UNIVERSIDADE DE SÃO PAULO
Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto Departamento de Física e Matemática

José Eduardo Batista

Avaliação de cálculos de distribuição de dose de radiação utilizando unidades de processamento gráfico e algoritmo Monte Carlo: rumo a um suporte ao planejamento em radioterapia.

Ribeirão Preto
2023

JOSÉ EDUARDO BATISTA

Avaliação de cálculos de distribuição de dose de radiação utilizando unidades de processamento gráfico e algoritmo Monte Carlo: rumo a um suporte ao planejamento em radioterapia.

Dissertação apresentada à Faculdade de Filosofia Ciências e Letras de Ribeirão Preto da Universidade de São Paulo como parte dos requisitos para obtenção do título de Mestre em Computação Aplicada. VERSÃO CORRIGIDA

Área de concentração: Computação Aplicada

Orientador: Prof. Dr. Luiz Otavio Murta Junior.

Ribeirão Preto
2023

Batista, José Eduardo

Avaliação de cálculos de distribuição de dose de radiação utilizando unidades de processamento gráfico e algoritmo Monte Carlo: rumo a um suporte ao planejamento em radioterapia. Versão Corrigida. Ribeirão Preto, 2023.

92 p. : il. ; 30 cm

Dissertação de Mestrado, apresentada à Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto da Universidade de São Paulo. Área de concentração: Computação Aplicada.

Orientador: Prof. Dr. Luiz Otavio Murta Junior

1. Método Monte Carlo. 2. GPU. 3. PENELOPE. 4. Tratamento radioterápico. 5. Transporte de radiação. 6 Depósito de dose. 7. Programação Paralela

Dedico à minha companheira, Mariane Kiyoto Moyses, com amor e gratidão por sua compreensão e incentivo constante durante todo o longo período de elaboração deste trabalho.

AGRADECIMENTOS

Ao Prof. Drº. Luiz Otavio Murta Junior. Agradeço pelas suas orientações e ensinamentos que foram cruciais para elaboração deste trabalho. Obrigado por sua paciência e disponibilidade para constantes reuniões, mesmo em uma situação atípica vivida por nós nesses últimos dois anos, sempre disposto a me mostrar o melhor caminho a ser seguido.

À Prof.ª. Drª Patrícia Nicolucci. Obrigado pelos seus ensinamentos que foram importantes para compreensão e desenvolvimento deste trabalho.

Ao Instituto Federal de São Paulo, instituição de ensino na qual trabalho e que me deu apoio para realização do curso de mestrado.

Aos meus colegas do Laboratório de Computação em Sinais e Imagens Médicas da Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto: Vinícius, Vivian, Jackson, Leonardo e Marcos. Obrigado pelas opiniões e esclarecimentos durantes nossas reuniões em grupo.

Aos professores do Programa de Pós-graduação em Computação Aplicada dessa universidade, que realizam um trabalho excepcional pela qualidade do curso.

E a todos os meus amigos e familiares que, mesmo não citados nominalmente, tiveram uma participação na elaboração deste trabalho.

RESUMO

BATISTA, J. E. Avaliação de cálculos de distribuição de dose de radiação utilizando unidades de processamento gráfico e algoritmo Monte Carlo: rumo a um suporte ao planejamento em radioterapia. 2023. Dissertação de Mestrado – Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto, Universidade de São Paulo, Ribeirão Preto, 2023.

A simulação computacional utilizando o Método Monte Carlo (MMC) aplicado ao transporte de radiação é considerado padrão ouro em radioterapia, apresentando-se como o método mais exato para a cálculo da dose depositada no tecido alvo e nos órgãos adjacentes. No entanto, o seu longo tempo de processamento computacional necessário para atingir valores satisfatórios de incerteza estatística, torna sua aplicação inviável na clínica médica de rotina. A programação paralela, com implementação de algoritmos para execução em unidades de processamento gráfico (GPUs), tem se mostrado uma das alternativas exploradas por pesquisadores para reduzir o tempo computacional e tornar o uso do MMC clinicamente viável. Este trabalho teve por objetivo a adaptação para execução paralela, em unidades de processamento gráfico, do algoritmo Monte Carlo, aplicado ao transporte de radiação, presente no PENELOPE-2014, visando manter a exatidão e diminuir o tempo computacional necessário para obtenção de resultados como depósito de dose nos corpos e mapa de distribuição de dose. O desenvolvimento foi realizado na plataforma CUDA e tomou como base o pacote PENELOPE-2014, que possui estrutura de execução sequencial em unidades centrais de processamento (CPUs). A comparação dos resultados obtidos das simulações executadas no novo algoritmo paralelo, em GPU, com as simulações executadas no PELENOPE-2014, em CPU, demonstraram um grau de exatidão com diferenças menores do que 1% para depósito de dose e diferenças imperceptíveis para os mapas de distribuições de doses a um fator de desempenho de cerca de 13 vezes mais rápido. A combinação dos ganhos de desempenho com a manutenção da exatidão da simulação, demonstra que a execução paralela na GPU apresenta um potencial de tornar o uso do MMC uma técnica viável para sistemas de planejamento radioterápico.

