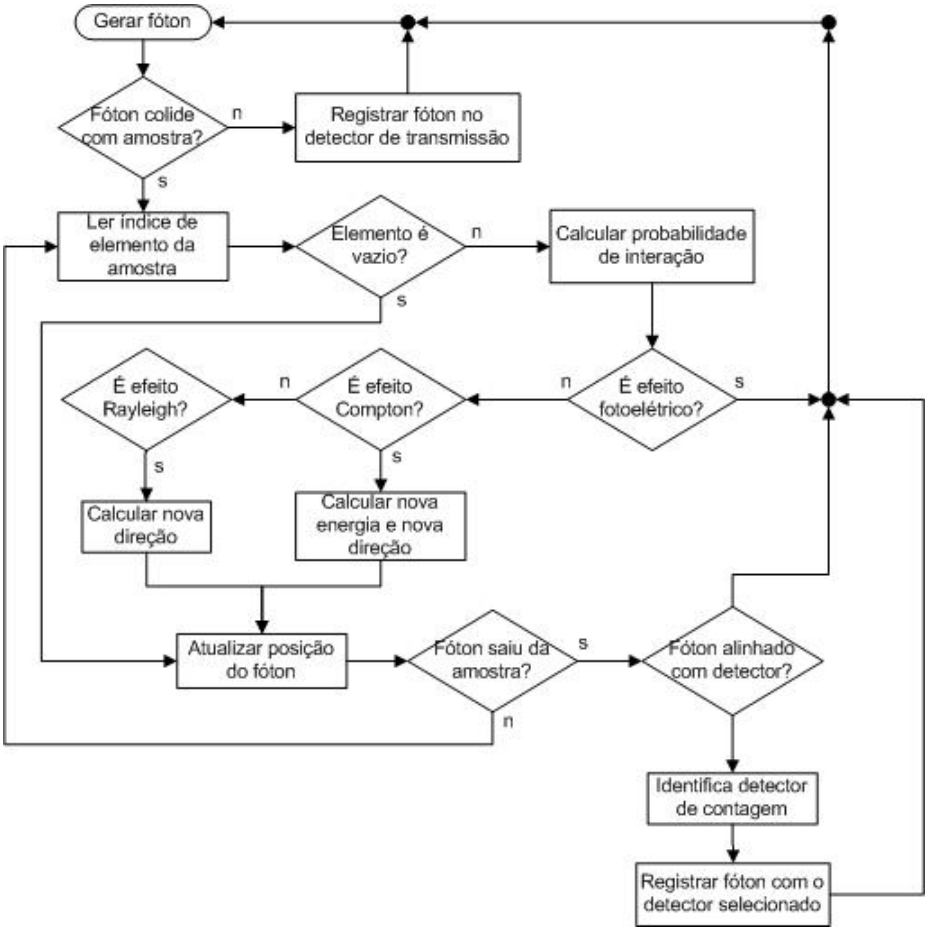


Figura 3.3: Fluxograma geral do simulador.

fonte
-pos : double -dir : double -tpasso : double -rpasso : double -foco : double -abertura : double -*ene : double -*prb : double -iRAND_MAX : double -tam : int
+Pos() : void +Dir() : void +PassoDeTranslacao() : double +PassoDeTranslacao() : double +Foco() : double +Rotacao() : double +Emissao() : double +Dados() : void +Config() : void

Figura 3.4: Diagrama UML da classe fóton

de detecção do detector.

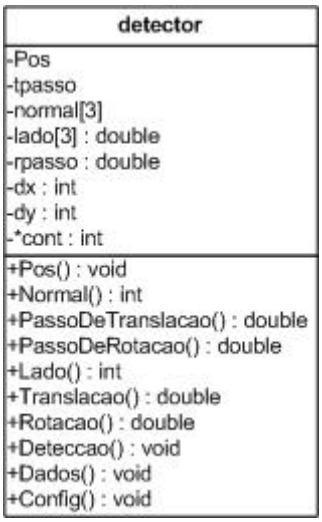


Figura 3.5: Diagrama UML da classe detector

O próximo passo do simulador é a realização da configuração da amostra após a leitura do arquivo *Amostra.txt* através da classe *amostra* (figura 3.6), o simulador configura a matriz da amostra com as características de tamanho e quantidade de pixels, realiza a leitura do arquivo de dados e desenha a amostra de acordo com o arquivo Dados.txt. Se a mostra a ser simulada é um cilindro oco devemos escrever duas linha no arquivo de configuração. O simulador desenha a primeira linha e depois a segunda, sobrescrevendo a configuração anterior quando é o caso. Portanto se o arquivo tem as seguintes linhas *cx 1 250 250 200 200 0 450* e *cx 0 250 250 150 150 0 450* o simulador irá desenhar a primeira linha com o índice 1 e depois irá escrever a segunda linha, sobrescrevendo o interior do primeiro cilindro, formando um cilindro oco. É importante ressaltar que os atributos responsáveis pelo tamanho da matriz da amostra (dx, dy e dz) são inteiros. Isto ocorre devido ao número de *pixels* ser uma variável discreta.

Na figura 3.7 temos as relações entre cada classe do simulador. Através dela podemos ver que a classe Sistema se relaciona com todas as classes do simulador, realizando o papel de gerência. Os pontos que aparecem da figura informa que é possível ter várias classes amostra interagindo com várias classes fóton, poe exemplo.

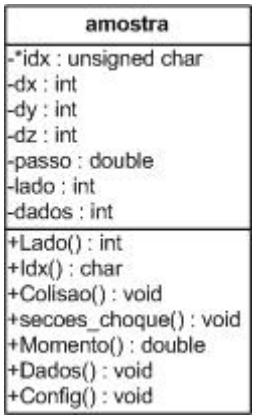


Figura 3.6: Diagrama UML da classe Amostra

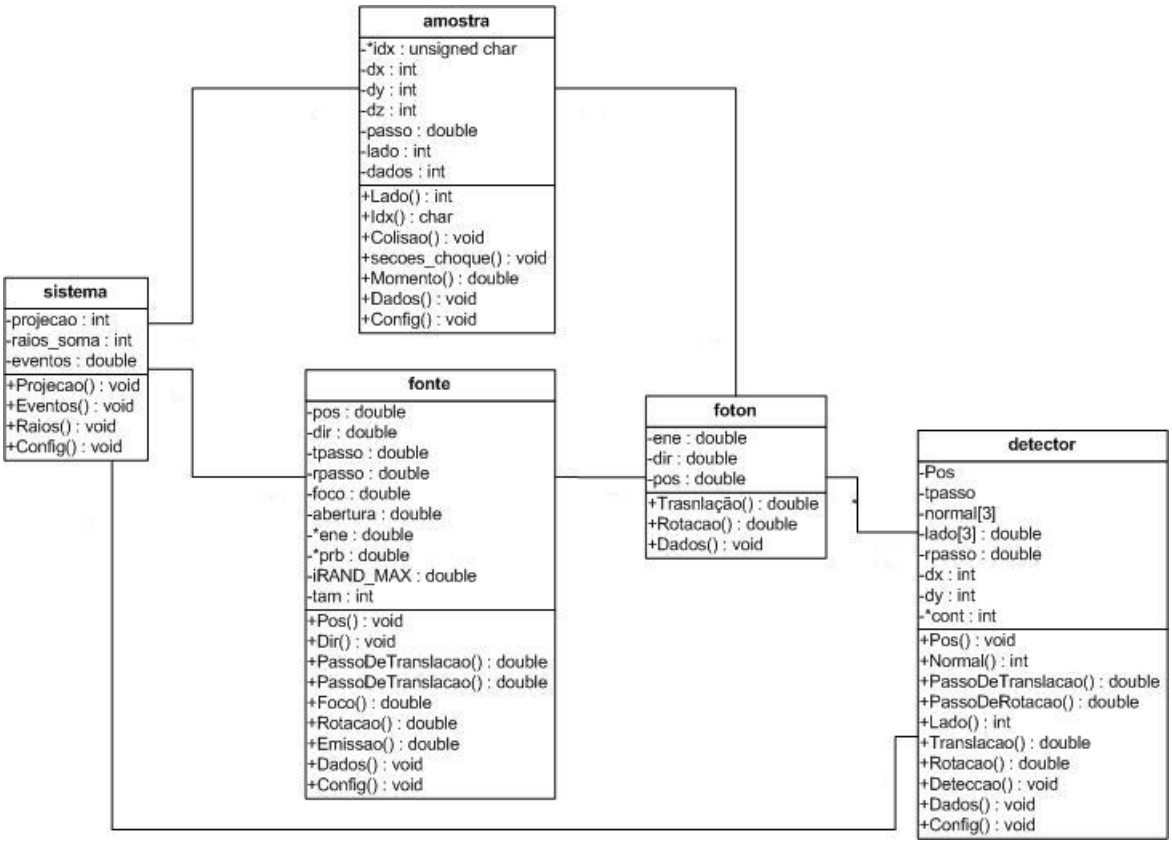


Figura 3.7: Diagrama UML das relações ebtre as classes do simulador

3.3 Materiais

Um cluster é basicamente um conjunto de computadores interligados por rede que podem trabalhar em conjunto para a realização de cálculos científicos, simulações, previsão

climática, entre outras aplicações que requerem grande tempo computacional quando comparadas com a performance de um único computador (*stand alone*) ou para aplicações que rodam em sistemas distribuídos, ou seja, em computadores que podem não estar fisicamente na mesma sala e até mesmo no mesmo país.

Porém, um cluster não é somente um conjunto de computadores ligados em rede através de um *hub* ou *switch* como seria uma rede de computadores em uma empresa. Para termos um cluster, devemos nos preocupar com o tempo de comunicação entre os computadores, a disponibilidade dos computadores para os usuários, a capacidade de armazenamento dos dados gerados, o gerenciamento dos trabalhos a serem processados (fila de *jobs*) e as ferramentas que os usuários irão utilizar (editores, compiladores, e outros). Além disso, deve-se dar grande atenção à segurança de todo o sistema.

Todo o cluster tem, no mínimo, um computador que realiza a gerência de todo o sistema do cluster como a fila de *jobs* e a disponibilidade dos computadores incluindo computadores específicos para determinado grupo de pesquisa. Este computador é denominado *nó mestre* ou simplesmente *mestre*, enquanto os computadores destinados unicamente para processamento são chamados *nós escravos* ou *nós*.

3.3.1 Infra-estrutura para processamento distribuído

Sistema de comunicação

Como dito anteriormente, um *HUB* ou um *switch* não podem garantir que uma rede de computadores forme um cluster. Como estamos querendo aumentar o poder de processamento para tornarmos as aplicações mais rápidas, devemos, entre outras coisas, diminuir o tempo de comunicação entre os computadores designados para o processamento, já que podemos utilizar vários nós para resolver o mesmo problema. Se a comunicação entre os computadores (mestre-escravos e/ou escravos-escravo) for lenta podemos perder performance e ter até um tempo computacional maior que em um único computador.

Atualmente as interfaces de redes que são utilizadas para a comunicação entre todos os computadores de um *cluster* são basedas nos seguintes padrões: *Gigabit Ethernet*, *Myrinet* e *Infiniband* [16].

- *Gigabit Ethernet*: é um dos padrões mais populares por utilizar a tecnologia *ethernet* e por seu baixo custo. O gigabit ethernet é uma evolução dos padrões ethernet de 10/100 Mbps (mega-bytes por segundo) pois ele realiza a transmissão de dados até 1000 Mbps. Este padrão foi desenvolvido através da necessidade criada pelo aumento da largura de banda entre o nó mestre e um único nó do cluster.

- *Myrinet*: é baseado na tecnologia usada para comunicação e troca de pacotes entre processadores que trabalham em paralelo. A interconexão de um sistema Myrinet deverá ser capaz de detectar e isolar falhas de comunicação e de utilizar percursos alternativos, pois as interfaces de *host* mapeiam redes, selecionam rotas, controlam o tráfico de pacotes e transformam endereços de rede em rotas.

