Uma Abordagem Amigável e Massivamente Paralela para Geração de Imagens Radiográficas Sintéticas e de Tomografias Computadorizadas Utilizando Simulação de Monte Carlo

Vagner da Silva de Jesus
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Rua Emídio dos Santos, S/N, Barbalho
Salvador, Bahia
vagnerdasilva94@gmail.com

Antonio Carlos Santos Souza
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Rua Emídio dos Santos, S/N, Barbalho
Salvador, Bahia
acsantossouza@gmail.com

RESUMO

Na atualidade, com o objetivo de reduzir os riscos ao realizar radiografias, muitas formas de realizar tais simulações acabaram sendo criadas. Ao utilizar o poder de processamento resultante dos avanços tecnológicos atuais, alternativas fazendo uso de paralelismo surgiram. Entretanto, algumas dessas alternativas não acompanharam apropriadamente a evolução das ferramentas que foram utilizadas na sua concepção, resultando em erros e conflitos gerados pela incompatibilidade de certas ferramentas atuais. Em razão disso, neste trabalho encontram-se otimizações e melhorias no software MC-GPU com o intuito de resolver os problemas encontrados ao começar seu processo de instalação e utilização, aliado a uma abordagem amigável e menos suscetível a erros para utilização da ferramenta.

Palavras-chave

Radiografia, Monte Carlo, Paralelismo, CUDA, MC-GPU.

1. INTRODUÇÃO

Com os contínuos avanços tecnológicos contribuindo para medicina atual, todo o apoio prestado gera uma evolução enorme e contínua em todo o processo de assistência médica. Nesse âmbito, os exames preventivos se destacam, sendo de grande importância por possibilitar que diagnósticos mais precisos e assertivos sejam realizados, propiciando maior agilidade e bem estar tanto ao corpo médico responsável quanto ao paciente.

Para a realização de diagnósticos, os exames de imagem são um dos processos mais utilizados, possuindo oito tipos de procedimentos distintos, estes podendo ser classificados em dois tipos: os que utilizam radiação ionizante e os que utilizam radiação não-ionizante[1]. Dentre os que utilizam radiação ionizante temos a radiografia.

A radiologia é uma área da medicina que se dedica ao estudo e aplicação das radiações ionizantes para diagnóstico e tratamento de diversas doenças. Através do uso de técnicas de imagem, é possível visualizar estruturas internas do corpo humano, identificar lesões, fraturas e outras anomalias, auxiliando no diagnóstico e planejamento de tratamentos. Dada sua ampla aplicação, a radiologia é uma especialidade multidisciplinar que envolve físicos, médicos e tecnólogos em radiologia, sendo essencial para a prática médica em diversas especialidades, como oncologia, neurologia, cardiologia, entre outras.

Nos últimos anos, a radiologia tem passado por uma grande transformação com o avanço da tecnologia, em especial as ligadas a computação. O uso de sistemas digitais de aquisição de imagens têm permitido maior precisão e eficiência na obtenção de imagens radiográficas, com menor exposição à radiação ionizante. Além disso, técnicas computacionais têm sido cada vez mais utilizadas na área, permitindo a realização de simulações e análises de imagens com maior precisão e rapidez.

Entre as técnicas computacionais utilizadas na radiologia, podemos destacar a tomografia computadorizada (TC), que permite a obtenção de imagens tridimensionais com alta resolução espacial, e a ressonância magnética (RM), que utiliza campos magnéticos e ondas de rádio para gerar imagens de alta definição dos tecidos e órgãos internos. Ambas as técnicas utilizam algoritmos computacionais para reconstrução de imagens, processamento e análise de dados.

Outra técnica computacional utilizada na radiologia é a simulação de radiação, que é de extrema importância na terapia de câncer. A técnica de Monte Carlo é utilizada para simular as interações entre a radiação e os tecidos do corpo humano, permitindo a criação de modelos precisos que ajudam os médicos a determinar as doses de radiação ideais para o tratamento de câncer. Essa técnica é amplamente utilizada na radioterapia, sendo essencial para

garantir a eficácia do tratamento e minimizar os efeitos colaterais.

Dentre as aplicações que fazem uso da técnica de Monte Carlo para simulação de radiação, destaca-se o MC-GPU por utilizar computação massivamente paralela e possuir código livre. Entretanto, por possuir muito tempo sem atualizações, sua instalação ocorre com diversos erros e inconsistências, além de basicamente funcionar puramente por linha de comando. Considerando este apelo, o presente trabalho objetiva propor uma abordagem amigável para utilização da ferramenta, desde correções e otimizações em sua instalação e código fonte até uma interface gráfica para melhor manipulação do parâmetros necessários para realizar uma simulação.

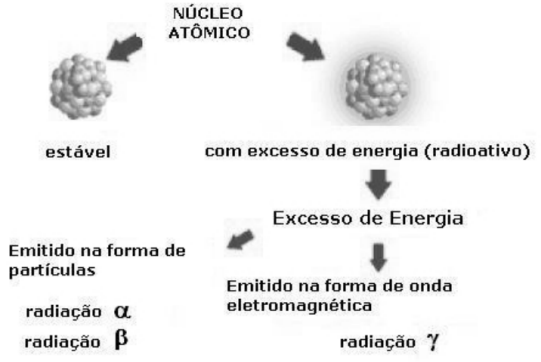

Figura 1: Ilustração da radioatividade[24]

2. REFERENCIAL BIBLIOGRÁFICO

Nesta seção é apresentada as principais referências usadas para compreensão dos temas correlacionados com o problema e o desenvolvimento das soluções propostas. Sua organização está dividida de tal forma: a subseção 2.1 apresenta conceitos de radiografia, na subseção 2.2 explana uma breve explicação sobre o Método de Monte Carlo, a subseção 2.3 aborda sobre paralelismo e suas possíveis aplicações.

2.1. Radiografia

Radiação é energia que se propaga a partir de uma fonte emissora através de qualquer meio, podendo ser classificada como energia em trânsito[7].

Ela se apresenta em forma de partícula atômica ou subatômica energética, que podem ser produzidas de forma artificial, como em aceleradores de partículas ou em reatores, quanto natural, a partir do decaimento dos átomos radioativos.

A radiação pode se apresentar também em forma de onda eletromagnética, caracterizada pelo comprimento de onda ou frequência da onda. Existem vários níveis de ondas que constituem o espectro eletromagnético, indo desde ondas com frequência extremamente baixa como a presente em ondas de rádio e TV, até o outro extremo onde reside os raios X e raios gama.

A radiação não ionizante é uma modalidade de radiação de baixa frequência e baixa energia, possuindo frequência igual ou menor que a faixa de frequência da luz visível. Este tipo de radiação possui energia suficiente para mover ou vibrar átomos, sendo incapaz de remover elétrons de sua órbita[8].

As radiações não ionizantes estão presentes em diversas situações do dia a dia das pessoas, como ondas de rádio, ondas de TV, fontes de calor e a própria luz.

A radiação ionizante é aquela que possui energia suficiente para remover elétrons da órbita do átomo, assim criando íons. Os íons que são criados de tal forma podem ser detectados a partir da radiação que emitem.

A radiação ionizante inclui os raios-X, raios gama, raios cósmicos, partículas alfa, partículas beta e nêutrons. Cada tipo de radiação possui diferenças no grau de energia e capacidade de penetração nas células do corpo.

O dano dessa penetração pode ser direto no DNA da célula indireto pela produção de radicais livres (nota de rodapé).

As fontes de radiação ionizantes podem ser naturais ou não naturais. Dentre as naturais podem ser citadas as rochas, solos e águas que possuem elementos radioativos em baixas quantidades em sua composição. Também é possível citar os raios cósmicos vindos do espaço que bombardeiam a atmosfera terrestre a todo momento. Para as não naturais, a exemplo temos os equipamentos de diagnóstico radiológico como de raios-X e tomografia computadorizada (TC).

Dentre os benefícios da radiação ionizante pode-se destacar a geração de energia para utilização em diversos processos, tanto na indústria quanto na medicina. No âmbito da medicina, ela é muito útil tanto no tratamento de doenças, como a radioterapia, quanto no seu diagnóstico, utilizando radiografia.

De forma simplificada, a radiografia é um exame diagnóstico que utiliza radiação ionizante para produzir imagens das estruturas internas do corpo[2]. É um dos

exames mais comuns na área médica, usado para diagnosticar uma ampla gama de condições médicas.