Palavras chaves: Método Monte Carlo, GPU, PENELOPE, Tratamento radioterápico, Transporte de radiação, Deposito de dose, Programação paralela.

Graduate Program in Applied Computing
Master's dissertation

Evaluation of Radiation Dose Distribution Estimations using Graphic Processing Units and Monte Carlo Algorithm: toward a Support in Radiotherapy Planning.

Student: José Eduardo Batista
Supervisor: Prof. Luiz Otavio Murta Junior

Computer simulation using the Monte Carlo Method (MCM) applied to radiation transport is considered the gold standard in radiotherapy, presenting itself as the most accurate method for calculating the dose deposited in the target tissue and adjacent organs. However, its long computational processing time, necessary to reach satisfactory statistical uncertainty values, makes its application unfeasible in routine medical practice. Parallel programming, with the implementation of algorithms for execution in graphic processing units (GPUs), is one of the alternatives explored by researchers to reduce computational time and make the use of MMC clinically viable. This work's objective was to adapt the Monte Carlo algorithm for parallel execution, in graphic processing units, applied to radiation transport, present in PENELOPE-2014, aiming to maintain accuracy and reduce the computational time necessary to obtain results such as dose deposit in bodies and dose distribution map. The development was carried out on the CUDA platform and was based on the PENELOPE-2014 package, which has a sequential execution structure in central processing units (CPUs). The comparison of the results obtained from the simulations executed in the new parallel algorithm, in GPU, with the simulations executed in PELENOPE-2014, in CPU, demonstrated a degree of accuracy with differences lower than 1% for dose deposit and imperceptible differences for the maps from dose distributions to a performance factor of about 13 times faster. The combination of performance gains keeping the simulation accuracy demonstrates that parallel execution on the GPU can make MCM a viable technique for radiotherapy planning systems.

Keywords: *Monte Carlo Method, GPU, PENELOPE, radiotherapy treatment, radiation transport, dose delive, parallel programming.*

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE GRÁFICOS

ÍNDICE

1 INTRODUÇÃO

Uma das principais etapas do planejamento radioterápico consiste no cálculo da dose de radiação depositada no tecido alvo e nos órgãos adjacentes. Existem diversos métodos para se estimar o cálculo de deposição de dose no corpo, dentre eles, a simulação computacional, utilizando o Método Monte Carlo, é considerado o padrão ouro em radioterapia se apresentando como o método mais exato (JIA *et al.*, 2011).

Para que a simulação Monte Carlo aplicada ao transporte de radiação alcance uma alta precisão, reduzindo a incerteza estatística da grandeza desejada, faz-se necessário um longo tempo de processamento computacional tornando-a imprópria sua utilização na clínica médica de rotina. Abordagens como a redução da complexidade física das funções do algoritmo que trata do transporte de partículas, a redução da variância por adoção de métodos como roleta russa e divisão de pares e a utilização de computação paralela em unidades de processamento gráfico (GPUs), têm sido adotadas na busca por acelerar o Método Monte Carlo e torná-lo clinicamente viável (WANG *et al.*, 2016).

O lançamento da plataforma de programação paralela CUDA, desenvolvida pela NVIDIA e lançada em 2006, e o avanço tecnológico em hardware, proporcionando um maior poder computacional das novas GPUs, inspirou pesquisadores durante a última década que passaram a explorar simulações computacionais pesadas como o Método Monte Carlo aplicado ao transporte de radiação (MIRZAPOUR *et al.*, 2020).

Como alternativa ao CUDA, uma solução proprietária da Nvidia, existe também a plataforma de programação aberta OpenCL. Esta última fornece uma estrutura para os desenvolvedores escreverem programas que são executados em arquiteturas de CPU e GPU de diferentes fabricantes, o que facilita a portabilidade de programas em hardwares diferentes. No entanto, essa portabilidade tem um custo que pode reduzir a eficiência computacional dos programas desenvolvidos (TIAN *et al.*, 2015). Em especial, situações em que o tempo de processamento é um aspecto de grande relevância como é na simulação Monte Carlo aplicada ao transporte de radiação, essa redução de eficiência pode comprometer o desempenho final da aplicação.

Dessa forma, este trabalho fará uso da plataforma de programação paralela CUDA, com linguagem de programação C++, para a adaptação do um algoritmo Monte Carlo, aplicado ao transporte de radiação, presente no PENELOPE-2014, com o objetivo de manter a exatidão e diminuir o tempo computacional necessário para obtenção de resultados como deposição de dose nos corpos e mapa de distribuição de dose. O desenvolvimento tomará como base o próprio pacote PENELOPE-2014, originalmente escrito em linguagem de programação em Fortran 77,

com estrutura de execução sequencial em unidades centrais de processamento (CPUs).