  O que faz com que o padrão Myrinet seja classificado como de alta performance é a distribuição da demanda computacional entre os nós do cluster para sistemas de alta disponibiliade, as altas taxas de transmissão de dados, as baixas taxas de erro e o controle de fluxo em todos os links.

- *Infiniband*: utiliza uma estrutura hierárquica com comunicação do tipo ponto-a-ponto. Isto permite que qualquer nó possa iniciar de um canal de comunicação direto para qualquer outro nó e ainda permite que vários dispositivos de entradas/saídas (E/S) peçam dados simultaneamente ao processador, pois o *infiniband* destina-se a fornecer aos centros de dados uma conectividade para E/S melhoradas e adaptadas a qualquer tipo de tráfego.

  As principais vantagens do *Infiniband* são a alta velocidade de comunicação (20Gbps de conexão com o host e 60Gbps de *switch* para *switch*), baixa latência e

alto nível de integridade de dados por verificação cíclica de redundância. Essas características indicam que o InfiniBand é uma tecnologia adequada para aplicações de HPC (*High Performance Computing*), porém é um padrão em evolução e deve sofrer muitas melhorias como a inclusão de protocolos de qualidade de serviço (QoS), entre outras.

Nós para processamento

Uma das razões principais para se utilizar um *cluster* de computadores é realizar tarefas computacionais de forma que o tempo de processamento seja menor. Para isso, o número de nós disponíveis no *cluster* para processamento influencia diretamente nesse tempo, portanto é natural pensar que quanto maior o número de processadores utilizados por um *job* menor será o tempo de processamento.

Porém, mesmo tendo um sistema de comunicação muito rápido, isso não é necessariamente uma verdade, pois o *job* pode não utilizar a mesma carga de processamento para todos os nós, já que podem existir nós com diferentes poder de processamento em um mesmo cluster. Neste caso, deve-se conhecer o *hardware* dos nós e mandar somente a carga de trabalho que não irá deteriorar o tempo de computação. Uma outra saída é a utilização de ferramentas de balanço de carga garantindo a utilização máxima de cada processador. O ideal para um *job* em um cluster é utilizar todo o poder de processamento do processador durante o tempo de processamento deste *job*.

Em quase todos os sistemas de alto desempenho, existe alguma limitação para os usuários. Estas limitações podem ser, o número máximo de processadores por *job* ou tempo máximo de processamento por *job*. Também é possivel que estas limitações sejam aplicadas aos usuários.

Sistema de gerenciamento de filas

Para gerenciar todas as limitações, existem vários sistemas de gerenciamento de filas.

Esses sistemas são programas que verificam, entre outras coisas, o número de nós pedidos pelo *job* e o número de processadores disponíveis. Caso o *job* requisite um número de nós maior que o disponível o, *job* fica na fila a espera da liberação de processadores necessários para rodar o *job*. É importante resaltar que um *job* na fila de espera não reserva os nós disponíveis. Caso um segundo *job* entre na fila de espera e necesseite de um número menor de processadores que o anterior e este número de nós esteja liberado, o segundo *job* começa a ser executado antes do primeiro.

Os programas de gerenciamente de filas podem definir prioridade para cada usuário. Quanto maior for a sua prioricade menos tempo o seu *job* ficará na fila esperando a execução. Ou seja, caso existam dois jobs na fila para a execução e eles pedirem o mesmo número de processadores, o que tiver a prioridade mais alta irá rodar primeiro.

Os gerenciadores de filas mais comuns são: OpenPBS, Torke, SGE e Condor.

Existe uma pequena diferença entre o Condor e os demias citados. O OpenBPS, Torke e o SGE são gerenciadores de filas para *clusters* enquanto o Condor é um gerenciador para grids ou programação distribuída, permitindo inclusive que o *job* migre de um computador para outro mesmo quando esses computadores estejam em locais distintos.

O Torque, OpenPBS e o SGE possuem os mesmos comandos para submissão e monitoramento de *job*. As únicas diferenças entre eles são os *scripts* de submissão de *job* (ver tabelas 3.7 e 3.8).

Memória RAM

Uma das limitações da simulação Monte Carlo inclui a grande demanda de memória RAM do computador [17]. Caso um *job* necessite ocupar um espaço maior que a disponível na RAM do nó de processamento ele irá realizar *swap* em disco (transferência de dados da RAM para os discos rígidos). É comum encontrar *cluster* que não contenha discos rígidos nos nós e sim um servidor de discos, neste caso o nó irá realizar *swap* em disco

Tabela 3.7: Exemplo de *script* de submissão do OpenPBS e Torke

```
# nome do job
# PBS -N simulacao
# numero de nos e cpus (sendo serial: 1 1)
# PBS -l nodes=1:ppn=1
# junta erros e output
# PBS -j oe
# nome do arquivo de saida
# PBS -o saida_simulacao_e7.txt
# Posiciona o ponteiro na area do usuario
cd $ PBS_O_WORKDIR
rm nodelist.tmp
echo "Job started at ‘date‘"
echo "group main»> nodelist.tmp
for i in $(cat $PBS_NODEFILE); do
      echo "host $i»> nodelist.tmp
done;
./charmrun ++remote-shell ssh ++nodelist nodelist.tmp ++p 1 ./simulacao
echo "Job ended at ‘date‘"
exit 0
```

Tabela 3.8: Exemplo de um arquivo de submissão do SGE

```
#$ -N simulacao
#$ -pe mpi 16
#$ -o /home/u/tomodif/simulacao/saida_simulacao.txt
#$ -e /home/u/tomodif/simulacao/error.txt
cd /home/u/tomodif/simulacao
echo "Job started at ‘date‘ "
./charmrun ++remote-shell ssh ++nodelist nodelist +p16 /home/u/tomodif/simulacao/simulacao
echo "Job ended at ‘date‘ "
```

via servidor de arquivos, aumentando sensivelmente a transferência de dados pela rede, podendo causar uma grande latência e no pior caso pode causar a terminação brusca do *job*. De qualquer maneira, um *job* que realiza *swap* em disco terá um tempo de processamento muito maior, pois a transferência de dados para o disco rígido é muito mais lenta do que a transferência para a memória RAM.

3.3.2 Equipamentos utilizados

O projeto de simulação foi realizado em três configurações diferentes. A primeira delas foi realizada no *cluster* Ssolar I do CBPF que é um *cluster* composto de 20 computadores, sendo um *master* e 19 nós clientes. Todos os nós clientes possuem dois processadores (AMD Athlon MP 1800+), formando assim um *cluster* com 38 processadores disponíveis para processamento dos *jobs*, 2 Gb de memória RAM para cada nó, discos rígidos com capacidade de 40 Gb. Os computadores estão conectados por rede *Ethernet Gigabit* formando um *cluster* do tipo *Beowulf* (*clusters* baseados em computadores pessoais interligados em sistema de rede privada).

Figura 3.8: Foto de um dos nós do cluster Ssolar I do CBPF.

A segunda configuração utilizada foi a do CESUP (Centro Nacional de Supercomputação) que tem um *cluster* composto por um nó de gerência, cinco nós para processamento, com processadores AMD Opteron quad core de 1.8 GHz, sendo que cada nó possui dois processadores totalizando 20 processadores. Neste *cluster* não existe limitação para o usuário, pois o CESUP optou por uma política de compartilhamento de processadores. Isto quer dizer que um processador pode executar simultâneamente mais de um processo,

o que pode causar uma queda de performance.

A terceira configuração usada não foi realmente um *cluster*. Foram utilizados oito computadores pertencentes à rede do laboratorio de física computacional da UERJ que estão ligados através de uma rede extremamente lenta. Porém foi neste cenário que vários testes foram realizados, o que contribuiu para um melhor entendimento do código do simulador e funcionamento do CHARM++. São computadores com processadores Pentium 4, com 1 Gb de memória RAM. É importante ressaltar que neste caso não existem nós de processamento nem nó master pois, são apenas computadores ligados em rede através de um *switch*.

3.3.3 Suporte de programação e gerenciamento utilizados

O simulador de TC foi totalmente desenvolvido em CHARM++ que é uma linguagem de programação orientada a objeto baseada em C++ para sistemas de processamento em paralelo. É basicamente uma fusão do AMPI com C++, permite a passagem de mensagem sem a necessidade de uma linha explícita de código dizendo quando e para quem mandar uma mensagem, como seria no MPI. Porém o CHARM++ permite que o programador faça isso, o que o torna ainda mais interessante.

A programação em CHARM++ é muito parecida com a programação em C++, sendo 'fácil' a migração de um programa C++ para CHARM++, basta acrescentar os métodos internos do CHARM++ para troca de mesagem e 'suporte' para ponteiros. Na prática os ponteiros em CHARM++ funcionam da mesma maneira que em C++, só que como estamos em um ambiente paralelo e os endereços de memória são distindos de um micro para o outro, para contornar isso o CHARM++ copia os conteúdos dos ponteiros de um micro e envia como mensagem. O método responsável por esta operação é chamado *PUPable*, o programador deve colocar neste método os parâmetros que devem ser passados aos outros computadores. Abaixo temos um exemplo do método PUPable, neste exemplo

estamos informando que o atributo x, y e z da classe xyz será enviado como mensagem.

```
PUPable_decl(xyz);
xyz(CkMigrateMessage *m) : PUP::able(m) {}
virtual void pup(PUP::er &p) {
PUP::able::pup(p);
p|x; p|y; p|z;
}
```

Um programa escrito em CHARM++ começa criando uma instância simples em cada processo no processador 0, chamando o método construtor desses *chares* (objetos concorrentes). Estes *chares* podem criar outros *chares* em outros processadores rodando simultaneamente.

Os *chares* podem ser organizados em diferentes categorias como *chare-array*, *chare-group* e *chare-nodegroup*. A categoria *chare-array* é um vetor de chares indexados, na prática é um vetor de processos concorrentes. O *chare-array* envia para os processadores o número de processos que foi definido pelo programador para todos os processadores. Por exemplo, se rodarmos uma simulação com 10 processadores e pedirmos para criar 100 processos, o *chare-array* irá criar os 100 processos e enviar 10 processos para cada processador, que ficam na fila esperando para serem executados.

Caso algum desses *chares* fiquem muito tempo na fila de algum processador pode ocorrer erro de *time-out* ocasionando a interrupção do programa. Esta categoria é apropriada para chares de rápida execução ou para sistemas com alta eficiência de comunicação entre os nós, já que não existe nenhuma ferramenta nativa do CHARM++ para recuperar a parte do código que já foi executada.

O *chare-group* é um grupo de chares criado pelo o processador 0. Ele cria todos os *chares* no processador 0 e envia apenas um para cada processador disponível deixando a fila dos processsadores livres, isto inclui processadores com mais de um núcleo já que o

CHARM++ os identifica como dois processadores. Assim, se rodarmos um programa que cria 100 *chares* em 20 processadores o *mainchare* irá mandar um para cada processador e somente após o término de um *chare* é que um outro *chare* é enviado para aquele processador que já terminou a tarefa.

Já o *chare-nodegroup* é uma espécie de *chare-group* porém ele é mais indicado para computadores com mais de um processador, pois permite o compartilhamento da RAM e a otimização de comunicação de um *chare* com outro, quandos eles estão rodando dentro do mesmo computador, só que em processadores diferentes[18].

Capítulo 4

Resultados

O simulador desenvolvido foi avaliado de duas maneiras. A primeira por meio de alguns experimentos para verificar à fidelidade dos dados simulados e a segunda em relação ao tempo de simulação.

O primeiro experimento realizado teve como objetivo avaliar o programa de reconstrução. Este experimento consiste em realizarmos a reconstrução de uma simulação de *background*. Já o segundo experimento foi para avaliarmos se a interação da radiação com a matéria está sendo simulada corretamente através da simulação e reconstrução de uma amostra em forma de escalímetro. Quanto ao posicionamento dos objetos na amostra, foi realizado um terceiro experimento com o objetivo de verificar se há alguma distorção em função da posição do objeto dentro da matriz da amostra. O quarto experimento realizado foi feito utilizando-se dados do projeto *Visible Human*[19]. Também foram realizados outros dois experimentos, um deles é um quadrado e o outro em forma de 'grade', estes dois tem como objetivo verificar se a reconstrução dos limites da amostra está coerente.