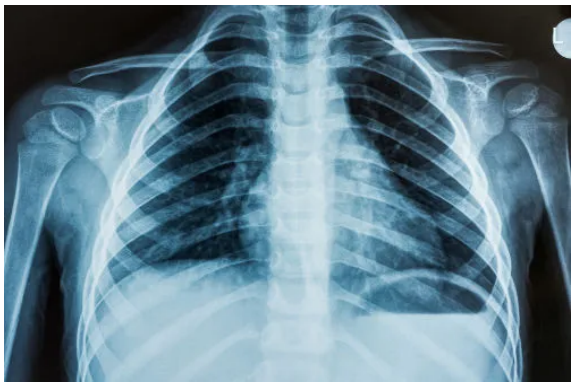

Figura 2: Imagem gerada pela radiografia do tórax[25]

A técnica da radiografia envolve a emissão de um feixe de raios-x através do corpo do paciente. Quando esse feixe atinge os tecidos corporais, algumas partes são absorvidas e outras são transmitidas. Um detector captura os raios-x que passam através do corpo, criando uma imagem digital em preto e branco que representa a densidade dos tecidos. Os ossos aparecem brancos porque absorvem mais raios-X, enquanto os tecidos moles, como músculos e órgãos, aparecem mais escuros.

A tomografia computadorizada também utiliza raios-X, criando imagens de alta resolução espacial de todo o corpo humano. Quanto maior o número de lâminas(rodapé) utilizadas, melhor será a resolução da imagem gerada.

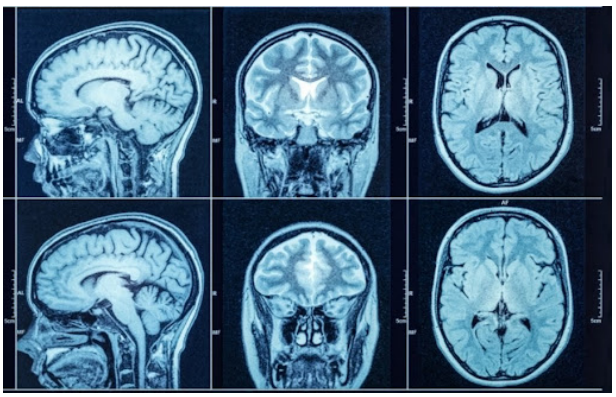

Figura 3: Imagem gerada pela tomografia computadorizada de um crânio[26]

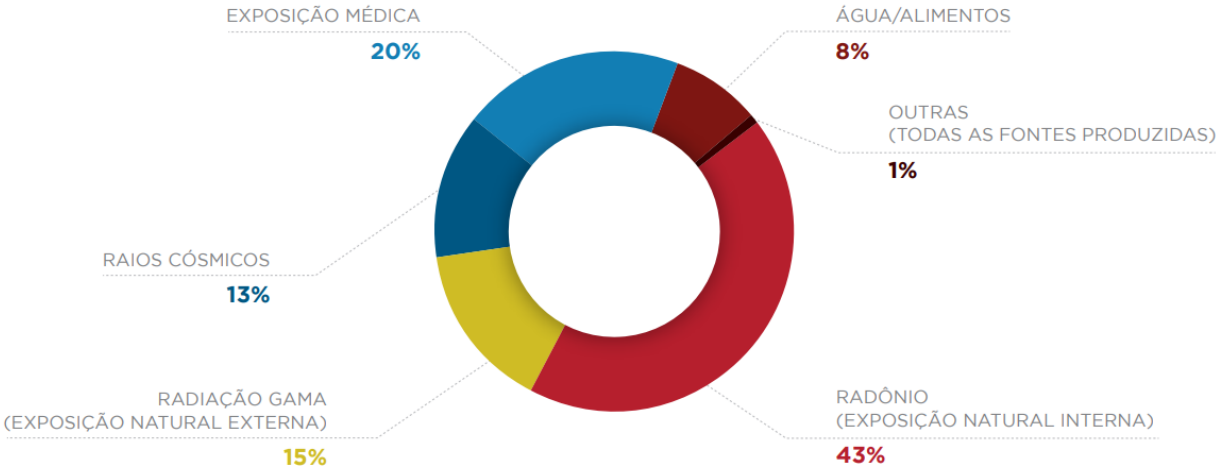

Figura 4: Distribuição das fontes de exposição[8]

A radiografia é usada para diagnosticar uma variedade de condições médicas, desde problemas ósseos até lesões internas nos órgãos. As radiografias de ossos são muito úteis para detectar fraturas, tumores ósseos e deformidades, enquanto as radiografias de tórax são frequentemente usadas para diagnosticar problemas pulmonares, como pneumonia ou enfisema. Além disso, as radiografias também são usadas para diagnosticar problemas cardíacos, como doença arterial coronariana.

Embora a radiografia seja um exame amplamente utilizado, ela apresenta alguns riscos. A exposição a altas doses de radiação ionizante pode ser prejudicial à saúde humana, por isso, é importante conhecer as fontes e modelos de radiação para que se possa avaliar e controlar os riscos associados.

As fontes de radiação ionizante podem ser divididas em duas categorias: fontes naturais e fontes não naturais[3]. Fontes naturais incluem raios cósmicos e no próprio corpo humano. As fontes não naturais incluem a radiação proveniente de aparelhos de raio-x e usinas nucleares.

Os modelos das fontes de radiação ionizante variam dependendo do tipo de fonte. Para fontes externas, os modelos levam em consideração a energia e o tipo de partícula da radiação, bem como a distância entre a fonte e o alvo. Para fontes internas, os modelos consideram o tipo

de material radioativo e sua concentração no corpo humano.

Para simular o comportamento da radiação ionizante, os cientistas utilizam materiais especiais que imitam as propriedades dos tecidos humanos e outros materiais biológicos. Dentre esses materiais temos água e tecidos humanos, sejam reais ou sintéticos. Esses materiais são frequentemente utilizados em estudos de dosimetria, onde a quantidade e a distribuição da radiação são medidas em diferentes partes do corpo humano.

Os materiais utilizados em simulações de radiação ionizante devem possuir propriedades semelhantes às dos tecidos humanos para que possam ser utilizados como simuladores. O tecido humano é composto principalmente por água, proteínas e gorduras, por isso, os materiais utilizados em simulações devem ter densidade, composição química e propriedades físicas semelhantes às dos tecidos humanos.

2.2. Método de Monte Carlo

O Método de Monte Carlo é baseado em simulações aleatórias para estimar quantidades estatísticas[4]. Ele usa uma série de números aleatórios para modelar eventos que ocorrem em um sistema e então analisa os resultados para obter uma solução aproximada do problema. Esse método é particularmente útil em situações em que as soluções analíticas exatas são muito difíceis ou impossíveis de serem encontradas.

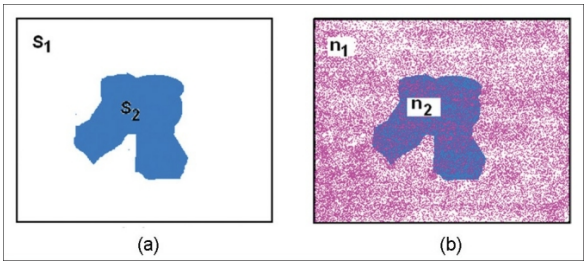

Figura 5: Um exemplo da técnica de Monte Carlo para cálculo de área de uma forma irregular[4]

O primeiro registro de uso prático do Método de Monte Carlo é geralmente o relacionado ao conde de Buffon, em 1777. Nele foi proposto um método semelhante ao de Monte Carlo com o objetivo de avaliar a probabilidade de lançar uma agulha em uma folha pautada.

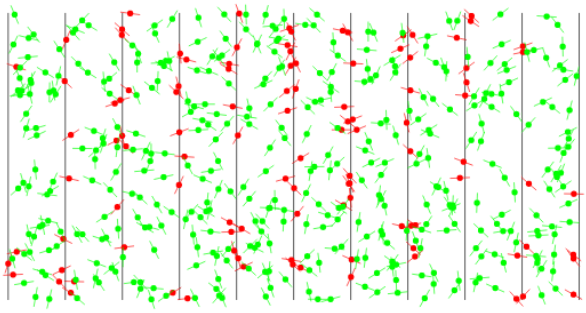

Figura 6: Exemplo de um experimento de Buffon onde foram lançados 500 agulhas, onde as vermelhas cruzam uma linha e as verdes não cruzam[27]

O método recebeu este nome em homenagem à cidade de Monte Carlo, em Mônaco, conhecida por seus cassinos. O processo é baseado em simular aleatoriamente a realização de um grande número de eventos para estimar a probabilidade de um resultado específico ocorrer. A ideia é repetir um experimento diversas vezes com diferentes valores aleatórios para obter uma amostra grande o suficiente para calcular a probabilidade do evento em questão.