O estudo está organizado em seis partes. Primeiro é apresentada uma introdução sobre o problema de pesquisa. Na segunda parte é apresentada uma proposta com os objetivos do trabalho. Na terceira etapa foi realizada uma pesquisa sobre os fundamentos teóricos que embasam o desenvolvimento do projeto. Na quarta parte são apresentados os materiais e métodos utilizados para o desenvolvimento do projeto e alcance do objetivo proposto. Na quinta etapa são realizadas discussões sobre os resultados obtidos. Na sexta e última parte é aprestada uma conclusão sobre o trabalho desenvolvido.

2 OBJETIVO

O objetivo deste trabalho é a realização de uma adaptação e avaliação para execução paralela, em unidades de processamento gráfico, do algoritmo sequencial de Monte Carlo, aplicado ao transporte de radiação, contido no PENELOPE-2014, com o intuito de diminuir o tempo computacional necessário para obtenção de resultados de deposição de dose nos corpos e o mapa de distribuição de dose.

Especificamente, este estudo se propõe a:

a) Compreender e elencar as sub-rotinas e funções da versão 2014 do PENELOPE, originalmente escrito em Fortran, que possibilitem a execução de uma simulação Monte Carlo, possuindo fótons como partícula primária e estruturas de corpos em geometria quadrática;
b) Desenvolver em linguagem C++ o pacote de sub-rotinas e funções elencadas visando a validação da simulação em uma linguagem de programação próxima da plataforma CUDA;
c) Adaptar o novo algoritmo desenvolvido em C++ para a linguagem de programação paralela CUDA C++ de modo a buscar um melhor desempenho da aplicação;
d) Simular três cenários diferentes, a saber, câncer cerebral, câncer de pulmão, e câncer de próstata nos três algoritmos disponíveis, PENELOPE-2014, versão em C++ e CUDA C++;
e) Avaliar os resultados das simulações dos diferentes cenários em temos de precisão e performance.

3 FUNDAMENTAÇÃO TEÓRICA

Para uma adequada compreensão da metodologia utilizada e dos resultados obtidos, foi realizada uma revisão de literatura dos fundamentos teóricos relevantes para este trabalho. A seção 3.1 apresenta conceitos importantes no contexto da radioterapia, como suas grandezas dosimétricas e as formas de interações das partículas com a matéria. Na seção 3.2 apresenta-se uma breve síntese histórica sobre o Método Monte Carlo e sua aplicabilidade ao transporte de radiação. A seção 3.3 descreve as características do pacote PENELOPE e os resultados obtidos de sua simulação que são pertinentes ao trabalho. A seção 3.4 traz os conceitos de programação paralela em GPUs. A seção 3.5 apresenta trabalhos correlatos envolvendo a aplicação da programação paralela em GPUs e a utilização de simulação Monte Carlo no transporte de partículas.

3.1 Radioterapia

O câncer é uma doença que acomete pessoas ao redor de todo o mundo. Em 2018 foram registrados 18,1 milhões de casos e este número continuará crescendo com estimativa de alcançar 29,4 milhões de pessoas em 2040 (WHO, 2020).

Os tratamentos de câncer existentes consistem na realização de cirurgia, radioterapia e quimioterapia, podendo ser realizado uma combinação dessas técnicas de acordo com o planejamento médico elaborado. A radioterapia se baseia na utilização de radiação ionizante cujo objetivo é atingir o tecido tumoral tendo como fator limitante os riscos de danos aos tecidos sadios adjacentes (COSTA, 2013).

Existem duas possibilidades de realizar a aplicação da radioterapia: braquiterapia e teleterapia. Na técnica de braquiterapia, o tratamento radioterápico é realizado por meio de implantes de materiais radioativos dentro ou muito próximo ao tumor, de modo a se obter uma maior concentração de energia no tecido alvo e reduzir a dose nos órgãos e estruturas adjacentes (BVSMS, 2013). Já a teleterapia, também chamada de radioterapia de feixe externo, envolve a utilização de equipamentos externos com fontes radioativas (geralmente ^{60}Co) ou aceleradores lineares de partículas (*Linacs – Linear Particles Accelerator*) capazes de gerar feixes de elétrons e fótons com diferentes energias e direcioná-los a áreas do corpo humano (MEDEIROS, 2018).

Diferentes tipos de radiação podem ser utilizados na radioterapia tais como elétrons, fótons, prótons e nêutrons. O tratamento do câncer por radioterapia com emissão de prótons tem se tornado uma opção promissora visto aos benefícios proporcionados como a melhora na distribuição de dose no volume alvo e na melhor preservação dos tecidos sadios adjacentes em

comparação com a radioterapia efetuada com uso de feixes de fótons. No entanto, o alto investimento para instalação e realização dessa técnica faz da terapia com feixe de fótons ainda a mais utilizada atualmente (PEETERS *et al.*, 2010).