Todas as amostras dos experimentos foram configuradas com 5 cm de lado e com 500 *voxels*, resultando em um *voxel* de 0,01 cm de tamanho com exceção do experimento de posicionamento, que foi realizado com uma amostra de 10 cm de lado e com 500 voxels,

ou seja um *voxel* de 0,02 cm. Já para o experimento do projeto *Visible Human*, que neste caso a matriz da amostra tem 20 cm de tamanho da imagem, e com resolução de 370 x 422 *pixels*.

4.1 Simulação de objetos básicos

4.1.1 Ruído da reconstrução

Este experimento tem como objetivo verificar se a reconstrução da amostra simulada está inserindo algum tipo de anomalia nas imagens reconstruídas. Para isso foi realizado uma tomografia de *background*, ou seja, uma tomografia sem a presença da amostra (figura 4.1)

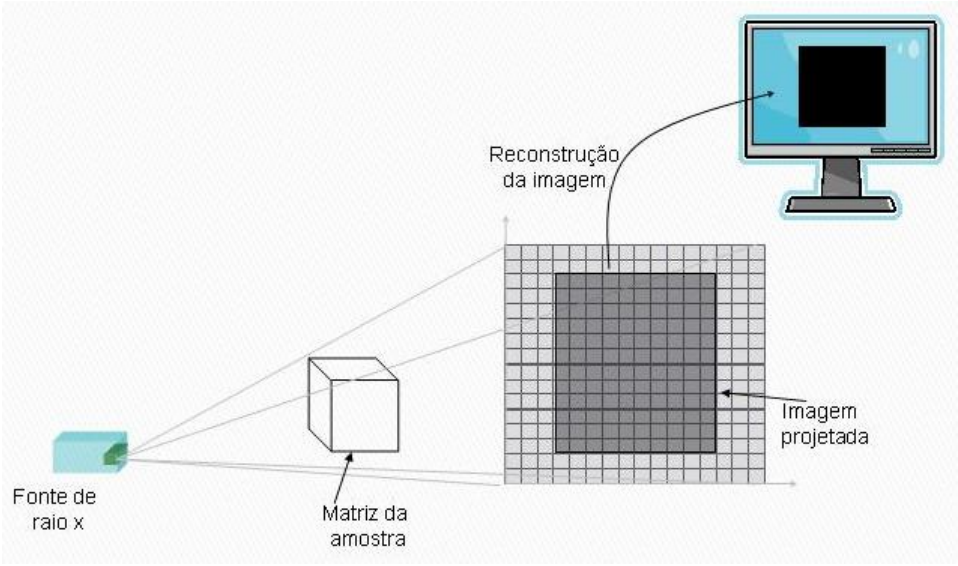
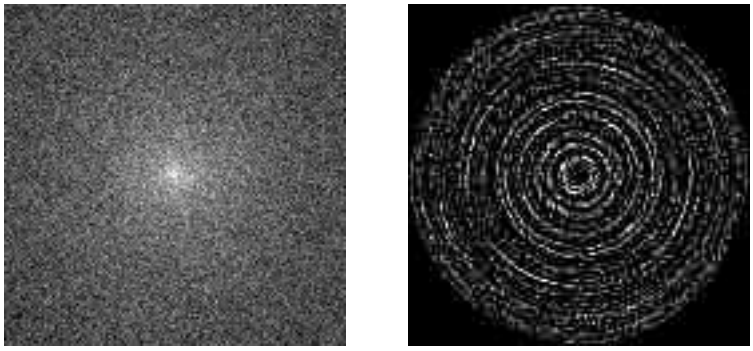


Figura 4.1: Esquema da simulação de *background*

Teoricamente a reconstrução desta simulação deveria mostrar uma imagem completamente preta, já que todos os fótons simulados devem atingir o detector de maneira uniforme e com a mesma energia que foram emitidos. Porém a reconstrução da figura 4.2-(a) mostra uma imagem que não é completamente preta (figura 4.2-(b)).



(a) (b)

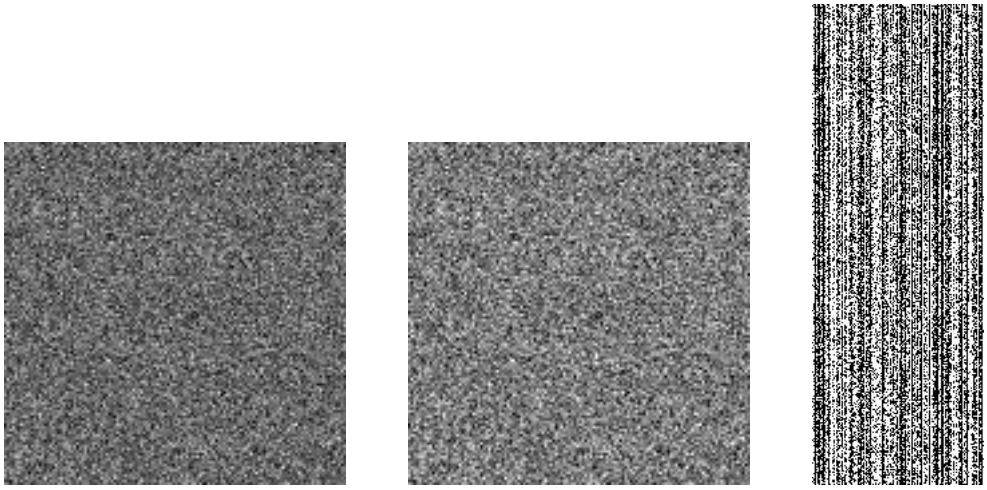
Figura 4.2: Em (a) simulação realizada sem a presença da amostra. Em (b) a reconstrução da imagem (a) com uma pequena mudança de contraste.

O ruído que aparece na figura 4.2-(b) tem um padrão circular. Este padrão é devido ao movimento circular do sistema fonte-detector. Para reconstruirmos uma imagem simulada é necessário recuperarmos o coeficiente de atenuação linear da amostra, que pode ser obtido através da equação 2.12, assim temos:

$$\frac{I_0}{I} = e^{\mu x} \Rightarrow ln\left(\frac{I_0}{I}\right) = \mu x \tag{4.1}$$

O procedimento da reconstrução da amostra está ilustrado na figura 4.3. A imagem 4.3-(c) que indica que o sorteio da direção do fóton é, na verdade, pseudo-aleatório, pois podemos ver claramente a formação de linhas em posições bem definidas.

É importante ressaltar que este tipo de ruído está presente em todas as tomografias realizadas, porém nos tomógrafos comerciais (médicos e industriais) existe um sistema de auto calibração, que verifica a eficiência de todos os pixels do detector e, se necessário, corrige os *pixels* (detectores) mais sensíveis para um valor médio de sensibilidade, minimizando este ruído. Outra maneira é realizar essa correção via *software* ainda na imagem gerada pelo o detector e não na imagem reconstruída.



(a) (b) (c)

Figura 4.3: Em (a) o resultado de I_0/I. Em (b) O resultado do loraritmo natural de (a) segundo a equação 4.1. Em (c) corte transversal de (b).

4.1.2 Coeficiente de atenuação linear

O objeto simulado neste experimento é chamado escalímetro, pois com ele é possível identificar a penetrabilidade do feixe de raio X. Neste experimento cada degrau têm 0,1 cm de altura, 1,0 cm de largura e 1,5 cm de comprimento e foi posicionado de acordo com a figura 4.4. O escalímetro tem um total de 9 degraus e é composto de hidroxiapatita (figura 4.5).

Com este objeto, podemos recuperar o coeficiente de atenuação linear da hidroxiapatita, para isso devemos calcular o valor de (μ). De a cordo com a equação 2.12 obtemos:

$$\mu = \frac{1}{x} ln \left(\frac{I_0}{I} \right) \tag{4.2}$$

onde I_0 é a imagem de *background* e I é uma projeção da simulação. Medindo a contagem de fótons que chegam no primeiro degrau na figura 4.5-(b) e dividindo por sua espessura obtemos o valor de μ. A tabela 4.1 traz os valores obtidos para μ.

O valor médio do coeficiente de atenuação linear foi de $(2,52 \pm 0,2) cm^{-1}$ e o valor teórico é de $2,50$ cm^{-1} ou seja um erro relativo de 0,8%. O gráfico da figura 4.6 mostra

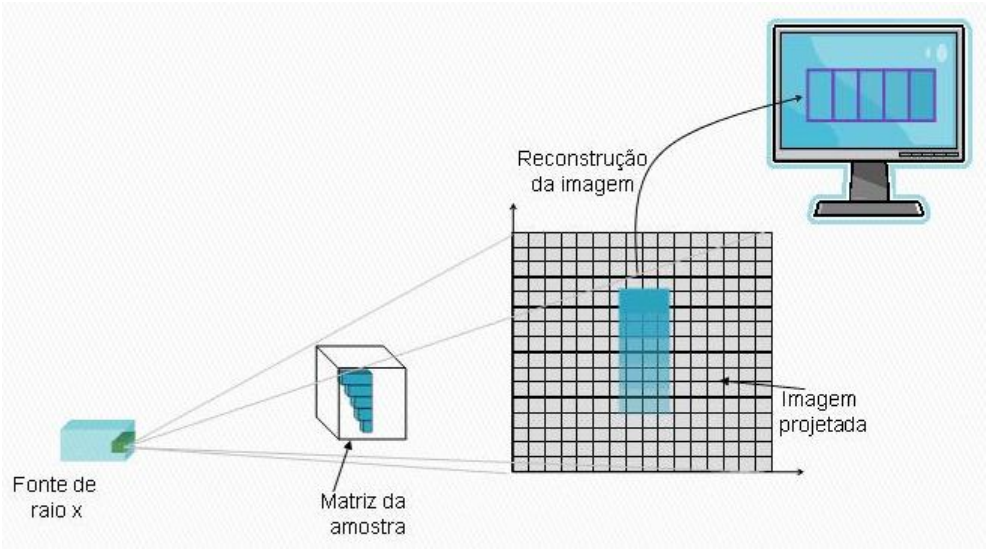
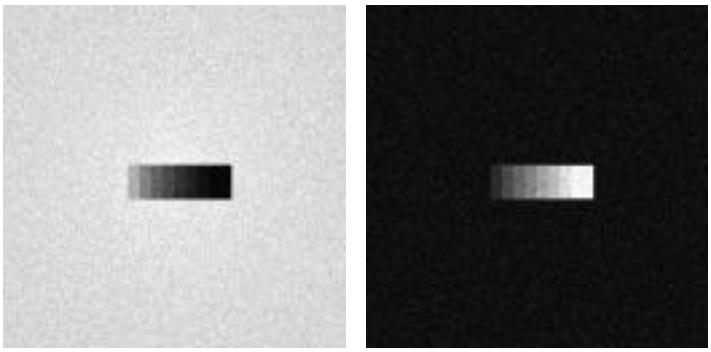


Figura 4.4: Esquema da simulação de um escalímetro



(a) (b)

Figura 4.5: (a) Imagem simulada de um escalímetro de hidroxiapatita (osso) (b)Imagem obtida após calcularmos o $ln\frac{I_0}{I}$, representando a reconstrução da amostra.

os valores obtidos.