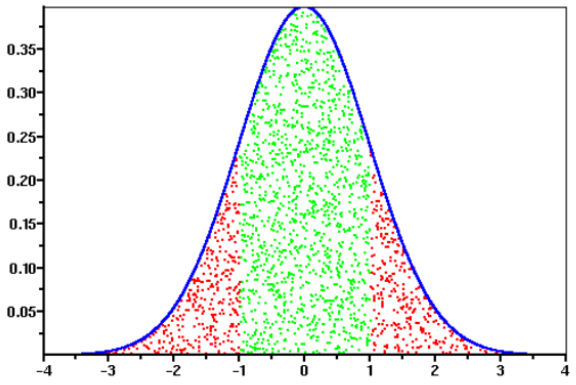

Figura 7: Cálculo da estimativa de uma área (verde) com o Método de Monte Carlo, utilizando 2000 pontos[28]

Na física, o Método de Monte Carlo é frequentemente usado para simular sistemas físicos complexos, como moléculas, átomos e cristais. Essas simulações são usadas para prever propriedades físicas e químicas de materiais, como condutividade elétrica, reatividade química e propriedades ópticas.

Outra vantagem do Método de Monte Carlo é sua versatilidade. Ele pode ser usado para resolver problemas em uma ampla variedade de campos, desde a física teórica até a engenharia de sistemas complexos. Além disso, ele é facilmente escalável, permitindo que os usuários ajustem a precisão dos resultados de acordo com as suas necessidades.

Na simulação de Monte Carlo para o transporte de radiação, a história (da trajetória) de uma partícula é visto

como uma sequência aleatória de lançamentos livres que terminam com um evento de interação em que a partícula altera sua trajetória, perde energia e, ocasionalmente, produz partículas secundárias[9].

Ao realizar uma simulação de Monte Carlo em um determinado arranjo experimental, é necessária a geração numérica de histórias aleatórias. Para simular essas histórias é necessário um modelo de interação que determina as interações relevantes. Essas interações determinam as funções de distribuição de probabilidade das variáveis aleatórias que caracterizam uma trajetória:

- Caminho livre entre sucessivos eventos de interação;
- Tipo de interação que está ocorrendo;
- Perda de energia e ângulo de deflexão em um evento particular (e o estado inicial de partículas secundárias emitidas caso exista).

Com as funções de distribuição de probabilidade conhecidas, as histórias aleatórias podem ser geradas. Se gerado um número de histórias grande o suficiente para essa simulação, as informações sobre o processo de transporte podem ser obtidas basicamente calculando a média sobre as histórias geradas nas simulações[9].

(Falar sobre técnicas de redução de variância)

No mundo real, o processo de colisão de um elétron acarreta em um número muito grande de interações inelásticas e elásticas, tornando impossível simular explicitamente cada colisão de elétrons. Dessa forma, ao realizar os cálculos com Monte Carlo, as colisões elásticas são representadas por mudanças de direção a cada passo do elétron em sua movimentação. Já as colisões inelásticas são agrupadas em um processo discreto(rodapé), com perda contínua de energia, chamado de simulação de histórico condensado.

A técnica de histórico condensado foi desenvolvida por Martin J. Berger. Sua premissa considera que grandes números de processos de transporte e colisão são “condensados” em uma única etapa de elétrons[12]. O efeito cumulativo das interações leva em consideração a mudança da energia e direção de movimento da partícula no final da etapa. Essa abordagem tem como base o fato de que as colisões simples com átomos resultam, em sua maioria, apenas pequenas mudanças na energia e trajetória da partícula.

Na Figura 8 é possível observar uma representação do algoritmo de histórico condensado. A trilha do elétron (acima) inclui muitos elétrons e fótons secundários (linha tracejada no painel superior). A trilha do elétron na caixa sombreada é simulada com um algoritmo de histórico condensado. As posições inicial e final do elétron em uma etapa são A e B, que incluem o espalhamento do elétron no meio. No entanto, a implementação do histórico condensado não fornece informações sobre como a partícula vai de A a B. A linha tracejada curva conectando A e B é uma representação mais realista da trajetória do que uma linha reta de A a B[4].

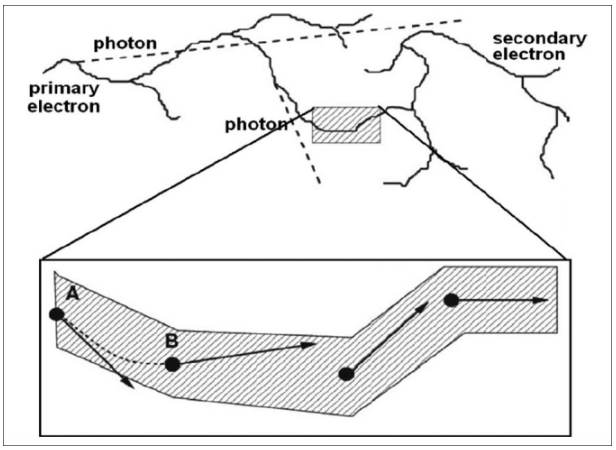

Figura 8: Ilustração do algoritmo de histórico condensado[4]

Apesar de ser uma técnica poderosa, o método de Monte Carlo tem suas limitações. Como seu uso reside na natureza aleatória, um grande número de amostras são requeridas para reduzir sua incerteza estatística. Com o aumento do número de amostras, a simulação torna-se muito demorada ou muito complexa, exigindo grandes quantidades de memória e processamento de dados. Além disso, a precisão da estimativa pode ser afetada por erros na modelagem matemática ou pela escolha inadequada de valores aleatórios.

2.3. Paralelismo

O paralelismo é uma técnica usada para aumentar o desempenho dos computadores, permitindo que dois ou mais processos sejam executados simultaneamente[5]. Isso é especialmente útil em tarefas que exigem muito processamento, como a renderização de gráficos 3D, simulações físicas, mineração de dados e outras aplicações de alto desempenho.

Uma das principais vantagens do paralelismo é o aumento do desempenho do sistema. Quando várias tarefas são executadas simultaneamente, a carga de trabalho é distribuída por vários processadores ou núcleos, resultando em uma redução significativa no tempo de processamento. Além disso, o paralelismo permite a execução de tarefas que, de outra forma, seriam impossíveis de executar em um tempo razoável.

É possível fazer uso de técnicas de paralelismo utilizando tanto a CPU (unidade de processamento central), presente na grande maioria dos computadores e notebooks, quanto GPU (unidade de processamento gráfico). A GPU foi criada para ser um processador auxiliar com o objetivo de realizar o processamento gráfico e reduzir a quantidade de processamento lógico aritmético realizado pela CPU[10].

Ao analisar a arquitetura de uma CPU e uma GPU, é notável que o núcleo de controle da CPU é bem desenvolvido enquanto as unidades de lógicas aritméticas (ALU) são menores. Em contrapartida, na GPU há poucos e menores núcleos de controle e possui muitas unidades de lógicas aritméticas.

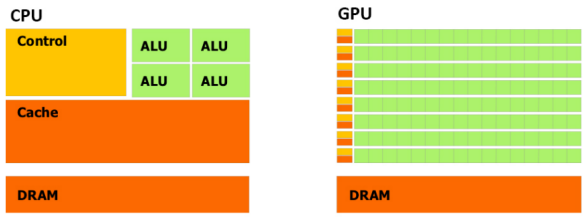

Figura 9: Arquitetura básica de CPU e GPU[10]

Tais arquiteturas utilizadas resultam em CPUs sendo dispositivos otimizados para gerenciamento paralelo de tarefas com pouca eficiência na realização de operações aritméticas. Por outro lado, as GPUs têm maior eficiência ao realizar operações aritméticas, possuindo pouca capacidade em relação ao processamento de múltiplas tarefas.

Visando se aproveitar dessa capacidade das GPUs, surgiram novas tecnologias com o objetivo de utilizar de forma eficiente toda essa capacidade de processamento nas mais diversas aplicações, como aprendizado de máquina, simulações e análises de big data. Uma das tecnologias mais populares para tal fim é a NVIDIA CUDA.