Para a realização do processo de radioterapia é muito importante elaborar um plano de tratamento para o paciente. Normalmente esse plano é desenvolvido por meio de sistemas de planejamento de tratamento (TPS) que fornecem informações de dose em 3D não apenas no volume alvo, mas também nos tecidos e órgãos saudáveis ao redor do tumor. Essas informações possibilitam otimizar a dose prescrita sem causar morbidade ao paciente, maximizando a probabilidade de controle do tumor (TCP) e minimizando a probabilidade de complicação tecidual normal (NTCP) (PODGORSAK, 2016).

Entre os algoritmos existentes para o TPS estão o feixe estreito (*pencil beam*) e o Monte Carlo. Esses algoritmos apresentam vantagens e desvantagens com relação à precisão e ao tempo de computação necessário para sua execução. A Comissão Internacional de Unidades e Medidas de Radiação (ICRU), estabelece como precisão mínima que valor da dose calculada por esses algoritmos deva estar entre 2% e 3% do esperado. O feixe de lápis é um algoritmo muito rápido, porém não modela com precisão a distribuição de elétrons secundários em meios heterogêneos. O Monte Carlo, por sua vez, é um algoritmo que necessita de um longo tempo de processamento, pois realiza um rastreio explícito das partículas primárias e secundárias conferindo ao método uma maior exatidão sendo, portanto, considerado padrão ouro em radioterapia (ELCIM; DIRICAN; YAVAS, 2018).

3.1.1 Grandezas Dosimétricas

A radiação ionizante tem a capacidade de interagir com a matéria em uma série de processos em que a energia da partícula ionizante é convertida podendo ser depositada no corpo no qual ela se encontra. As quantidades dosimétricas relevantes para o trabalho são apresentadas abaixo:

3.1.1.1 Energia Depositada

De acordo com o relatório da Comissão Internacional de Unidades e Medidas das Radiações (ICRU número 85), que trata sobre as grandezas e unidades fundamentais para radiação ionizante, a energia depositada *E* em uma única interação *i* é dada por:

$$Ei = Ein - Eout + Q$$

onde, *Ein* é a energia da partícula incidente, *Eout* é a soma das energias das partículas que deixam a interação e *Q* é a mudança das energias de repouso do núcleo de todas as partículas elementares envolvidas na interação. A unidade para energia depositada é dada em joule (J) e a energia transmitida à matéria num dado volume é a soma de todos os depósitos de energia no volume.

3.1.1.2 Dose Absorvida

O relatório do ICRU, número 85, (ICRU, 2011) também traz a definição da dose absorvida, *D*, como sendo a energia média cedida pela radiação ionizante, *dE*, à matéria de massa *dm*. Assim, temos:

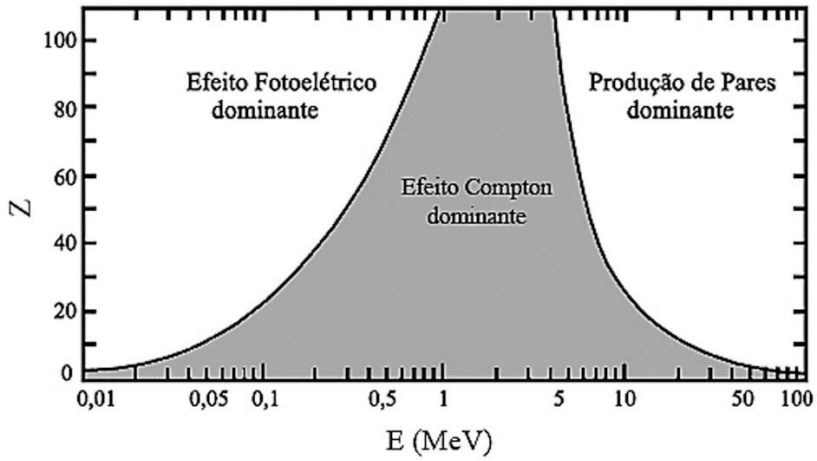
$$D = \frac{dE}{dm}$$

A unidade de dose absorvida é o joule por quilograma ($\frac{\mathrm{J}}{\mathrm{kg}}$), tendo como nome especial de tratamento o gray (*Gy*).

3.1.2 Física das interações dos fótons com a matéria

No contexto da radioterapia, os tipos de interações que podem ocorrer com fótons na faixa de energia variando de centenas de keV até alguns MeV são o efeito fotoelétrico, o efeito Compton e a produção de pares. Essas interações transferem energia para novas partículas carregadas (elétrons ou pósitrons) sendo os fótons considerados como partículas indiretamente ionizantes. Importante destacar que a probabilidade de ocorrência de cada tipo de interação depende da energia do fóton incidente e do número atômico do material com o qual o fóton interage. Esta probabilidade é relacionada à grandeza chamada seção de choque do material (SÁ *et al.*, 2016). A figura 1 mostra a predominância dos efeitos de interação dos fótons ao percorrer a matéria.

Figura 1 - Tipo de interação predominante, em função da energia (E) e do número atômico do material (Z) com o qual o fóton interage.