Tabela 4.1: Valores de μ obtidos

degrau	Espessura (cm)	$\mu \cdot x$	Desvio Padrão	$\mu(cm^{-1})$
1	0,1	0,25	0,04	2,50
2	0,2	0,51	0,04	2,55
3	0,3	0,77	0,05	2,57
4	0,4	1,29	0,04	2,57
5	0,5	1,52	0,07	2,58
6	0,6	1,52	0,06	2,54
7	0,7	1,82	0,08	2,60
8	0,8	2,06	0,09	2,57
9	0,9	2,04	0,07	2,26

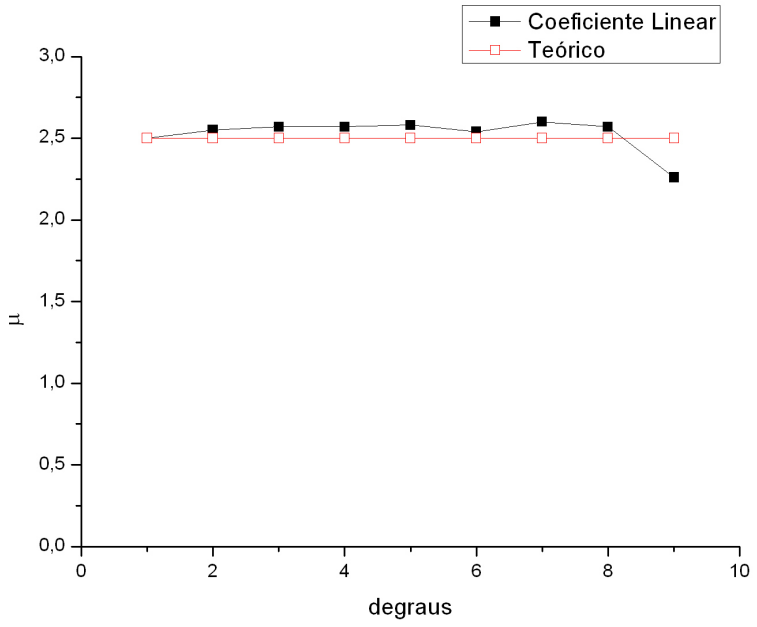


Figura 4.6: Grafido dos valores de μ obtidos.

4.1.3 Deformação

Este experimento foi realizado dispondo várias esferas na matriz da amostra e tem como objetivo verificar se existe alguma distorção relativa ao posicionamento da amostra dentro da sua matriz. A figura 4.7 representa o esquema da simulação realizada neste experimento.

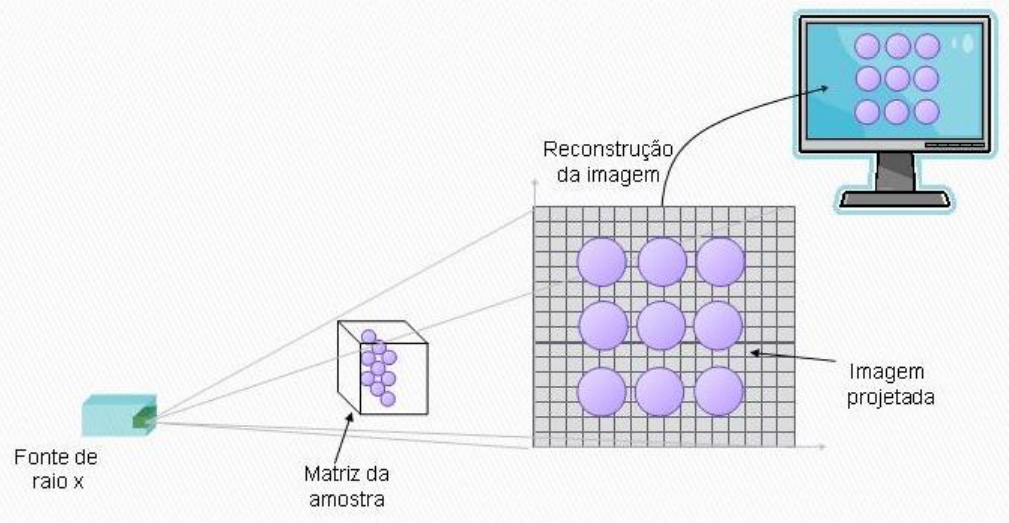
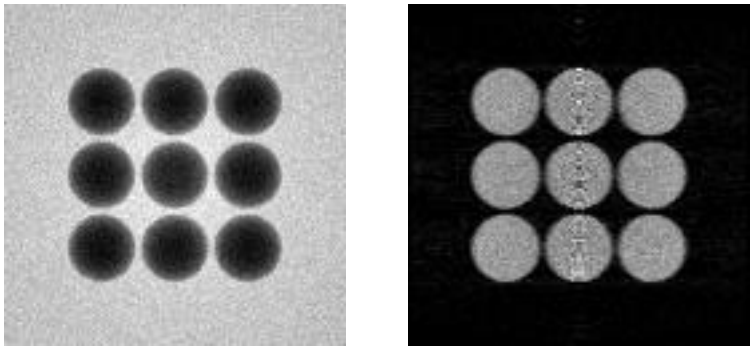


Figura 4.7: Esquema da simulação para o experimento de reconstrução do coeficiente de atenuação

Para descobrir se há ou não uma deformação dependendo da posição dos objetos dentro da matriz da amostra, foi medido o diâmetro da esfera nos eixos x e y, tanto na imagem simulada quanto na reconstruída. Na figura 4.8 temos a primeira projeção das esferas e um corte da reconstrução.

Os resultados da medida do diâmetro das esferas estão na tabela 4.2, sendo que a esfera 1 é a de cima e da esquerda, a esfera 2 é a da linha de cima e do meio e assim por diante.

Para obtermos a medidas em milímetros da tabela 4.2 devemos multiplicar o valor em *pixels* por 0,2 mm, que é o tamanho de cada *pixel* nesta simulação.



(a) (b)

Figura 4.8: Em (a) imagem simulada das esferas. Em (b) reconstrução da amostra.

Tabela 4.2: Medidas de diâmetro da figura 4.8-(a)

Esfera	Diâmetro em y		Diâmetro em x	
	(pixel)	(mm)	(pixel)	(mm)
1	26	5,2	26	5,2
2	26	5,2	26	5,2
3	26	5,2	26	5,2
4	26	5,2	26	5,2
5	26	5,2	26	5,2
6	26	5,2	26	5,2
7	26	5,2	26	5,2
8	26	5,2	26	5,2
9	26	5,2	26	5,2

Tabela 4.3: Medidas de diâmetro da figura 4.8-(b)

Esfera	Diâmetro em y		Diâmetro em x	
	(pixel)	(mm)	(pixel)	(mm)
1	26	5,2	27	5,4
2	27	5,4	27	5,4
3	27	5,4	26	5,2
4	26	5,2	27	5,4
5	27	5,4	27	5,4
6	27	5,4	27	5,4
7	26	5,2	27	5,4
8	26	5,2	27	5,4
9	27	5,4	26	5,2

4.1.4 Ângulos retos

A figura (4.10) mostra a reconstrução de um quadrado formado por 4 cilindros elípticos de hidroxiapatita com 20 *pixels* de raio na direção x e 10 *pixels* na direção z. A mostra foi configurada de forma que o início de um cilindro coincida com o fim de outro, formando desta maneira, um ângulo de 90 graus. Esta amostra está posicionada de acordo com a figura 4.9. Porém na reconstrução os cantos parecem arredondados. Isto ocorre pois a espessura que o feixe atravessa nas quinas é muito pequena, assim o feixe emergente tem quase que a mesma intensidade do feixe emitido (figura 4.11).

Por este motivo os cantos desta e de outras amostras aparecem arredondados, pois o feixe colide com o detector com praticamente a mesma energia que foi emitido.

As partes mais escuras da figura 4.10 são devido ao endurecimento de feixe. Este fenômeno ocorre quando o feixe de fótons atravessa um objeto e perde parte de sua energia. Como a seção de choque do objeto depende da energia de incidência do fóton, a probabilidade de interação aumenta, isso faz com que um número pequeno de fótons

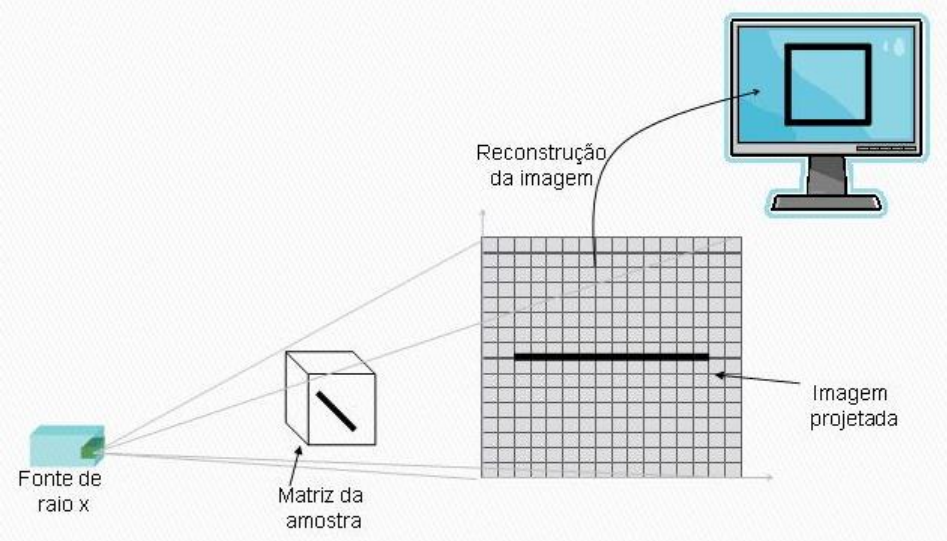


Figura 4.9: Esquema da simulação do quadrado.

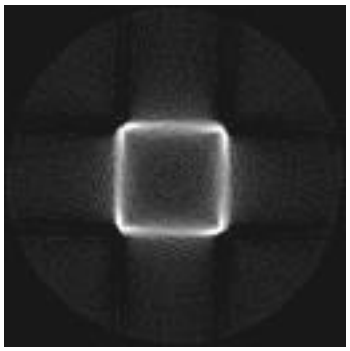


Figura 4.10: Reconstrução de um quadrado composto de hidroxiapatita.

colidam com os detectores. Assim o feixe que era inicialmente monocromático terá um espectro de energia, abaixo da energia inicial (ver figura 4.11).

4.1.5 Ruído de espalhamento

Este experimento consiste em colocar uma grade na matriz da amostra. Esta amostra tem as mesmas características da amostra apresentada na figura 4.9. A figura 4.12 mostra o esquema montado para este experimento.

Com a reconstrução desta amostra (figura 4.13) podemos verificar os arredondamentos das bordas de cada cilindro. A distorção do quadrado central é devida à absorção dos

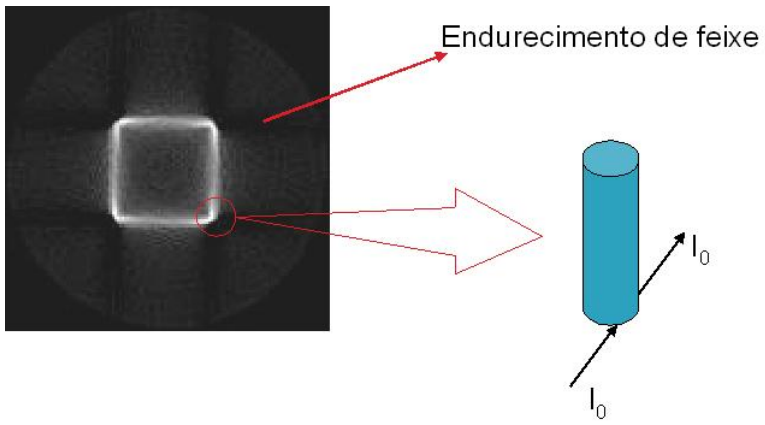


Figura 4.11: Ilustração dos fenômenos de endurecimento de feixe e das bordas.