CUDA (Compute Unified Device Architecture) é uma plataforma de computação paralela desenvolvida pela NVIDIA, que permite que os programadores usem a unidade de processamento gráfico (GPU) para acelerar o processamento de dados[6].

A principal vantagem do uso de CUDA é que ele permite que os desenvolvedores aproveitem a arquitetura paralela das GPUs NVIDIA, que tem milhares de núcleos de processamento. Isso significa que os desenvolvedores podem acelerar o processamento de tarefas complexas em ordens de magnitude em comparação com a execução em uma CPU tradicional.

CUDA pode ser programado em diversas linguagens de programação, tais como C, C++, Fortran, Python e MATLAB[6], cada uma possuindo comando e bibliotecas próprias e otimizadas. Cada GPU possui um grupo de processadores compostos por múltiplos núcleos CUDA, fazendo com que cada GPU tenha uma quantidade intrínseca de núcleos CUDA atrelada a ela, não podendo ser alterado pelo código.

A plataforma CUDA é usada em muitas áreas, incluindo pesquisa científica, análise de dados, processamento de imagens e jogos. Por exemplo, na pesquisa científica, a plataforma CUDA é usada para simulações de física de partículas, modelos climáticos e análise de genoma. Na análise de dados, a plataforma é usada para análise de big data, mineração de dados e aprendizado de máquina.

3. TRABALHOS RELACIONADOS

Nesta seção serão apresentados alguns trabalhos que tratam de temas similares ao da aplicação a ser modificada, realizaram alterações em um software existente para ter alguma vantagem em relação à aplicação base. Para isso, foram abordadas algumas aplicações que realizaram modificações próprias para situação proposta.

3.1. PENELOPE

PENELOPE é uma aplicação de simulação de radiação desenvolvida pela Organização Europeia para a Pesquisa Nuclear (CERN) em 1996, contando com a colaboração da Comissão Europeia[9]. A ferramenta é capaz de simular o transporte de partículas carregadas e fótons em materiais complexos, como os encontrados em detectores de radiação, materiais biológicos e materiais nucleares.

Seu nome é gerado do acrônimo de PENetration and Energy LOss of Positrons and Electrons (Penetração e Perda de Energia de Pósitrons e Elétrons)[9]. Em atualizações posteriores foi introduzido a simulação de fótons em seu leque de opções.

A simulação de radiação é essencial para muitas áreas da pesquisa em física, medicina, segurança nuclear e outras. No entanto, a simulação precisa ser muito precisa, pois erros podem levar a resultados imprecisos e a consequências graves, como a exposição excessiva à radiação.

O PENELOPE utiliza o Método de Monte Carlo para realizar simulações de transporte de partículas carregadas e fótons em materiais. Esse método é baseado em amostragem estatística para calcular as trajetórias das partículas. Ele é capaz de simular interações complexas entre partículas e materiais, incluindo o espalhamento de elétrons, a ionização e a excitação de átomos.

A aplicação é capaz de simular interações de radiação em uma ampla gama de materiais, incluindo plásticos, líquidos, metais e tecidos biológicos, tornando a ferramenta muito útil para uma variedade de aplicações, como modelagem de dosimetria em radioterapia, estudos de segurança nuclear, modelagem de detectores de radiação, entre outros.

O PENELOPE foi desenvolvido em colaboração com a Comissão Europeia, o qual é disponibilizado gratuitamente por possuir uma licença de software livre. Isso torna a ferramenta acessível a pesquisadores de todo o mundo, ajudando a promover a colaboração internacional e o compartilhamento de conhecimento.

Uma das principais vantagens do PENELOPE é sua capacidade de simular interações de radiação de baixa energia, possuindo uma escala que tange de algumas

centenas de eV a, aproximadamente, 1 GeV(rodapé). Isso é importante para aplicações em medicina, onde a radiação de baixa energia é comum. A ferramenta também é capaz de simular a resposta de detectores de radiação em ambientes complexos, o que é útil para aplicações em segurança nuclear.

O PENELOPE também tem sido usado em estudos de dosimetria em radioterapia. A ferramenta é capaz de simular o transporte de partículas ionizantes em tecidos biológicos, o que permite que os pesquisadores estudem a dose absorvida por tecidos específicos durante a terapia.

A precisão das simulações do PENELOPE é uma das suas principais vantagens. A ferramenta é capaz de simular interações de radiação com alta precisão, sendo de extrema importância tal precisão para muitas aplicações em física, medicina e segurança nuclear. No entanto, a precisão vem com um custo computacional, tornando a simulação demorada e exigindo grandes recursos de computação.

Muitas proposições são feitas com o propósito de paralelizar seu código de modo a utilizar de forma mais eficiente os recursos todo o poder computacional que se possui atualmente.

3.2. GEANT4

GEANT4 é uma aplicação de simulação de partículas desenvolvida pelo CERN (Organização Europeia para a Pesquisa Nuclear) que permite simular o transporte e interação de partículas na área de física de partículas, astronomia e medicina nuclear[11].

Possuindo uma licença de software livre, o GEANT4 foi desenvolvido pela colaboração GEANT4, uma parceria internacional de instituições acadêmicas e de pesquisa. O software foi desenvolvido para simular a interação de partículas com a matéria e é capaz de modelar uma ampla variedade de processos físicos, incluindo interações com elétrons, fótons e núcleos.

Uma das principais aplicações do GEANT4 é a simulação de detectores de partículas, que são dispositivos usados para medir a radiação ionizante. A simulação do comportamento dos detectores de partículas é crucial para interpretar as medidas experimentais e otimizar o projeto dos detectores. Adicionalmente, o GEANT4 também é usado em simulações de radiação em terapia por radiação, onde é necessário calcular com precisão a dose de radiação depositada no tecido biológico.

O software também é usado em aplicações de segurança nuclear, como a simulação do transporte de radiação em ambientes nucleares. Essas simulações podem ser usadas para avaliar os riscos à saúde humana e ao meio ambiente em caso de acidentes nucleares. Além disso, o GEANT4 é usado para simular a radiação cósmica que os astronautas experimentam durante as missões espaciais.

O GEANT4 é um software avançado que requer habilidades técnicas para ser utilizado. No entanto, existem vários recursos disponíveis para ajudar os usuários a aprender como usar o software. A documentação oficial do GEANT4 contém tutoriais e exemplos que podem ser usados como ponto de partida para novos usuários.

3.3. PenRed

PenRed (Parallel ENgine for Radiation Energy Deposition) é uma aplicação criada com uma estrutura feita para ser modular, personalizável e totalmente em paralelo para simulações de Monte Carlo para passagem de radiação através da matéria[13]. Sendo baseado no código do PENELOPE, ele herda seus modelos físicos únicos e algoritmos de rastreamento para partículas carregadas.

O PenRed foi desenvolvido na linguagem de programação C++, seguindo o paradigma de programação orientada a objetos. Seu mecanismo de paralelismo foi implementado seguindo duas abordagens: threads com C++ e MPI. Com C++ é usado threads para acessar a memória compartilhada do sistema, melhorando seu acesso e uso de memória. O MPI foi implementado para infraestruturas que fazem uso de memória distribuída. Essas duas implementações podem ser combinadas ao realizar uma mesma simulação, inclusive podendo realizar balanceamento de carga (rodapé) para maximizar o desempenho.

Outro ponto de vantagem é o fato do PenRed fornecer uma estrutura modular em sua codificação que o torna facilmente estendível, possibilitando ao usuário a criação de módulos e adaptações para atender suas necessidades sem perder a vantagem do paralelismo já incorporado na aplicação. Esses módulos são fornecidos como interfaces e classes abstratas, incluindo toda a biblioteca presente no PENELOPE.

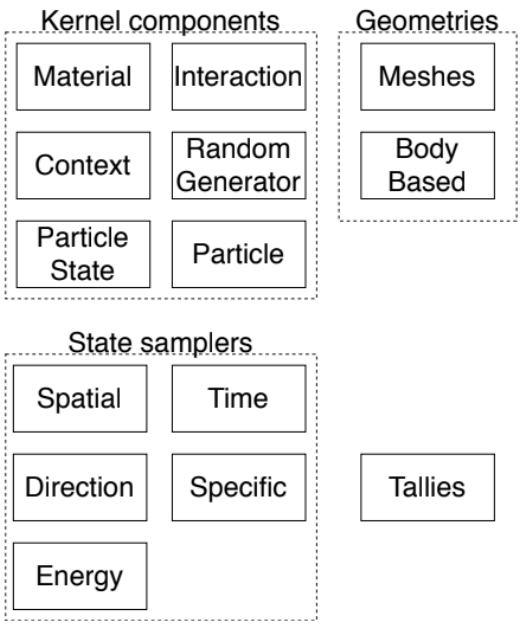

Figura 10: Classificação dos módulos do PenRed[13]

A implementação de paralelismo em uma aplicação demanda um conhecimento muitas vezes avançado, podendo gerar limitações do quanto o usuário pode alterar o código da aplicação. Diante disso, o PenRed foi estruturado de modo que o usuário possa escrever seus módulos de modo sequencial e o software os executará utilizando multithreading e multiprocessos(rodapé).