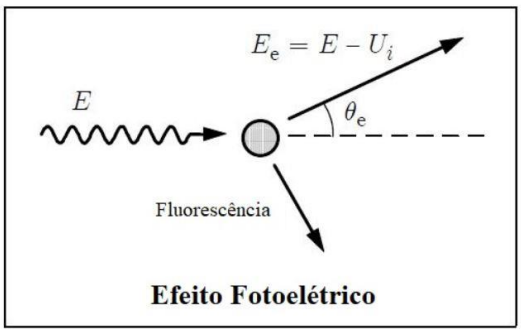


Fonte: (SÁ *et al.*, 2016)

3.1.2.1 Efeito Fotoelétrico

O efeito fotoelétrico ocorre com maior probabilidade em fótons de baixa energia percorrendo um material de alto número atômico. Nesta interação, o fóton é totalmente absorvido pelo átomo e um elétron é liberado para se mover no material (YOSHIMURA, 2009). A figura 2 demostrada o efeito fotoelétrico onde um elétron é ejetado do átomo em um ângulo *θe*, com energia cinética *Ee* que é igual à diferença entre a energia do fóton incidente *E* e à energia de ligação do elétron *Ui* no átomo.

Figura 2 - Efeito Fotoelétrico



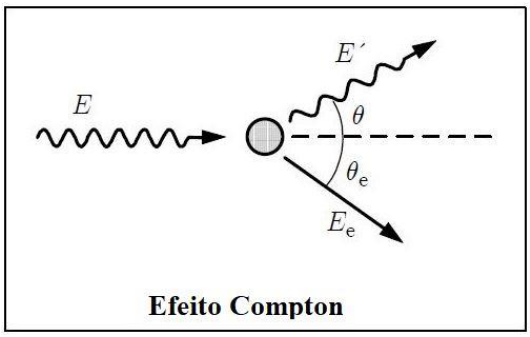
Fonte: (SALVAT, 2015)

Além da emissão do elétron, o efeito fotoelétrico cria um átomo ionizado, com vacância em uma de suas camadas eletrônicas que ao ser ocupada por um elétron de camada superior emite um fóton chamado de fluorescência.

3.1.2.2 Espalhamento Incoerente ou Efeito Compton

O efeito Compton tem maior probabilidade de ocorrer em qualquer material quando os fótons possuem energias de valor intermediário, entre algumas centenas de keV e alguns MeV, e em materiais de baixo número atômico independente de sua energia (YOSHIMURA, 2009). Nesta interação, o fóton possui uma energia maior do que a energia de ligação do elétron e isso torna altamente improvável que o fóton consiga transferir toda a sua energia a um único elétron e ser absorvido pelo átomo. Neste caso, o fóton só consegue transferir parte de sua energia para o elétron ocasionando sua ejeção da camada eletrônica e o espalhamento do fóton em uma nova direção (MEDEIROS, 2018). A figura 3 demostra o efeito Compton onde um fóton *E'* é espalhado em um ângulo θ e um elétron *Ee* é emitido em um ângulo θe decorrentes da interação de um fóton *E* com o átomo.

Figura 3 - Efeito Compton

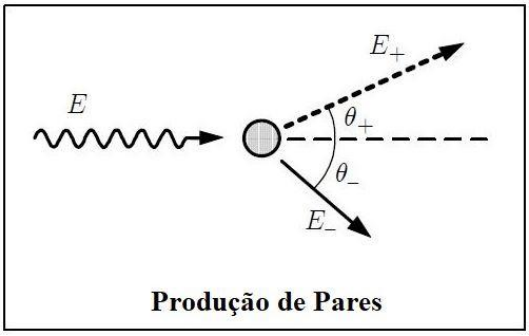


Fonte: (SALVAT, 2015)

3.1.2.3 Produção de Pares

A produção de pares ocorre com maior probabilidade em fótons com alta energia, acima de 1,022 MeV, e em materiais de alto número atômico. Neste tipo de interação o fóton é absorvido e toda sua energia é convertida em massa de repouso e energia cinética de um par elétron/pósitron (YOSHIMURA, 2009). A figura 4 demonstra a produção de pares onde um pósitron E_+ é emitido em um ângulo θ_+ e um elétron E_- é emitido em um ângulo θ_- após a interação do fóton *E* com o átomo.

Figura 4 - Produção de Pares



Fonte: (SALVAT, 2015)

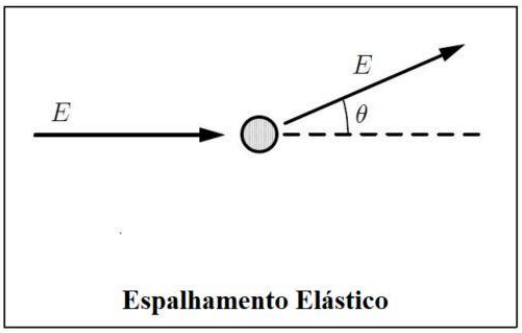
3.1.3 Física das interações dos elétrons e pósitrons com a matéria

Os elétrons energéticos sofrem interações coulombianas com elétrons orbitais e núcleos atômicos à medida que percorrem a matéria. Essas colisões assumem um caráter elástico ou inelástico e podem provocar perda de energia cinética ou apenas mudar sua direção de movimento (PODGORSAK, 2005).