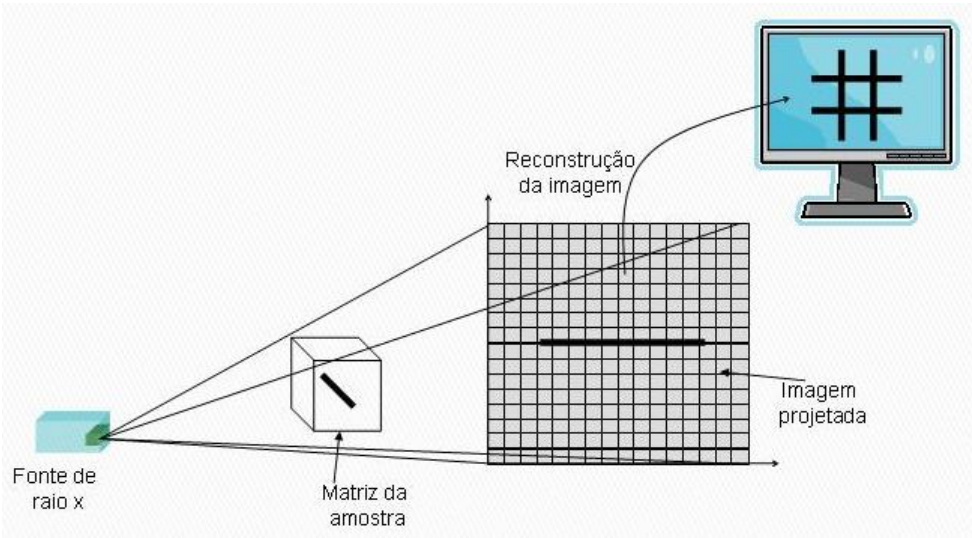


Figura 4.12: Representação do experimento de espalhamento de feixe.

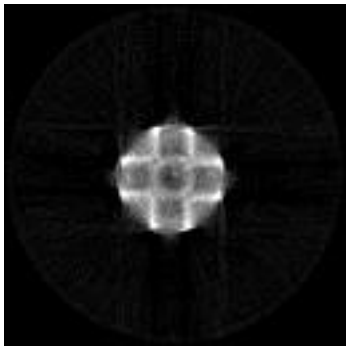


Figura 4.13: Imagem de uma amostra em forma de grade

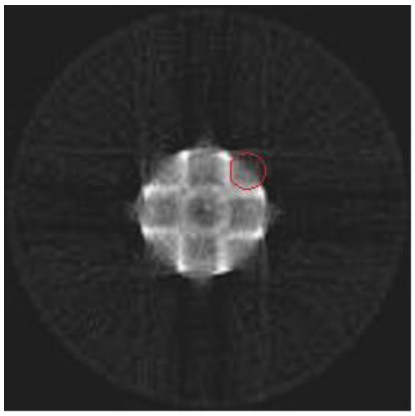


Figura 4.14: Ruído devido ao espalhamento

feixes pelas outras partes da amostra

A parte destacada na figura 4.14 mostra que os cantos da grade ficam sem muita definição devido ao espalhamento do feixe incidente. Este espalhamento faz com que o fóton colida com um pixel diferente do detector. Assim o programa de reconstrução interpreta que este fóton foi originado em uma outra posição, causando o ruído na parte indicada.

4.1.6 *Visible Human*

O simulador de TC proposto é capaz de realizar a configuração da amostra através de leitura de arquivo. A única exigência é que este arquivo seja no formato '.raw' (arquivo em formato binário onde as imagens ficam armazenadas em seqüência como em um filme

em película) e que a imagem esteja segmentada em escala de cinza. Esta habilidade permite ao simulador utilizar uma imagem, ou seqüências de imagens para simular o objeto, não ficando restrito apenas a amostras formadas por objetos básicos (cilindro, esfera e paralelepípedo). A 4.15-a mostra a imagem original do projeto *Visible Human*. A imagem, ou melhor, a seqüência de imagens, da coxa Humana foi utilizada como amostra para simularmos uma coxa. Já a 4.15-b mostra a imagem do projeto *Visible Human* segmentada com três índices, que se referem à pele, músculo estriado e osso.

A simulação desta amostra foi feita com $1x10^8$ eventos, com uma distância fonte detector de 2,00 m, sendo a matriz da amostra com 1/10 do tamanho da imagem e a resolução da imagem é de 800x800 *pixels*. O resultado desta simulação está apresentado na 4.15-c.

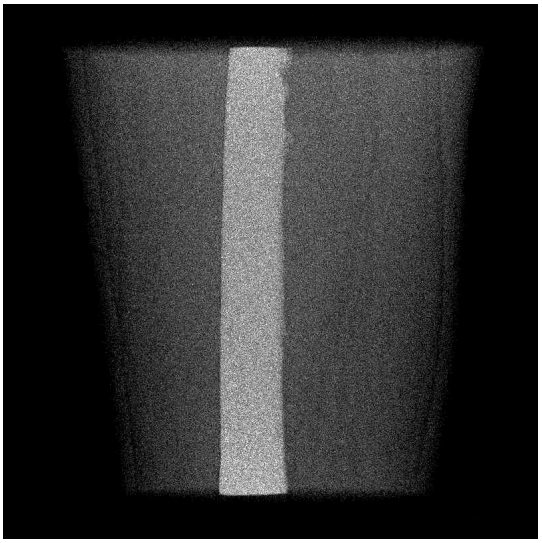
4.2 Características da simulação

A simulação da tomografia computadorizada gera, ao final de cada projeção, uma imagem que contém informações sobre a estrutura da amostra que se está estudando. Como estamos realizando uma simulação deste sistema devemos estudar o comportamento do simulador. Para isso é necessário variarmos os parâmetros de configuração do simulador, como o tamanho da matriz do detector e medir o tempo total de simulação e o tempo de cada projeção, variar o tamanho da matriz da amostra, variar o número de eventos por raio-soma entre outros.

Estes experimentos não foram realizadas em um *cluster* e sim em computadores pessoais e com diferentes configurações. Um destes computadores tem um processador Pentium 4 de 3.2GHz com 512 Mb de memória RAM rodando o sistema Fedora Core 7 de 32 bits. O outro computador tem um processador Atlhon 64 X2 (núcleo duplo) de 3.0GHz com 4Gb de memória RAM rodando o sistema Ubuntu 8.04 de 64 bits.

O primeiro experimento tratou de variar o tamanho da matriz da amostra. Foram

(a)



(b) (c)

Figura 4.15: Em (a) Imagem original do projeto *Visible Human*. (b) Imagem segmentada com tons de cinza e (c) Simulação de parte da coxa do projeto *Visible Human*

feitas várias simulações que consistiram em variar o tamanho da amostra de 101x101x101 até 601x601x601 utilizando 10^7 eventos com uma amostra simulada em forma de haltere composta de gordura e tecido com algum tipo de anomalia. Os resultados estão expressos nas tabelas 4.4 e 4.5.

Tabela 4.4: Experimento simulado em um Pentium 4 de 3.2GHz.

tamanho da matriz da amostra	tamanho do arquivo (Kb)	tempo de simulação (h:m:s)
101x101x101	171,1	0:01:50,49
201x201x201	176,1	0:01:57,57
401x401x401	176,1	0:02:05,43
601x601x601	176,1	0:02:29,85

Tabela 4.5: Experimento simulado em um Atlhon X2 de 3GHz.

tamanho da matriz de amostra	tamanho do arquivo (Kb)	tempo de simulação (h:m:s)
101x101x101	171,1	0:00:49,31
201x201x201	176,1	0:00:49,47
401x401x401	176,1	0:00:49,31
601x601x601	176,1	0:01:02,34

As tabelas 4.4 e 4.5 mostram que o tamanho da matriz da amostra tem um pequeno impacto sobre o tempo de simulação. Um aumento no tamanho da matriz da amostra melhora a qualidade da imagem pois aumenta a densidade de pixels. Nestes testes o tamanho da imagem é de 128x128 pixels, ou seja é uma imagem com uma resolução de 16Kb.

A diferença de tempo de simulação da tabela 4.4 para a tabela 4.5 se deve ao fato de estarmos realizando a simulação em um computador com um único núcleo enquanto o segundo computador é de núcleo duplo.

Embora exista esta diferença de tempo entre das duas configurações utilizadas não existe qualquer diferença nos resultados da simulação mesmo se variarmos o tamanho da matriz do detetor (tabela 4.6). Esta tabela mostra a dependência do simulador com o tamanho da matriz do detector. Aqui pode-se observar que quanto maior for a densidade de pixel (tamanho do detetor) maior será o espaço ocupado pelas imagens das projeções. A tabela 4.6 mostra o tamanho de uma única imagem (uma projeção).

Tabela 4.6: Relação entre o tamanho do detector e o tempo de simulação por projeção.

Matriz do detector	Arquivo da imagem(Kb)	tempo de simulação(s)
128x128	176,1	48,45
256x256	704,3	47,80
521x521	2867	49,30
640x640	4403	49,98
800x600	5120	49,93
1000x1000	10752	50,05
2000x2000	43008	52,96

4.3 Imagens simuladas

Ao fim de cada projeção, ou seja, após a simulação terminar de realizar a última interação da radiação com a matéria de uma determinada projeção, o programa armazena os dados em um arquivo no disco com um nome determinado pelo usuário acrescido do número da projeção no formato *'.pgm'* (Ex. saida.241.pgm). Este formato *'.pgm'* foi escolhido por ser um formato de imagem que, na verdade, é um arquivo texto organizado em tabela, sendo de fácil manipulação e criação. Este formato de imagem pode ser armazeando em formato texto ou em formato binário. A única diferença entre esses dois formatos é o tamanho do arquivo em disco, pois no formato texto o arquivo fica

significativamente maior e é muito mais fácil de se trabalhar.

A qualidade da imagem depende básicamente do número de eventos por raio-soma, como podemos ver na figura 4.16. Aumentamdo-se o número de eventos aumentamos também o tempo de simulação. Este aumento no tempo de simulação é praticamente linear, por exemplo, se aumentarmos em uma ordem de grandeza o número de eventos o tempo de simulação irá aumentar em 10 vezes para cada projeção. Em uma simulação completa o tempo de simulação fica em torno de 3600 vezes maior, pois devemos realizar, pelo menos, 360 passos angulares (tabela 4.7).

Tabela 4.7: Dependência do tempo de simulação com o número de eventos.

Número de eventos	Tempo de simulação (hh:mm:ss)
1×10^6	00:00:11
1×10^7	00:01:37
1×10^8	00:15:53
1×10^9	02:38:50
1×10^{10}	26:29:31