Para conseguir esse resultado, o PenRed limita o que os membros de uma função podem alterar no estado de uma classe. A outra alternativa seria duplicar toda a simulação, incluindo geometria, amostras, etc. Entretanto, isso geraria uma enorme quantidade de memória sendo duplicada, aumentando as chances de falhas de memória[13].

3.4. CUBMC

CUBMC (CUDA Based Monte Carlo) é um software desenvolvido usando como base o Método de Monte Carlo, sendo destinado a simulações de transporte de fótons em objetos voxelizados. Tem como objetivo utilizar a capacidade de processamento paralelo das GPUs para acelerar as simulações utilizando máquinas menores e de baixo custo[10].

Sua primeira versão foi concebida para simular o transporte de fótons em uma geometria de voxels. Voxel é uma extrapolação de três dimensões do pixel[10]. Tendo como premissa que um conjunto de quadrados (ou pixels) podem formar uma imagem bidimensional, um conjunto de paralelepípedos (ou voxels) podem gerar uma geometria tridimensional. Tal forma de descrever a geometria foi escolhida com o intuito de descrever geometrias mais complexas de forma facilitada.

Ele utiliza como base o código presente no PENELOPE. Como o PENELOPE é um software muito robusto, foram feitas simplificações para restringir o escopo do programa ao transporte de fótons.

Essa decisão busca facilitar a descrição de modelos anatômicos gerados por imagens de Tomografia Computadorizada (TC) ou Ressonância Magnética (RM), podendo, com certa facilidade, serem transportadas para um modelo de voxels.

Ao realizar um comparativo dos tempos de simulação realizados no CUBMC utilizando um GPU GTX 560Ti e no PENELOPE utilizando um Intel Core i7-3520M, foi observado um ganho de desempenho significativo, com médias que variam entre 15 e 70 vezes dependendo do modo utilizado.

	PENELOPE	CUBMC			
		Woodcock	Ganho	Paradas	Ganho
ÁGUA	10051s	121s	83x	751s	13x
PULMÃO	9839s	210s	47x	684s	14x
OSSO	36030s	478s	75x	2349s	15x

Figura 11: Comparativo entre os tempos de execução para simulações realizadas com o código PENELOPE e CUBMC[10]

3.5. MC-GPU

MC-GPU é um código de simulação de Monte Carlo que pode gerar imagens sintéticas de radiografia e tomografia computadorizada de modelos realistas da anatomia humana fazendo uso do poder computacional presente nas GPUs[14]. Seu código implementa um algoritmo de simulação de Monte Carlo massivamente paralelizado para realizar simulações de transporte de raios-X em uma geometria voxelizada. O código-fonte do MC-GPU é um software livre de domínio público, tendo como principal desenvolvedor da aplicação Andreu Badal da U.S. Food and Drug Administration.

Para realizar o processo de paralelização, o MC-GPU foi desenvolvido usando o modelo de programação NVidia CUDA[6] para poder obter todo o desempenho que as GPUs NVidia oferecem. Adicionalmente, o código também pode ser compilado com um compilador da linguagem de programação C padrão para ser executado em uma CPU normal.

Ao simular as interações atômicas que ocorrem na aplicação, é utilizado o banco de dados de propriedades dos materiais do PENELOPE. Dentre seus arquivos que são disponibilizados, está um utilitário codificado na linguagem de programação Fortran[15] que possibilita a conversão de um arquivo de material do PENELOPE 2006 para um compatível com o MC-GPU.

O código fornece duas opções de contagem: contagem de imagem e contagem de dose de radiação. Na contagem de imagem são criadas imagens de projeção de raios-X, formadas pela contagem da energia que entra em uma grade definida pelo usuário, servindo de aproximação simples para um detector de painel plano(rodapé). Ao finalizar a simulação, são geradas quatro imagens:

- Raios-X que não interagiram entre a fonte e o detector (não dispersos/non-scattered);
- Raios-X que sofreram uma única interação Compton (inelástica/inelastic);
- Raios-X que sofreram uma única Interação de Rayleigh (elástica/elastic);
- Raios-X muiltiespalhados (multiple-scatter).

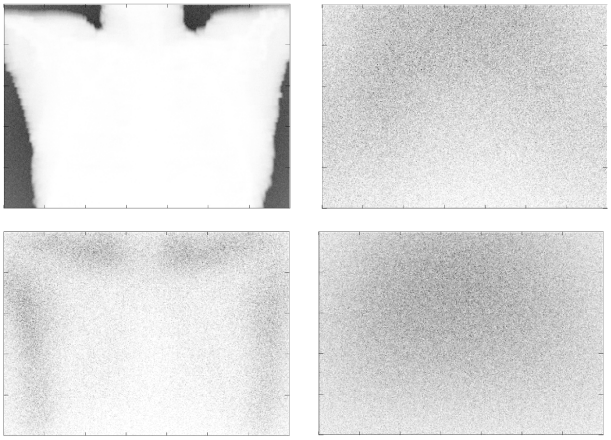

Figura 12: Imagens geradas a partir do voxel de Zubal Phantom[16] (Non-Scattered, Compton, Rayleigh, Multiple-Scatter) [própria autoria]

A contagem de dose de radiação conta a energia depositada por cada faixa de raio-X dentro de cada voxel da geometria. Isso ocorre dentro de uma região volumétrica de interesse previamente definida pelo usuário. Ao finalizar a simulação, a dose média que foi depositada em cada voxel e em cada material, juntamente com suas incertezas estatísticas associadas, são reportadas.

4. SOLUÇÃO DESENVOLVIDA

4.1. Tecnologias utilizadas

No desenvolvimento da solução foram utilizadas algumas tecnologias durante as diferentes etapas de sua construção, cada uma com suas particularidades que foram necessárias ao longo do processo. Para realizar o desenvolvimento da interface foi utilizado o framework Qt [17] em conjunto com o ambiente de desenvolvimento integrado (IDE) Qt Creator [23]. Para realizar o versionamento do que foi desenvolvido foi utilizado a plataforma Git. A seguir é discorrida uma breve descrição de cada uma das tecnologias.

4.1.1 Qt

Qt é um framework escrito em C++ para o desenvolvimento para aplicações tanto desktop quanto embarcadas e móveis, onde está disponível em diversas plataformas como Linux, Windows e MacOS. Como sua estrutura é escrita em C++, antes da compilação seu pré-processador gera o código fonte da aplicação em sua linguagem base, tornando toda a estrutura gerada pela aplicação compilável em qualquer compilador padrão de C++, como o GCC.

4.1.2 Qt Creator

Qt é um ambiente de desenvolvimento integrado (IDE) multiplataforma, fornecendo suporte durante a codificação como execução, depuração e controle de versão, além de ser fácil e intuitivo de utilizar. Nele é possível escrever códigos em diversas linguagens como C++, JavaScript, Python.

4.1.3 Git

Para o versionamento de todo o código gerado durante as modificações e implementações foi utilizada a plataforma Git, um projeto de código aberto em constante manutenção desde seu lançamento utilizado para controle de versão distribuído.

4.2. Configuração do ambiente

Para a configuração do ambiente de desenvolvimento foi necessária a importação de várias bibliotecas. Abaixo segue uma descrição breve sobre as principais bibliotecas utilizadas.

4.2.1 NVidia CUDA

CUDA (Compute Unified Device Architecture) é uma plataforma desenvolvida pela NVIDIA que faz uso do poder computacional de suas unidades de processamento gráfico (GPU) ao utilizar computação paralela para acelerar o processamento de dados[6]. Suas aplicações são diversas, tais como aprendizado de máquina e inteligência artificial, permitindo simular e processar um grande volume de dados de forma eficiente, principalmente se comparado ao realizar tais ações na unidade de processamento central (CPU) de forma tradicional.