3.1.3.1 Espalhamento Elástico

A grande maioria das interações sofridas pelas colisões dos elétrons com a matéria são caracterizadas como espalhamento elástico. Nessa interação a partícula é espalhada pelo núcleo e perde apenas uma insignificante quantidade de energia cinética requerida para conservar o momento (PODGORSAK, 2016). Esse tipo de interação praticamente não contribui com a deposição de energia no meio. A figura 5 demostrada o espalhamento elástico onde apenas a direção do movimento do elétron E é alterado pelo ângulo θ de espalhamento.

Figura 5 - Espalhamento Elástico

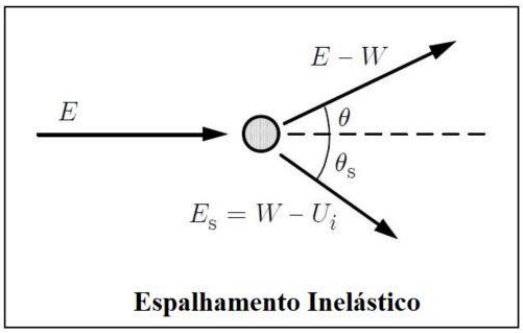


Fonte: (SALVAT, 2015)

3.1.3.2 Espalhamento Inelástico

O espalhamento inelástico representa uma pequena porcentagem das interações do elétron com a matéria, neste caso, com o elétron orbital atômico, mas pode resultar em uma perda significativa de energia e contribuição para o depósito de energia no meio (PODGORSAK, 2016). A figura 6 mostra o Espalhamento Inelástico onde o elétron incidente *E* pode ser espalhado em um ângulo θ com energia menor. O elétron orbital atômico *Es*, que sofreu a interação, é liberado em um ângulo θs com energia *W* igual a transmitida pelo elétron *E* incidente menos a energia de ligação *Ui*, referente a camada atômica.

Figura 6 - Espalhamento Inelástico

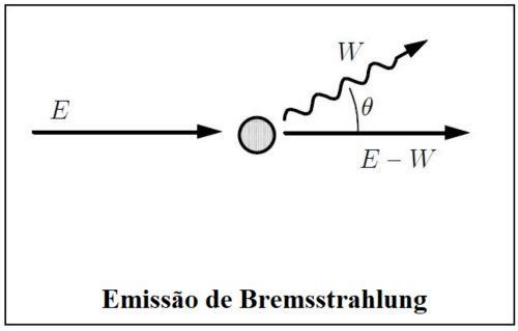


Fonte: (SALVAT, 2015)

3.1.3.3 Emissão de *Bremsstrahlung*

A emissão de *bremsstrahlung* também representa um tipo de colisão inelástica entre o elétron e a matéria, neste caso, com o núcleo atômico. O processo de interação tem como resultado a liberação de um fóton com energia entre zero e a energia cinética do elétron incidente (PODGORSAK, 2016). Na figura 7 é possível visualizar a Emissão de *Bremsstrahlung* onde o fóton é emitido com energia *E*, do elétron incidente, menos *W*, referente a energia transmitida. Em aceleradores lineares, trabalhando na faixa de mega voltagem, os fótons tendem a serem emitidos na mesma direção e sentido dos elétrons incidentes (PODGORSAK, 2005).

Figura 7 - Emissão de *Bremsstrahlung*

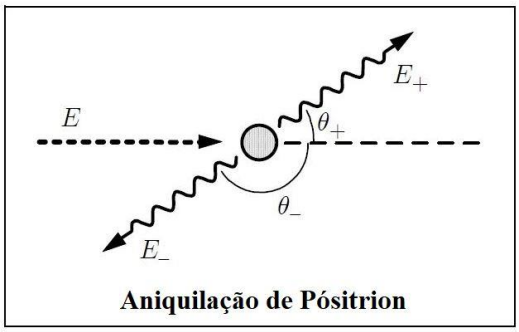


Fonte: (SALVAT, 2015).

3.1.3.4 Aniquilação de Pósitron

O pósitron é uma antipartícula do elétron com massa e energia idêntica, porém com cargas opostas. Os elétrons são negativos e os pósitrons são positivos. O seu processo de aniquilação ocorre quando o pósitron colide com um elétron orbital do átomo absorvedor e resulta na emissão de dois fótons de energia E = 0,511 MeV movendo-se quase que em direções opostas, com cerca de 180º, garantindo a conservação da carga da energia e do momento total (PODGORSAK, 2016). A figura 8 demonstra o processo de aniquilação onde um pósitron incidente com energia *E*, interage com um elétron orbital resultando na emissão de dois fótons em direções opostas (ângulos θ_+ e θ_-) *e* com energias E_+ e E_- iguais a 0,511 MeV.