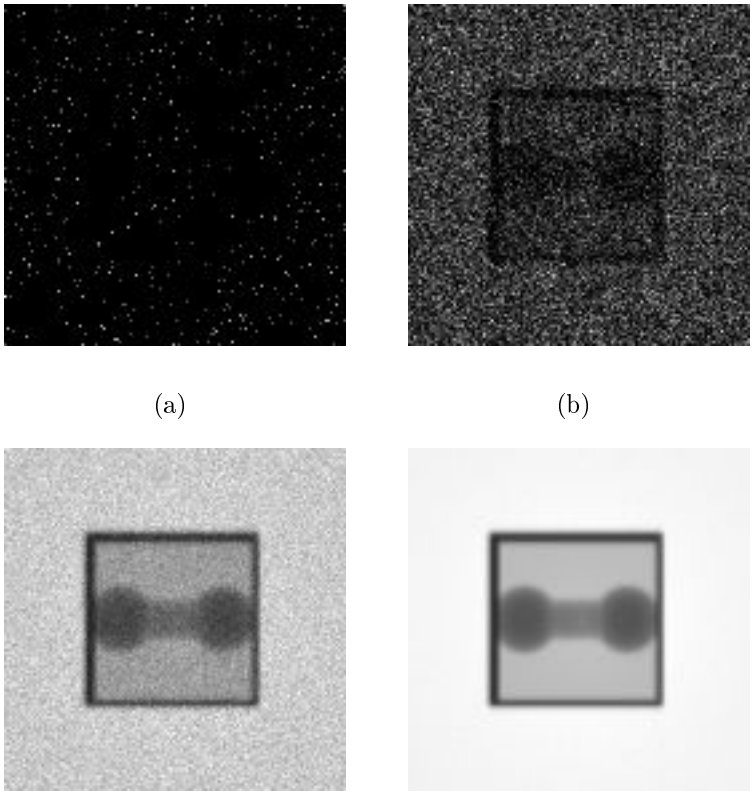
Além das imagens o simulador mostra na tela os dados da simulação. No caso do uso de um *cluster*, os gerenciadores de fila salvam todas as saídas de tela em um arquivo em formato texto no diretório do usuário. Assim é possível realizar uma melhor análise do simulador. Este arquivo armazena todos os parâmetros de configuração do simulador além de registar o tempo de cada projeção, do *background*, a hora de início do *job* no cluster e também a hora de término do *job*. Os horários de início e fim do job são lidos do relógio do computador. A primeira linha deste arquivo é escrita palo o programa que gerência a fila do *cluster*. Abaixo temos o arquivo de saída da simulação feita com 10^9 eventos e seis processadores.

```
Job started at Sat Jul 19 22:51:30 BRT 2008
Abrindo arquivo de dados do sistema      : ./config/Sistema.txt
Abrindo arquivo configuracao da fonte    : ./config/Fonte.txt
```

```
Abrindo arquivo de dados do espectro     : ./config/Espectro-Fonte.txt
Abrindo arquivo configuracao do detector : ./config/Detector.txt
Projecoes: 1
Raios    : 1
Eventos  : 1e+09
Posicao  : -20.000000   0.000000    0.000000
Direcao  : 1.000000  0.000000    0.000000
Foco     : 0.050000
Abertura : 0.075000
Detector
  Posicao   : 20.000000   0.000000    0.000000
    Direcao   : -1.000000   0.000000    0.000000
    Espessura : 0.100000
    Largura   : 2.000000
    Altura    : 2.000000
    Matriz    : 128 128
Numero de processadores : 6
Eventos por processador : 1.667e+08
Proj. inicial: 0
Proj. final  : 0
Abrindo arquivo de dados da amostra      : ./config/Amostra.txt
Abrindo arquivo de construcao da amostra : ./config/Dados.txt
Abrindo arquivo de composicao da amostra : ./config/Composicao.txt
Abrindo arquivo com secoes de choque     : ./config/cGordura.xmu
Abrindo arquivo com secoes de choque     : ./config/cCarcinoma.xmu
Abrindo arquivo de composicao da amostra : ./config/Composicao.txt
Abrindo arquivo com secoes de choque     : ./config/cAluminio.xcom
0 35905  1399.463416
1 40368  1000.017822
Job ended at Sat Jul 19 23:31:33 BRT 2008
```

A figura 4.16 mostra agumas imagens simuladas com uma amostra que é composta por gordura (paralelepípedo e esferas) e tecido com algum tipo de anomalia (cilindro central) e foi simulada com diferentes números de eventos. A escolha desta amostra se deve ao fato de serem tecidos amorfos e com uma alta taxa de espalhamento e ilustra as formas das figuras geométricas que o simulador é capaz de interpretar (esfera, cilindro e paralelepípedo).

Visualmente não existe diferença entre a simulação realizado com 1×10^8 e a simulação realizada com 1×10^9 ou com 1×10^{10} (figura 4.17). Por este motivo, as simulações estão sendo feitas, em geral, com 1×10^8 eventos, pois mesmo tendo uma qualidade menor, com esse número de eventos é possível realizar uma simulação completa (360 projeções) da



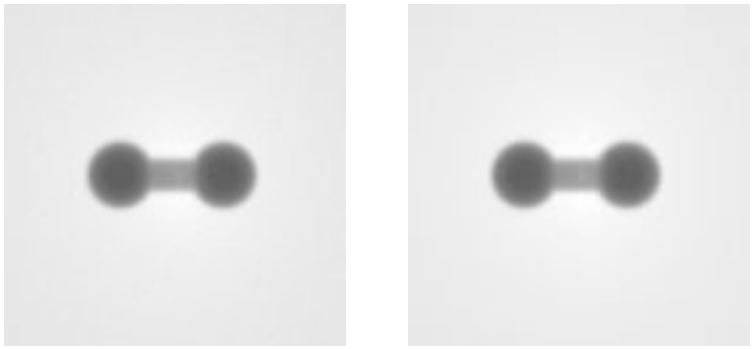
(a) (b)

(c) (d)

Figura 4.16: Imagens radiográficas simuladas com: (a) 10^3, (b) 10^5, (c) 10^7 e (d) 10^9 eventos por raio-soma.

amostra em torno de 24 horas. Assim o processo de simulação e reconstrução demora cerca de 1 dia, já que a reconstrução é feite de maneira muita rápida.

A figura 4.18 mostra uma amostra composta totalmente por alumínio em formato de escada. A simulação desta amostra foi realizada com 1×10^9 eventos por raio-soma em 16 processadores. Diferente da amostra em forma de haltere, o alumínio é um material cristalino porém muito denso, o que aumenta a taxa de absorção. Isto quer dizer que poucos fótons são capazes de atravessar toda a espessura da amostra. Isto pode ser contornado aumentando a energia com que o fóton é emitido.



(a) (b)

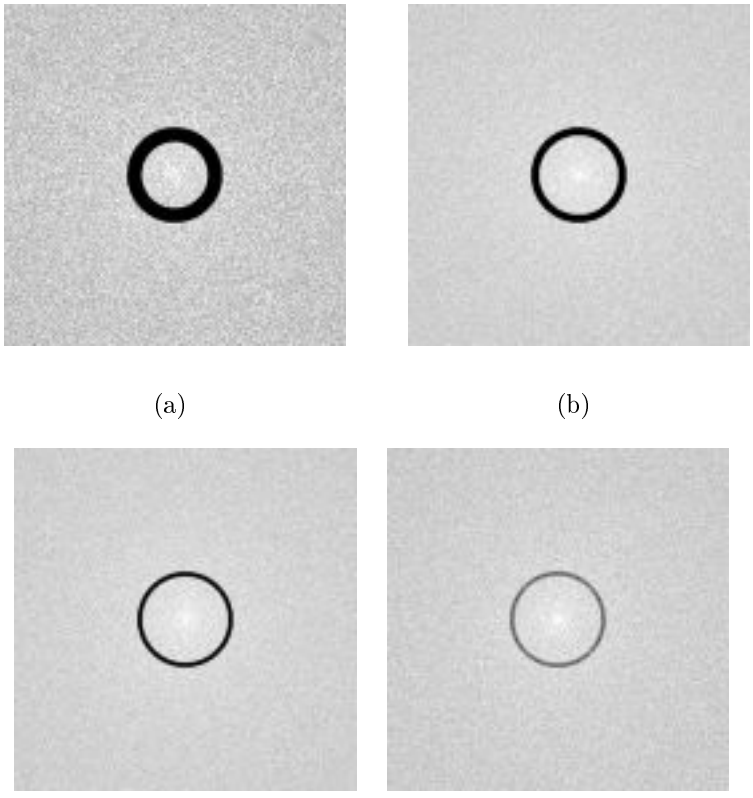
Figura 4.17: Diferença visual das imagens radiográficas simuladas com: (a) 10^9 e (b) 10^{10} eventos por raio-soma.

Figura 4.18: Imagem de uma 'escada' de alumínio de tamanho 128x128 pixels.

4.3.1 Resolução das imagens

A figura 4.19 mostra a resolução das imagens do simulador. Essas imagem foram simuladas com 1×10^8 eventos num total de 360 projeções para podermos realizar a reconstrução da amostra. A amostra é composta de dois cilindros concêntricos de alumínio, formando um cilindro oco. Foram realizadas quatro simulações com esta mesma amostra apenas variando a espessura do cilindro desde 5 mm até 500 μm (medidas relativas à matriz da amostra).

A matriz da amostra, que é o parametro responsável pelo o tamanho da amostra, foi definida com 500 *pixel* nos três eixos, sendo que o tamanho (comprimento) desta matriz é de cinco centímetros, assim podemos facilmente calcular que o tamanho de cada *voxel* é



(a) (b)

(c) (d)

Figura 4.19: Cilindros simulados com diferentes espessuras, em (a) 5mm, (b) 2mm, (c) 1mm e (d) 500 μm

de 0,01 cm. Para simularmos o cilindro com espessura de 500μm foi nescessário configurar os cilindros com apenas um *voxel* de distância entre os cilindros.

A figura 4.22 mostra uma esfera composta de gordura com diâmetro de 230 *voxels* (2,3 cm) com quatro pequenas esferas de hidroxiapatita, a menor com 10 *voxels* de raio, uma segunda com 15 *voxels* (0,15 cm) de raio, a terceira com raio de 25 *voxels* (0,25 cm) e a ultima com 50 *voxels* (0,5 cm). Essas microcalcificações foram colocadas dentro de uma esfera de forma a não ocorrer superposição em nenhuma projeção da simulação.

4.3.2 Imagens Reconstruídas

As reconstruções foram feitas com algoritmos de reconstrução de imagens tomográficas para simulações com feixes paralelos. O fato de simularmos um sistema tomográfico que

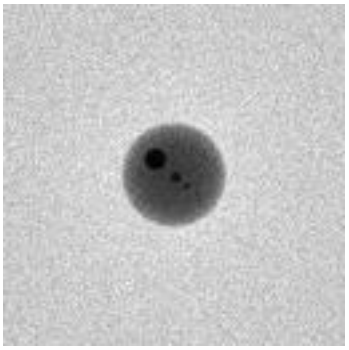


Figura 4.20: Simulação de 'microcalcificações' em uma esfera de gordura

utiliza feixe cônico, força uma pequena mudança no algoritimo de recosntrução. Nas simulações de tomografia com feixes paralelos, cada projeção forma uma fatia do objeto simulado, sendo necessário somente combiná-los para termos a imagem reconstruída da amostra. No caso de uma simulação de TC com feixe cônico, é necessário pegar a primeira linha de cada projeção para criar uma fatia do objeto simulado e somente depois de montadas todas as fatias o algoritmo de reconstrução é idêntico ao de reconstrução de sistemas tomográficos de feixes paralelos.

As imagens simuladas devem ser reconstruídas para verificar se a simulação está realmente simulando a amostra escolhida, incluindo as substâncias que compõem a amostra como a sua posição dentro da amostra e também o seu tamanho em *pixels* ou em centímetros. O simulador de TC está utilizando uma magnificação de 2, ou seja, se a amostra foi configurada com 1 cm de raio na matriz da amostra ela deverá ser projetada na imagem simulada e reconstruída com 2 cm de raio.

A figura 4.21 mostra um cilindro de alumínio com a mesma espessura da figura 4.19 [d]. Mostrando que o simulador está realizando corretamente a amostra simulada. Esta imagem foi montada com dois cilindros. O cilindro mais interno foi configurado com 200 *pixels* de raio e o interno com 195 *pixels* formando um cilindro ocô de $500\mu m$.

Para confirmar se o tamanho da amostra simulada é o mesmo da amostra reconstruida, foi utilizado o *software* ImageJ, que é um *software* livre baseado em Java. Com este

(a) (b) (c)

Figura 4.21: Cilindro de aluminio em (a) imagem simulada, (b) uma fatia lateral do cilindro reconstruido, (c) fatia frontal

software foram medidas a circunferência e a espessura das imagens simuladas e reconstruídas.

Tabela 4.8: Medidas da figura 4.21 (todas as medidas em pixels).