4.2.2 GCC

O GCC (GNU Compiler Collection) é um sistema de compilação para diversas linguagens de programação, sendo usado principalmente para compilar principalmente programas em C e C++, mas não se restringindo a elas [18]. Ele possui uma licença de código livre distribuído sob o código GPL (General Public License - Licença de Público Geral). O GCC é uma ferramenta que pode ser executada em diversos sistemas operacionais, sendo portado para Windows utilizando ferramentas como Gygwin, MinGW e MinGW-W64.

4.2.3 MPI

O MPI (Message Passing Interface) é uma interface de passagem de mensagens utilizado para enviar mensagens de um processo (seja num computador, estação de trabalho, etc.) para outro contendo tanto dados simples e primitivos, como inteiros e strings, quanto objetos mais complexos [19]. Por se tratar de interface, ela deve ser implementada na aplicação, podendo ser aplicada nas mais variadas linguagens como Python (pyMPI) e C# (MPI.NET).

4.2.4 GNUplot

O GNUplot consiste em um programa de plotagem de gráficos de propósito geral, sendo bastante flexível e poderoso podendo representar graficamente funções de duas e três dimensões [20]. Sendo um software de código aberto, possui versões para os principais sistemas operacionais, possuindo uma ampla quantidade de funções matemáticas que vão desde trigonometria simples até funções mais complexas como de Bessel [21] e gama [22].

4.3. Metodologia

O desenvolvimento das soluções iniciou-se já na preparação do ambiente. Ao instalar as aplicações necessárias para utilização do MC-GPU, foram encontrados problemas decorrentes da grande diferença de tempo entre as últimas releases dos software necessários e a aplicação em questão.

Durante a instalação do CUDA, devido ao um suporte não muito amadurecido, o driver necessário que acompanha a instalação do toolkit estava defeituoso, não sendo reconhecido e resultando em falha nas diversas tentativas de instalação. Como solução foi analisado os últimos drivers estáveis disponíveis e selecionado um que possibilitou uma melhor integração entre driver e utilitário: o driver versão 530.41.03 juntamente com a versão do CUDA 12.1

```
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 530.41.03              Driver Version: 530.41.03    CUDA Version: 12.1     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf           Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA GeForce GTX 1060 6GB     Off| 00000000:26:00.0  On |                  N/A |
|  0%   43C    P8              11W / 120W|    607MiB /  6144MiB |      6%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|    0   N/A  N/A      1698      G   /usr/lib/xorg/Xorg                          260MiB |
|    0   N/A  N/A      1982      G   /usr/bin/gnome-shell                         57MiB |
|    0   N/A  N/A      3978      G   ...ures=TFLiteLanguageDetectionEnabled      283MiB |
|    0   N/A  N/A     20000      G   ...er/Qt/Tools/QtCreator/bin/qtcreator        1MiB |
+---------------------------------------------------------------------------------------+
```

Figura 13: Versão driver e toolkit instalados e em pleno funcionamento [própria autoria]

Após essa seleção, o próximo passo foi a instalação do compilador GCC. A instalação padrão presente no Ubuntu 22.04 é a 13.01, o qual não era reconhecida após a instalação ser, aparentemente, bem sucedida. A solução mais satisfatória encontrada, dentre diversas pesquisas acerca do mau funcionamento e instalações de versões anteriores, foi a descida da versão para a 11.03 que se mostrou estável e totalmente funcional durante todos os testes feitos.

Durante o processo de instalação do MPI tudo ocorreu normalmente, sem apresentar nenhum contratempo.

Seguindo com a instalação dos componentes necessários, foi o momento de compilar o binário da aplicação. Ao tentar realizar a compilação, foram gerados erros e avisos em cadeia relacionados a arquivos de cabeçalho do CUDA que não foram encontrados no sistema.

```
./MC-GPU_v1.3.h:118:12: fatal error: helper_functions.h: Arquivo ou
diretório inexistente
  118 |    #include <helper_functions.h>
      |             ^~~~~~~~~~~~~~~~~~~~
compilation terminated.
```

Figura 14: Primeiro erro de importação ao tentar compilar a aplicação [própria autoria]

Como solução foi criado um script que busca o local na qual o CUDA foi instalado e realiza a cópia dos arquivos de cabeçalho necessários para a pasta da aplicação. Essa foi a melhor solução no momento, pois, dependendo de quando a instalação da biblioteca CUDA fosse feita, poderiam haver alterações nesses cabeçalhos, podendo gerar incompatibilidades futuras caso fossem disponibilizados juntamente com a aplicação após as melhorias.

O erro seguinte ao tentar compilar a aplicação residiu no caminho para os arquivos do MPI divergindo do presente no arquivo de configuração Makefile da aplicação e no comando disponibilizado para realizar a compilação. Esse problema vem do fato de que durante as constantes atualizações do MPI, seu caminho de instalação padrão foi alterado. A correção consiste na correção desse caminho no arquivo de configuração Makefile (Figura 15) e no comando utilizado para compilar o software (Figura 16).

```
36
37 # Include and library paths:
38 CUDA_PATH = /usr/local/cuda/include/
39 CUDA_LIB_PATH = /usr/local/cuda/lib64/
40 CUDA_SDK_PATH = /usr/local/cuda/samples/common/inc/
41 CUDA_SDK_LIB_PATH = /usr/local/cuda/samples/common/lib/linux/x86_64/
42 OPENMPI_PATH = /usr/include/openmpi
43
```

```
37 # Include and library paths:
38 CUDA_PATH = /usr/local/cuda/include/
39 CUDA_LIB_PATH = /usr/local/cuda/lib64/
40 CUDA_SDK_PATH = /usr/local/cuda/samples/common/inc/
41 CUDA_SDK_LIB_PATH = /usr/local/cuda/samples/common/lib/linux/x86_64/
42 OPENMPI_PATH = /usr/lib/x86_64-linux-gnu/openmpi/include
43
```

Figura 15: Comparativo do antes (acima) e depois (abaixo) do caminho do MPI (OPENMPI_PATH) no arquivo de configuração Makefile [própria autoria]

```
nvcc -DUSING_CUDA -DUSING_MPI MC-GPU_v1.3.cu -o MC-GPU_v1.3.x -O3
  -use_fast_math -L/usr/lib/ -I. -I/usr/local/cuda/include
  -I/usr/local/cuda/samples/common/inc -I/usr/local/cuda/samples/shared/inc/
  -I/usr/include/openmpi  -lmpi -lz --ptxas-options=-v
  -gencode=arch=compute_20,code=sm_20 -gencode=arch=compute_30,code=sm_30
```

```
nvcc -DUSING_CUDA -DUSING_MPI MC-GPU_v1.3.cu -o MC-GPU_v1.3.x -O3
  -use_fast_math -L/usr/lib/ -I. -I/usr/local/cuda/include
  -I/usr/local/cuda/samples/common/inc -I/usr/local/cuda/samples/shared/inc/
  -I/usr/lib/x86_64-linux-gnu/openmpi/include  -lmpi -lz --ptxas-options=-v
  -gencode=arch=compute_20,code=sm_20 -gencode=arch=compute_30,code=sm_30
```

Figura 16: Comparativo do antes (acima) e depois (abaixo) do caminho do MPI no comando de compilação do programa [própria autoria]

Após as modificações anteriores, o programa foi compilado, porém com ressalvas. Diversos warnings (avisos) são disparados devido às funções e instruções defasadas ou descontinuadas.

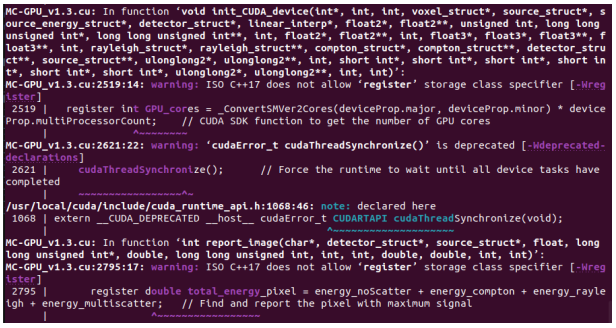

Figura 17: Alguns dos warnings disparados ao compilar a aplicação [própria autoria]

A resolução desses warnings consistiu basicamente em:

- Alteração de algumas chamadas do CUDA que foram descontinuadas e substituídas;
- Ajuste no tamanho de determinadas variáveis para evitar erros durante as simulações;
- Adição e remoção de termos e modificadores necessários nas versões atuais dos softwares de base para a aplicação.