Figura 8 - Aniquilação de Pósitron



Fonte: (SALVAT, 2015)

3.2 Método Monte Carlo

O Método Monte Carlo (MMC) é uma técnica que utiliza uma amostragem aleatória e outros métodos estatísticos para encontrar soluções para problemas matemáticos ou físicos. A técnica é útil quando a formulação exata que descreve o processo pode ser muito difícil, ou mesmo impossível de ser resolvida por métodos diretos. O MMC constrói um modelo

estocástico para representar o processo de interesse e por meio do uso de um conjunto de números aleatórios, de alta qualidade, realiza a amostragem da distribuição de probabilidade das funções definidas no modelo. O resultado é uma estimativa quantitativa de uma característica física do processo medido com um certo grau de confiança (KRAMER; CROWLEY; BURNS, 2000).

As técnicas de simulação de transporte de partículas por meio do MMC foram utilizadas pela primeira vez na década de 1940 para o desenvolvimento de armas atômicas. A partir de então, tornou-se uma aplicação essencial em diversas áreas da ciência tendo se mostrado como a ferramenta mais poderosa para modelar o transporte de radiação em radioterapia (VERHAEGEN; SEUNTJENS, 2003).

3.2.1 Método Monte Carlo aplicado ao transporte de radiação

Em essência, aplicado ao transporte de radiação, o MMC utiliza a física do transporte de partículas individuais, i.e., fótons, elétrons e pósitrons, simuladas uma a uma, com a finalidade de se determinar o padrão de deposição de doses. O trajeto de cada partícula no material é determinado por um gerador de números aleatórios e o rastreio de milhões de partículas possibilita a construção da distribuição de dose realizando a soma da deposição de energia de cada partícula no meio. (CHEN; XIAO; LI, 2014).

Na simulação do transporte de radiação pelo MMC, a história de uma partícula é vista como o caminho que ela percorre dentro de um meio material no qual sofre interações que podem ocasionar uma mudança da sua direção de movimento, uma perda de energia e produção de partículas secundárias. Se o número de partículas simuladas for suficientemente grande, informações quantitativas sobre o processo de transporte podem ser obtidas calculando a média sobre as histórias simuladas (SALVAT, 2015).

Todo o conjunto de eventos que ocorre com uma determinada partícula, desde sua emissão pela fonte até o momento em que ela é absorvida, é denominado de história da partícula. À medida que o número de histórias das partículas simuladas aumenta, melhora-se a qualidade dos resultados promovendo uma diminuição das incertezas estatísticas das grandezas de interesse (YORIYAZ, 2009). O problema decorrente da utilização do MMC é o longo tempo de simulação necessário para se obter um número suficiente de histórias a fim de se atingir a incerteza estatística desejada. A utilização da programação paralela, em GPUs, é considerada como uma das abordagens possíveis para promover a redução do tempo necessário para realização da simulação e tornar possível a sua utilização do MMC dentro de um tempo clinicamente aceitável (WANG *et al.*, 2016).

Existem diversos códigos para realização de simulação do transporte de radiação que foram implementados utilizando o Método Monte Carlo. Dentre eles, são comumente utilizados em radioterapia o MNCP (*Monte Carlo N-Particle*), GEANT4 (*Geometry and Tracking*) e o PENELOPE (*Penetration and Energy Loss of Positrons and Electrons*) (CINTRA, 2010).

3.2.2 Gerador de números pseudoaleatórios.

Toda a simulação Monte Carlo é efetuada por meio de amostragens das funções densidade de probabilidade. Essas amostragens são realizadas através de números aleatórios, sendo necessário um gerador de alta de qualidade para se obter uma simulação Monte Carlo com boa precisão. Os geradores de números aleatórios são baseados em algoritmos matemáticos que buscam simular a verdadeira aleatoriedade encontrada na natureza. No entanto, por serem regidos por algoritmos matemáticos, são formalmente chamados de números pseudoaleatórios (YORIYAZ, 2009).

Em geral, os algoritmos de amostragem aleatória são baseados no uso de números pseudoaleatórios uniformemente distribuídos no intervalo entre 0 e 1. Os geradores de números pseudoaleatórios chamados de congruenciais multiplicativos estão entre os bons algoritmos atualmente disponíveis. No entanto, esse tipo de gerador é capaz apenas de gerar uma sequência periódica da ordem de 10^9 números pseudoaleatórios o que pode não ser suficiente para evitar o seu reinício em uma única simulação. Uma excelente revisão sobre números pseudoaleatórios foi realizada por (JAMES, 1990), onde ele recomenda o uso de algoritmos que são mais sofisticados do que os simples congruenciais multiplicativos, como por exemplo o algoritmo RANECU, implementado no pacote PENELOPE, que possuiu a capacidade de gerar uma sequência periódica da ordem de 10^{18} números pseudoaleatórios (SALVAT, 2015).

3.3 PENELOPE

O programa PENELOPE foi originalmente escrito na linguagem Fortran 77 e teve sua primeira versão lançada em 1996. Foi inicialmente desenvolvido para simular a penetração e perda de energia de pósitrons e elétrons na matéria, o que dá origem ao seu nome. Posteriormente foi adicionado ao programa a possibilidade de se realizar a simulação do transporte de fótons com energias variando entre 50 eV até 1 GeV (SALVAT, 2015).