	Imagem simulada	Imagem reconstruída
Diâmetro externo	36	36
Diâmetro interno	32	32
Circunferência externa	988	998
Circunferência interna	812	812

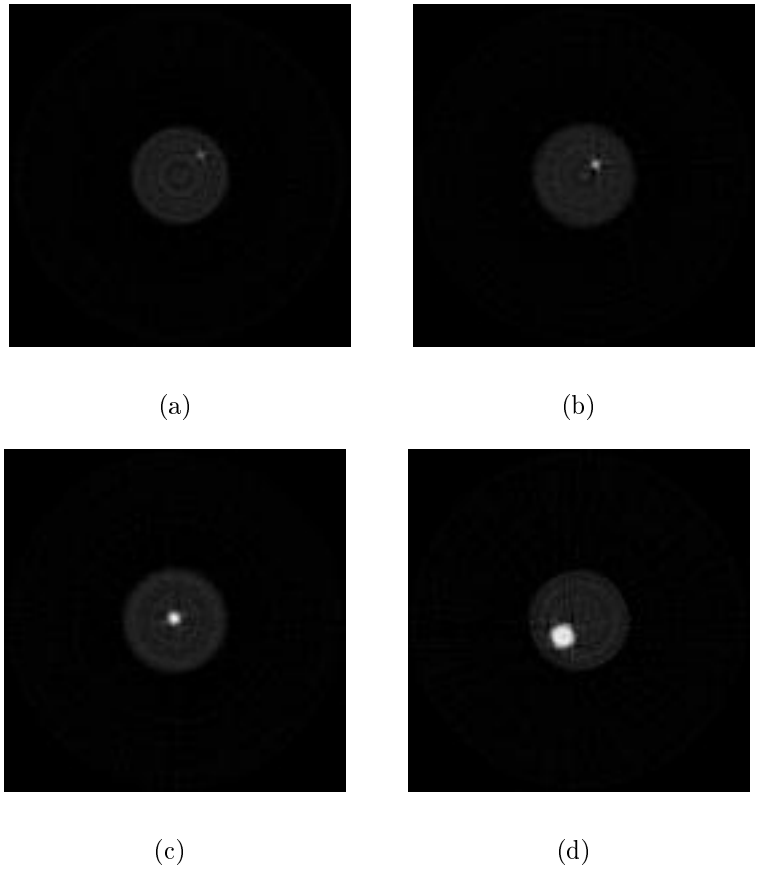
A tabela 4.8 mostra algumas medidas da amostra apresentada na figura 4.21, nela podemos ver que não há nenhuma perda de informação quanto ao tamanho da amostra e posicionamento dela dentro de sua matriz. Porém, a configurção da amostra é feita em *voxels* e não *pixels* e esta amostra foi configurada para ter um diâmetro externo de 200 *voxels* em uma matriz de 5 cm de lado, o que corresponde a um raio externo de 2 cm. Já o raio interno foi configurado com 195 *voxels* (1,95 cm). E os dois cilindros têm 4,5 cm de comprimento (450 *voxels*).

Já para a figura 4.22 a medição das esferas fica de difícil realização devido ao tamanho reduzido da imagem e também das esferas (tabela 4.9). Porém a reconstrução da amostra

Tabela 4.9: Área das microcalcificações da figura 4.22 (todas as medidas em pixels).

	Esfera 50 *voxels*	Esfera 25 *voxels*	Esfera 15 *voxels*	Esfera 10 *voxels*
Imagem simulada	80	21	9	1
Imagem reconstruída	80	26	9	1

nos permite identificar a menor das esferas (10 *voxels* de raio) muito mais fácilmente que apenas observando a imagem simulada (figura 4.22)



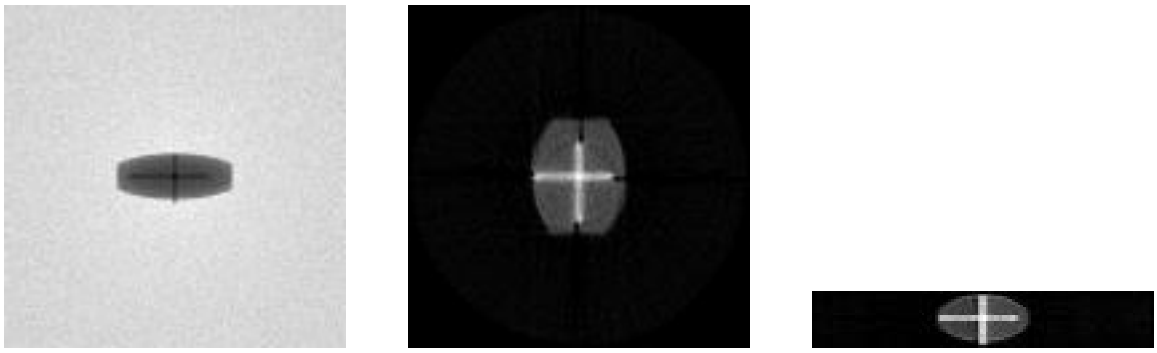
(a) (b)

(c) (d)

Figura 4.22: Reconstrução da figura 4.22, em (a) esfera de 10 *pixels* de raio, (b) esfera de 15 *pixels* de raio, (c) esfera de 25 *pixels* de raio e em (d) esfera de 50 *pixels* de raio

Já a figura 4.23 mostra um elipsóide de gordura truncado com três cilindro de hidroxiapatita formando uma estrutura semelhante a uma trabécula (formação ossea que se encontra na medula). Os cilindros têm 10 *voxels* de raio e o elipóide é preenchido de

gordura.



(a) (b) (c)

Figura 4.23: (a) Imagem simulada da 'trabécula' em (b) reconstrução da trabecula e em (c) outra fatia da imagem reconstruída

4.4 Análise de desempenho

Como dito anteriormente, uma simulação realizada com 1×10^8 eventos leva cerca de 24 horas para completar. Porém este tempo também depende no número de processadores que estamos utilizando. O gráfico da figura 4.24 mostra a relação entre carga de trabalho e número de processadores. Todas as simulações realizadas para o cálculo do *speed up* foram realizadas com 1×10^9 com 0.5 cm de tamanho na matriz da amostra. Este tamanho reduzido foi escolhido somente para realizar as análises de desempenho.

As quedas de performance com 14, 15 e 16 processadores pode ser explicada pela política de compartilhamento de processadores do CESUP e também pela a comunicação entre os processadores. A figura 4.25 mostra as linhas de tempo da simulação com 14, 15 e 16 processadores. Nela podemos ver que existe um processador que demora muito mais tempo para terminar a simulação da projeção. Como no *cluster* do CESUP todos os nós de processamento são idênticos a demora para o termino da projeção pode ser devido a um outro processo rodando no processador ao mesmo tempo que o processo do simulador.

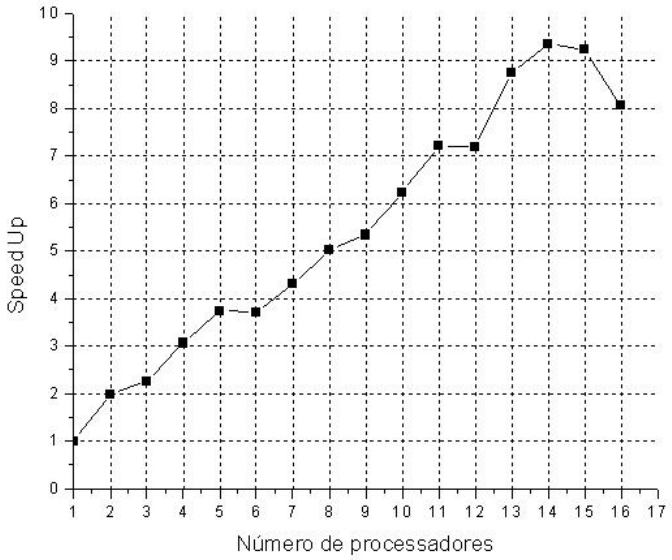


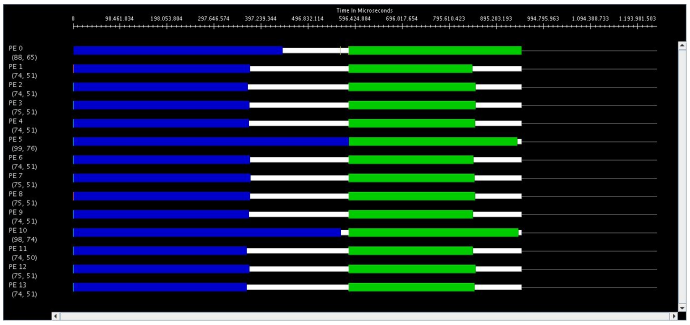
Figura 4.24: Gráfico do *Speed Up* do simulador realizado no CESUP.

Estes gráficos são obtidos com uma ferramenta nativa do *CHARM++* chamada *projections*.

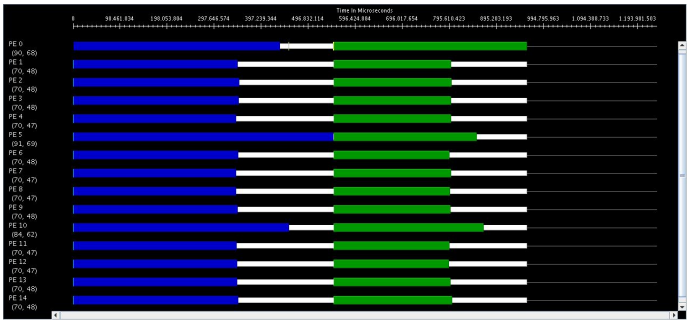
A análise das simulações da figura 4.21[a] estão apresentadas nas figuras 4.26. A parte que está em verde no início de cada processador representa a instância de configuração. Já a barra na cor azul, representa a instância de interação. Como esta simulação foi realizada com 360 projeções esta instância será chamada 360 vezes mais o *Background*.

Os gráficos de linha de tempo fornecidos pelo *projection* permitem rastrear os envios e recebimentos de mensagens, eles também nos dão informações sobre o tamanho da mensagem, início e fim da instância, tempo de criação da mensagem, entre outros. A figura 4.27 ilustra uma mensagem criada no processador 0 (zero) que foi enviada para o processador 9 informando ao processador mestre que a primeira projeção foi terminada. Assim o processador 0 (zero) envia uma outra mensagem para todos os processadores para iniciarem a próxima projeção.

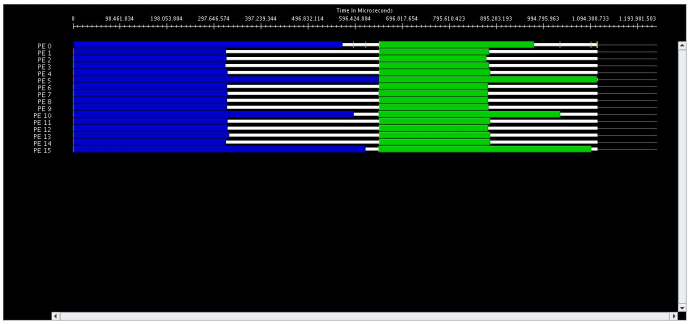
Além dessas informações a linha de tempo informa a porcentagem de utilização de cada processador logo abaixo do número do processador. O primeiro número é referente



(a) 14 processadores



(b) 15 processadores



(c) 16 processadores

Figura 4.25: Linhas de tempo do simulador de TC. As cores identificam diferentes projeções

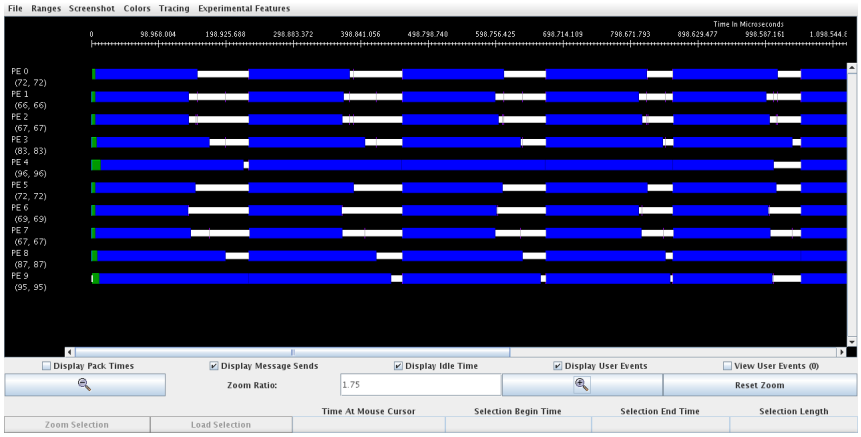


Figura 4.26: Linha de tempo da simulação do cilindro da figura 4.21[a]. Em verde a configuração inicial e em azul a primeira projeção

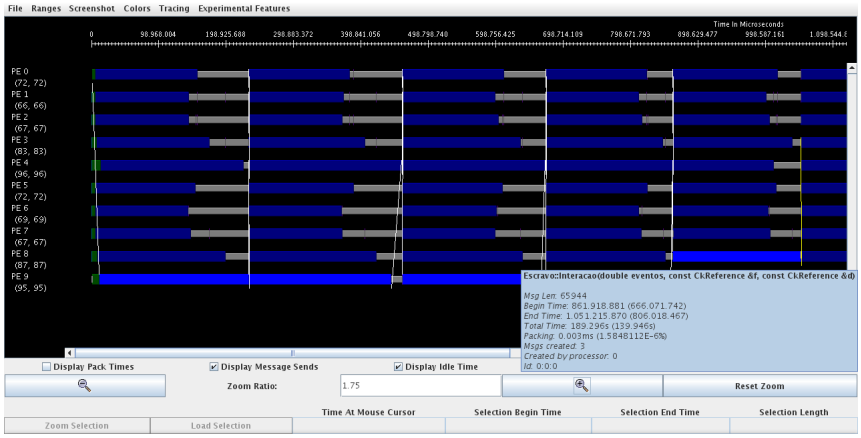


Figura 4.27: Rastreamento do envio de mensagens no simulador

à instância da interação e o segundo é referente instância do *background*

Uma outra ferramenta fornecida pelo *projection* é a visualização do recebimento de mensagens por processador. Na figura 4.28 podemos ver que o processador zero que recebe o maior número de mensagens. Isto ocorre devido ao papel de gerente desempenhado por este processador.