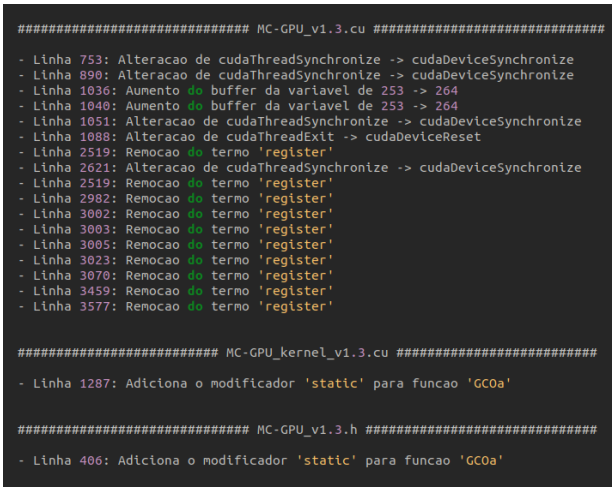

Figura 18: Locais e alterações realizadas para tratar os warnings [própria autoria]

Tendo em vista que algumas informações foram encontradas através de diversas pesquisas e tentativas, foi criado um manual para facilitar a instalação juntamente com alguns scripts para facilitar algumas partes do processo.

Seguindo após todas as modificações, o programa foi compilado sem problemas, prosseguindo para os testes com a ferramenta.

Para realizar as simulações e verificar as diferenças de desempenho ao simular no modo CPU e no modo GPU, foi utilizado o exemplo 6voxels disponibilizado juntamente com o MC-GPU. Para realizar as simulações na CPU foi utilizado um AMD Ryzen™ 9 5900X e para as simulações na GPU foi utilizada a GEFORCE GTX 1060 6GB.

Na simulação de 6voxels foram utilizadas histórias que variam de 100 milhões até 100 bilhões. Ao analisar seus resultados, a utilização da GPU gerou um ganho de 18 a 20 vezes em relação a CPU, ganho esse ainda modesto visto o desequilíbrio do sistema por utilizar uma GPU relativamente antiga para os padrões atuais.

Quantidade de histórias	Tempo CPU em segundos	Tempo GPU em segundos	Percentual de Ganho
100 bilhões	61.586	3.134	1865%
50 bilhões	31.319	1.567	1899%
30 bilhões	18.603	940	1879%
10 bilhões	6.072	313	1840%
5 bilhões	3.100	156	1887%
3 bilhões	1.875	94	1895%
1 bilhão	632	31	1939%
500 milhões	302	15	1913%
300 milhões	184	9	1944%
100 milhões	60	3	1900%

Figura 19: Comparação de tempo de simulações na CPU e GPU no exemplo 6voxels [própria autoria]

Tendo testado e comprovado a eficiência do programa, foi iniciada a criação da tela utilizando o framework Qt em conjunto com a IDE Qt Creator. A interface gráfica foi criada para possibilitar a alteração dos parâmetros principais para realização de uma simulação.

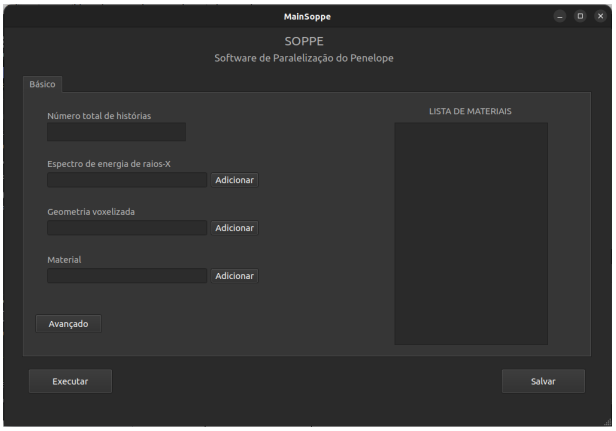

Figura 20: Tela inicial do software (tela básica) [própria autoria]

Também foi disponibilizada uma opção avançada que apresenta os demais dados mais sensíveis que podem ser manipulados para realização de uma simulação. Por serem parâmetros mais sensíveis, uma janela de confirmação foi adicionada antes de exibir as opções. Essa opção foi criada com o objetivo de evitar possíveis mudanças de posição nas entradas que podem acarretar no não funcionamento de uma simulação, algo que ocorre no PENELOPE.

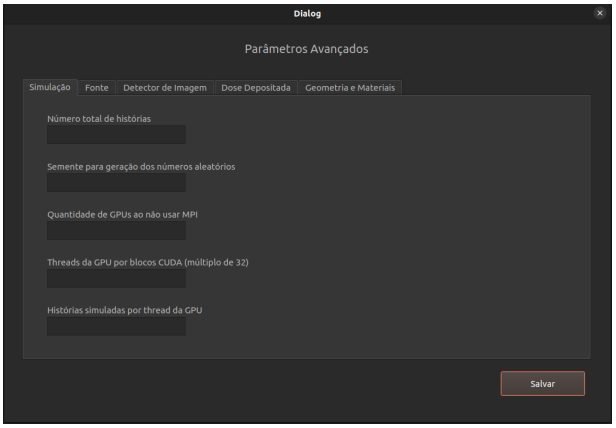

Figura 21: Tela avançada de manipulação de parâmetros [própria autoria]

Seguidamente foram feitas as funções da tela, iniciando pela inclusão dos parâmetros do modo básico e avançado. O modo básico utiliza um arquivo de entrada com algumas configurações padronizadas para ser possível realizar uma simulação. No modo avançado todas as entradas editáveis foram disponibilizadas para manipulação.

Posteriormente foi adicionada a funcionalidade de inserir os arquivos de materiais, sendo possível visualizar todos os que foram adicionados para a simulação em uma listagem presente no lado direito da tela correspondente. Adicionalmente, também foi implementada a possibilidade de utilizar arquivos de materiais extraídos do PENELOPE de forma direta, com a aplicação fazendo as devidas conversões de forma automatizada à medida que esses materiais são adicionados à listagem.

Figura 22: Listagem dos materiais [própria autoria]

No passo seguinte foi adicionada uma tela de saída para apresentar as informações geradas após a finalização da simulação juntamente com os arquivos para geração das imagens em seu formato .raw e .dat.

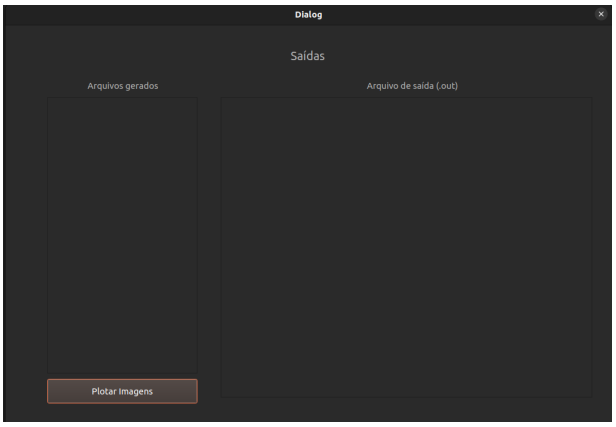

Figura 23: Tela de saída [própria autoria]

Também foi implementada a opção de plotagem das imagens a partir dos arquivos .dat gerados ao final de uma simulação.

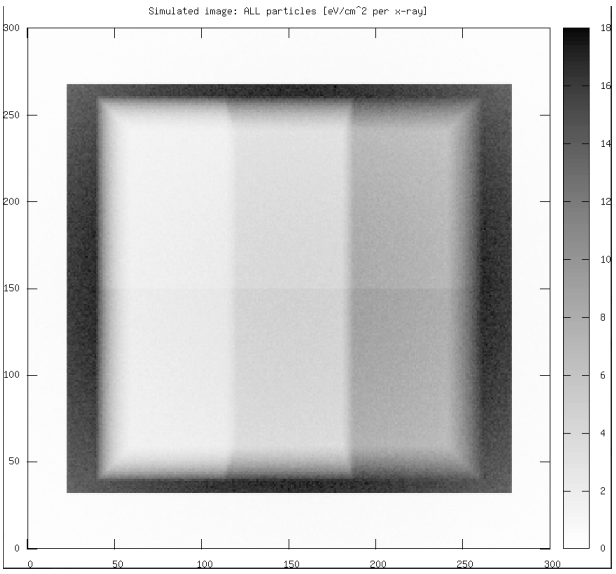

Figura 24: Plotagem da imagem da simulação 6voxels [própria autoria]

5. CONCLUSÃO

Como observado anteriormente, a aplicação proporcionou ganhos de 18 a 20 vezes menores em relação ao tempo necessário para realizar simulações de forma convencional com a CPU. Entretanto, a preparação do ambiente para sua utilização possui muitas inconsistências, além de possuir uma interface pouco amigável.

Considerando que a preparação do ambiente para utilização das simulações possui muitas inconsistências e complexidade, a abordagem proposta corrigiu tais problemas de instalação, resolvendo os erros de forma intuitiva. Além disso, foi disponibilizado um manual para auxiliar na sua instalação juntamente com alguns scripts para facilitar alguns passos.

Adicionalmente, foi criada uma interface gráfica para tornar mais intuitiva a manipulação dos parâmetros necessários para uma simulação, tanto com opções mais básicas quanto avançadas, além de poder visualizar as saídas e realizar a plotagem das imagens geradas ao final de uma simulação.

Para os arquivos de materiais que são utilizados nas simulações, foi implementado a possibilidade de utilizar os arquivos de materiais provenientes diretamente do PENELOPE, arquivos estes que são ajustados para os padrões da aplicação automaticamente quando adicionados à lista.

6. TRABALHOS FUTUROS

O trabalho objetivou as correções na preparação do ambiente e a criação da interface gráfica de forma a melhorar a experiência do usuário ao utilizar o programa.

Em trabalhos futuros, podem ser implementadas funções mais aprimoradas e avançadas, como a maior automação nas configurações do ambiente, importação de arquivos de parâmetros de entrada e a aceitação de arquivos de entradas provenientes do PENELOPE.

7. NOMENCLATURA

- TC: Tomografia Computadorizada
- RM: Ressonância Magnética
- CUDA: Compute Unified Device Architecture
- ALU: Arithmetic Logic Unit
- CPU: Central Processing Unit
- GPU: Graphics Processing Unit
- CERN: European Organization for Nuclear Research

8. REFERÊNCIAS

[1] Flávio Pereira das Posses. **Exames de imagem: 8 tipos mais comuns e para que servem**. Disponível em: <https://star.med.br/exames-de-imagem/>

[2] Flávio Pereira das Posses. **Radiografia: o que é raio X e como funciona o exame?**. Disponível em: <https://star.med.br/raio-x-o-que-e/#:~:text=O%20exame%20de%20radiografia%2C%20popularmente,seja%2C%20os%20raios%20X).>

[3] Instituto Nacional de Câncer - INCA. **Radiações ionizantes**. Disponível em: <https://www.gov.br/inca/pt-br/assuntos/causas-e-prevencao-do-cancer/exposicao-no-trabalho-e-no-ambiente/radiacoes/radiacoes-ionizantes#:~:text=As%20fontes%20naturais%20da%20radia%C3%A7%C3%A3o,e%20no%20pr%C3%B3prio%20corpo%20humano.>

[4] Jabbari K. **Review of Fast Monte Carlo Codes for Dose Calculation in Radiation Therapy Treatment Planning.** J Med Signals Sens. 2011 Jan;1(1):73-86. Disponível em: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3317764/>

[5] Edimário Silva Moura Júnior et al. **MÁQUINAS PARALELAS: MULTICOMPUTADORES**. Disponível em: <https://www.docentes.univasf.edu.br/max.santana/material/aoc-ii/SistemasMulticomputadores.pdf>

[6] NVIDIA Developer. **CUDA Zone**. Disponível em: <https://developer.nvidia.com/cuda-zone>

[7] Emico Okuno. **Efeitos biológicos das radiações ionizantes. Acidente radiológico de Goiânia.** São Paulo, 2013. Disponível em: <https://www.scielo.br/pdf/ea/v27n77/v27n77a14.pdf>

[8] Instituto Nacional de Câncer - INCA. **Ambiente, Trabalho e Câncer: Aspectos Epidemiológicos, Toxicológicos e Regulatórios.** Rio de Janeiro, 2021. Disponível em: <https://www.scielo.br/pdf/ea/v27n77/v27n77a14.pdf>

[9] NEA. **PENELOPE-2018: A Code System for Monte Carlo Simulation of Electron and Photon Transport.** Espanha, 28 de Janeiro - 1 de Fevereiro de 2019. Disponível em: <https://www.oecd-nea.org/jcms/pl_46441/penelope-2018-a-code-system-for-monte-carlo-simulation-of-electron-and-photon-transport >

[10] Murillo Bellezzo. **DESENVOLVIMENTO DE UM SOFTWARE DE MONTE CARLO PARA TRANSPORTE DE FÓTONS EM ESTRUTURAS DE VOXELS USANDO UNIDADES DE PROCESSAMENTO GRÁFICO.** São Paulo, 2014. Disponível em: <http://pelicano.ipen.br/PosG30/TextoCompleto/Murillo%20Bellezzo_M.pdf >

[11] GEANT4 Collaboration. **GEANT4--a simulation toolkikt.** Nucl. Instrum. Methods Phys. Res., A 506 (2003) 250-303. Disponível em: <http://cds.cern.ch/record/602040/?ln=pt >

[12] I. Kawrakow. **On the condensed history technique for electron transport.** National Research Council of Canada, Published by Elsevier Science B.V. (1998) 253-280. Disponível em: <http://websites.umich.edu/~nersb590/CourseLibrary/E7/6.pdf >

[13] V. Giménez-Alventosa. **PenRed: An extensible and parallel Monte-Carlo framework for radiation transport based on PENELOPE.** Universitat de València-CSIC, 2020.

[14] Andreu Badal and Aldo Badano. **Accelerating Monte Carlo simulations of photon transport in a voxelized geometry using a massively parallel Graphics Processing Unit.** Medical Physics 36, pp. 4878–4880 (2009). Disponível em: <https://didsr.github.io/MCGPU/wiki/html/index.html >

[15] Fortran Community. **Fortran High-performance parallel programming language.** Disponível em: <https://fortran-lang.org/en/>

[16] Fortran Community. **The Zubal Phantom Voxel-Based Anthropomorphic Phantoms.** Yale IPAG - Image Processing and Analysis Group. Disponível em: <https://noodle.med.yale.edu/zubal/index.htm >

[17] Qt Wiki. **About Qt.** Disponível em: <https://wiki.qt.io/About_Qt >

[18] JavaTpoint. **GCC Linux.** Disponível em: <https://www.javatpoint.com/gcc-linux >

[19] Loyola University Chicago. **Message Passing Interface.** Disponível em: <https://ds.cs.luc.edu/mpi/mpi.html >

[20] Bryan Wright. **Practical Computing for Science and Engineering.** Physics Department - University of Virginia. 2020 August; 549-573. Disponível em: <http://galileo.phys.virginia.edu/compfac/courses/practical-c/ >

[21] Matemática Simplificada. **A EQUAÇÃO E AS FUNÇÕES DE BESSEL | SOLUÇÃO DE E.D.O.'S POR SÉRIES.** Disponível em: <https://matematicasimplificada.com/equacao-funcoes-bessel/ >

[22] Matemática Simplificada. **FUNÇÃO GAMA | FATORIAL GENERALIZADO E GAMA INCOMPLETA.** Disponível em: <https://matematicasimplificada.com/funcao-gama/ >

[23] Qt Group. **Development Tools.** Disponível em: <https://www.qt.io/product/development-tools >

[24] Couto, R.R., Santiago, A.J. **Radioatividade e Irradiação de Alimentos.** Universidade do Estado do Rio de Janeiro – UERJ, Rio de Janeiro, RJ, 2010.

[25] Júnior, Joab Silas da Silva. **O que são os raios X?**. Brasil Escola. Disponível em: <https://brasilescola.uol.com.br/o-que-e/fisica/o-que-sao-os-raios-x.htm >

[26] Argonne Leadership Computing Facility. **Re-imagining Applications of X-ray Tomography with Deep Learning.** Disponível em: <https://www.alcf.anl.gov/events/re-imagining-applications-x-ray-tomography-deep-learning >

[27] Weisstein, Eric W. **Buffon's Needle Problem.** MathWorld--A Wolfram Web Resource. Disponível em: <https://mathworld.wolfram.com/BuffonsNeedleProblem.html >

[28] Maurício Moralles. **Método de Monte Carlo em simulações de transporte de partículas na matéria.** Instituto de Pesquisas Energéticas e Nucleares (IPEN/CNEN). Disponível em: <https://edisciplinas.usp.br/pluginfile.php/4400111/mod_resource/content/1/apresMonteCarlo.pdf >