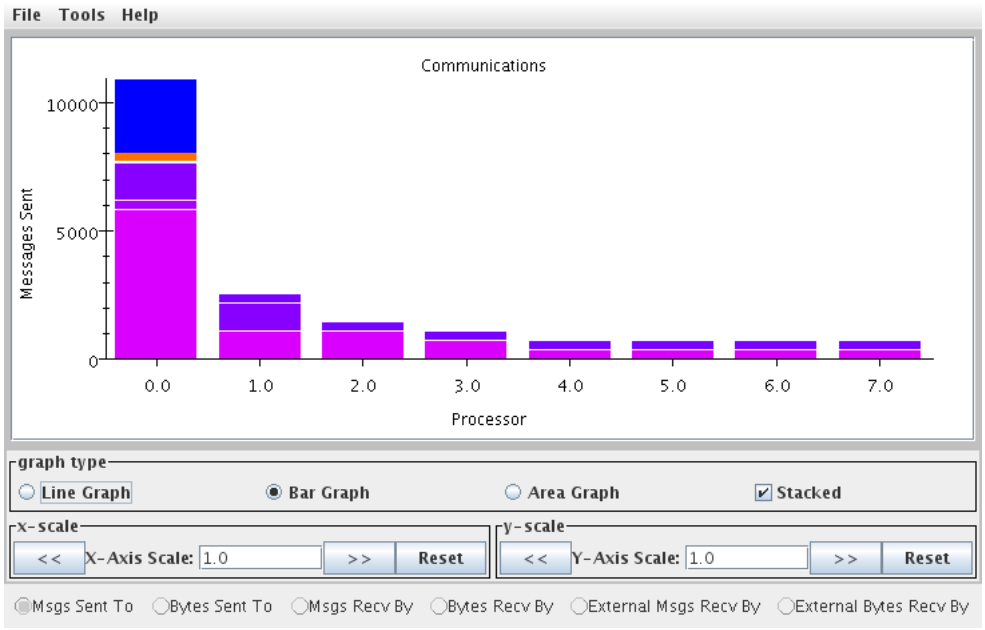


Figura 4.28: Gráfico de mensagens enviadas para cada processador

Capítulo 5

Conclusão

Apesar de existirem vários simuladores de interação da radiação com a matéria, não existe, na atualidade, nenhum que permita realizar a movimentação do sistema fonte-detector ou da amostra. Todos os simuladores atuais são voltados para sistemas fixos. Por este motivo optou-se por escrever um código de simulação específico para TC, utilizando uma biblioteca própria para programação paralela, uma vez que a utilização do método MC requer um grande período de simulação.

O propósito da criação e da utilização de um simulador de TC é avaliar algumas técnicas tomográficas quanto à sua aplicabilidade e resultados, sem que seja necessário recorrer a laboratórios que permitam realizar uma TC, pois, em geral o tempo disponível em laboratórios que forneçam a possibilidade de realizar a TC é escasso, a simulação do experimento pode antecipar algumas respostas ou indicar que certa técnica tomográfica é viável ou não. Além disso a simulação pode ajudar na construção de novos equipamentos.

O simulador de TC proposto, realiza a interação da radiação com a amostra de maneira coerente com a teoria e os resultados reconstruídos mostram que o simulador está interpretando os dados de entrada corretamente, o que inclui não só a forma geométrica da amostra como também os diferentes elementos químicos de que a amostra é composta.

Apesar do simulador de TC ter sua utilização voltada principalmente para um am-

biente de processamento paralelo, o tempo de simulação não se torna um problema, pois esses ambientes permitem diminuir sensívelmente o tempo de simulação. Uma das vantagens da programação em CHARM++ é a utilização do simulador tanto, em sistemas distribuídos como em sistemas *stand alone* sem necessitar modificação no código do simulador e sem alteração nos resultados obtidos, a única diferença será no tempo de simulação.

As figuras de linha de tempo mostram que, apesar do compartilhamento de processadores, o simulador está bem balanceado sem ter nenhum gargalo em sua estrutura. O sistema de balanceamento utilizado no simulador proposto é o sistema padrão do Charm++, que se mostou aduequado para simulação MC de um TC. Isso pode ser visto através do gráfico de *speed up* (figura 4.24). Já a queda de performance, com 15 e 16 nós, pode ser explicada pelo o compartilhamento de processadores, isto também pode ser visto nas linhas de tempo.

Apesar do simulador de TC estar gerando dados coerentes ainda necessita a realização de algumas melhorias, como um criador e visualizador de amostras mais amigável e que consiga formar amostras mais complexas que as atuais, simular completamente o efeito fotoelétrico e também a simulação de detecção de radiações secundárias (difração de raios-X). Para imagens com maiores resoluções (4Mega pixels, por exemplo) é necessário realizar um estudo de espaço em disco rígido ou adcionar um algoritimo compactação de imagens ao simulador.

Bibliografia

[1] "A computer code to simulate X-ray imaging techniques", Duvauchelle P., Freud N., Kaftandjian V., Babot D., Nuclear Instruments and Methods in Physics Research B v.170 (2000) 245-258

[2] "Monte Carlo simulation of X-ray spectra in diagnostic radiology and mammography using MCNP4C", Manual de utilização, Março 1997

[3] "Monte Carlo simulations of a high-resolution X-ray CT system for industrial applications", A. Miceli, R. Thierry, A. Flisch, U. Sennhauser, F. Casali, M. Simon, Nuclear Instruments and Methods in Physics Research A 583 (2007) 313-323

[4] "CTmodA toolkit for Monte Carlo simulation of projections including scatter in computed tomography", Malusek, A., Sandborg, M., Carlsson, G. A., Computer Methods and Programs in Biomedicine (2008) 167-178

[5] "A microCT X-ray head model for spectra generation with Monte Carlo simulations", R. Taschereau, P.L. Chow, J.S. Cho, A.F. Chatziioannou, Nuclear Instruments and Methods in Physics Research A 569 (2006) 373-377

[6] "A wide-beam X-ray source suitable for diffraction enhanced imaging applications", Chang H. Kim, Mohamed A. Bourham, J. Michael Doster, Nuclear Instruments and Methods in Physics Research A 566 (2006) 713-721

[7] Radiation Detection and Measurement, Knoll, Glenn F. - 1ª ed - 1979

[8] "Imaging modalities in x-ray computerizer tomography and in selected volume tomography", Carlsson, Carl A., Nuclear Instruments and Methods in Physics Research A 566 (2006) 713-721

[9] Elements of x-ray diffraction, Cullity, B. D. - 1ª ed. - 1956

[10] "Relevance of accurate Monte Carlo modeling in nuclear medical imaging", Zaidia, H., Medical Physics (1999) 574-608

[11] "Current status and new horizons in Monte Carlo simulation of X-ray CT scanners", Zaidi, H., Ay., M., R., Medical and Biological Engineering and Computing (2007) 809-817

[12] "Non-periodic pseudo-random numbers used in Monte Carlo calculations", Gaston E. Barberis, Physica B 398 (2007) 468-471

[13] "A Code System for Monte Carlo Simulation of Electron and Photon Transport", Francesc Salvat, José M. Fernández-Varea, Eduardo Acosta e Josep Sempau, Workshop Proceedings Issy-les-Moulineaux, France 5-7 (2001)

[14] National Institute of Standards and Technology - http://www.nist.gov/

[15] International Atomic Energy Agency - http://www.iaea.org/

[16] "Guia de estruturação e administração do ambiente de cluster e grid", *http* : *//guialivre.governoeletronico.gov.br/guiaonline/guiacluster/node*8*.php* (acessado em agosto de 2008)

[17] "Analytical Versus Voxelized Phantom Representation for Monte Carlo Simulation in Radiological Imaging", Peter J., Tornai M. P., Jaszczak R. J., IEEE Transactions on Medical Imaging, vol. 19, no. 5, (2000) 556-564

[18] "The Charm++ Programming Language Manual", Parallel Programming Laboratory University of Illinois at Urbana-Champaign

[19] National Library of Medicine, National Institute of Health - Estados Unidos, *http* : *//www.nlm.nih.gov/research/visible/visible_human.html* (acessado em fevereiro de 2009)

[20] "Analytical Versus Voxelized Phantom Representation for Monte Carlo Simulation in Radiological Imaging", Peter J., Tornai, M. P., Jaszczak, R. J. Fellow, IEEE Transactions on Medical Imaging, vol. 19, no. 5, (2000)

[21] "X-ray attenuation-coefficient measurements", McCrary, J. H., Plassmann, H. E., Puckett, J. M. Conner, A. L. - Physical review (1967) 307-312

[22] "A cone-beam tomography system with a reduced size planar detector: A backprojection-filtration reconstruction algorithm as well as numerical and practical experiments", Li L., Chen Z., Zhang L., Xing Y., Kang K., Applied Radiation and Isotopes 65 (2007) 1041-1047

[23] "Monte Carlo simulation of x-ray spectra in diagnostic radiology and mammography using MCNP4C", Ay M.R., Shahriari M., Sarkar S., Adib M., Zaidi H., Physics in Medicine and Biology, vol. 49 (2004) 4897-4917

[24] "Monte Carlo optimization of an industrial tomography system", Berdondinia A., Bettuzzia M., Bianconia D., Brancaccioa R., Casalia F., Cornacchia S., Flischb A., Hofmannb J., Lanconellia N., Morigia M.P., Pasinia A., Rossia A., Sauerweinc C., Simonc M., Nuclear Instruments and Methods in Physics Research A 580 (2007) 771-773

[25] "X-ray scatter data for flat-panel detector CT", Kyriakou Y., Kalender W. A., Physica Medica (2007) 3-15

[26] "Yet another application of the Monte Carlo method for modeling in the field of biomedicine", Cassia-Mouraa R., Sousab C.S., Ramosb A.D., Coelhoc L.C.B.B., Valençad, M.M., Computer Methods and Programs in Biomedicine (2005) 78, 223-235

[27] "Computed tomography simulation with superquadrics", Zhua J., Zhaob S., Yec Y., Wangd G., American Association of Physicists in Medicine (2005) 3136-3143

[28] "Spatial resolution properties in cone beam CT: A simulation study", Chen L., Shaw C. C., Altunbas M. C., Lai C-J., Liu X., Med. Phys. 35, (2008) 724-734