A versão 2014 do PENELOPE vem acompanhada de um manual com notas técnicas, escrito por Frances Salvat em 2015, um tutorial para execução de simulações exemplos e com uma vasta quantidade de comentários em seu código fonte. Esses documentos descrevem todas as suas funcionalidades e configurações necessárias para realização da simulação. Em termos

gerais, para se executar uma simulação é necessário criar o arquivo da geometria a ser simulada, gerar as tabelas com as propriedades e seções transversais dos materiais que formam os corpos e módulos da geometria e configurar o arquivo de programa principal com os parâmetros de entrada, de controle e de saída para obtenção dos resultados desejados.

3.3.1 Programa principal

O pacote de distribuição do PENELEPE-2014 possui dois exemplos de programas principais: o Pencyl (para realização de simulações com geometrias cilíndricas) e o Penmain (para realização de simulações com geometrias quadráticas). Por padrão, ambos descrevem fontes que emitem um único tipo de partícula podendo o Penmain simular fontes de feixes polienergético com uma configuração de espectro de energia. Os programas principais geram arquivos de saída com diversas informações de interesse como por exemplo: o número de partículas simuladas, a velocidade da simulação, a energia média depositadas nos corpos e, caso inserido os parâmetros necessários, o mapa de distribuição de dose.

A inserção dos parâmetros de entrada e de controle são lidos e interpretados pelo programa principal. Suas sub-rotinas e funções são responsáveis por dar início à realização da simulação de Monte Carlo realizando a chamada de outras sub-rotinas e funções como os presentes no arquivo Penelope.f que tratam das interações das partículas com a matéria.

3.3.1.1 Parâmetros de entrada

Para criação do programa principal é necessário inserir parâmetros de entrada com informações sobre tipo de partícula primária incidente (fóton, elétron ou pósitron), a energia inicial da partícula (monoenergética ou polienergética), as coordenadas da fonte de radiação, o tipo de feixe (cônico/circular ou retangular/quadrático), os nomes dos arquivos com informações sobre a seções transversais dos materiais envolvidos e seus parâmetros de controle, o nome do arquivo com informações da geometria simulada, os parâmetros de saída com os tipos de resultados desejados, o número desejado de partículas simuladas e o tempo máximo de execução de simulação.

3.3.1.2 Parâmetros de Controle

Os parâmetros de controle influenciam a velocidade e exatidão da simulação dos elétrons e pósitrons resultantes das interações do fóton com o meio material. Seus parâmetros são descritos conforme abaixo e devem ser especificados para cada material que compõe a

geometria:

a) E_{abs}: Energia de absorção local, caso a energia de uma partícula seja menor do que o valor estipulado neste parâmetro ela será localmente absorvida tendo sua história encerrada;
b) C_1: Deflexão angular média produzida entre colisões fortes de caráter elástica;
c) C_2: Máxima perda de energia fracionária produzida entre colisões fortes de caráter elástica;
d) W_{CC}: Perda de energia de corte produzida por colisões fortes de caráter inelástica;
e) W_{CR}: Perda de energia de corte produzida por colisões fortes com emissão de *bremsstrahlung*.

3.3.1.3 Parâmetros de saída e resultados

O pacote PENELOPE-2014 gera arquivos de saída de forma final ou parcial com um relatório global sobre a simulação e diversos resultados. O arquivo de saída Penmain-res.dat é gerado automaticamente ao se executar a simulação e contém informações como o tempo de execução, a velocidade da simulação, o número de partículas primárias simuladas, o tipo de partícula simulada, o número de partículas absorvidas, o número de partículas secundárias geradas e a energia média depositada nos corpos.

Para se obter o arquivo de saída 3d-dose-map.dat com informações sobre o mapa de distribuição de dose da geometria simulada é necessário inserir parâmetros de definição de uma caixa de dosagem fornecendo as coordenadas de seus vértices e o número de *voxels* desejado. Os parâmetros GRIDX, GRIDY e GRIDZ correspondem a um vetor com valores em centímetros de suas respectivas coordenadas e número de *bins*, ou *voxels* (<= 101, um valor padrão que pode ser modificado), na respectiva direção. De posse desse arquivo é possível gerar o mapa de distribuição de dose com a utilização de scripts de plotagem por meio de programas específicos, como por exemplo, o gnuplot.

3.3.2 Geometria da Simulação

A geometria da simulação é definida a partir de um arquivo texto de entrada com extensão ".geo". Ele é composto por uma sequência de blocos que definem os diferentes tipos elementos formadores da geometria (superfícies, corpos e módulos). O arquivo de geometria é interpretado pelo programa pengeom.f, que está habilitado para realizar a construção de qualquer sistema material que consista em corpos homogêneos limitados por superfícies

quadráticas. Por meio do programa pengeom.f é possível simular sistemas materiais complexos, de até 5.000 corpos, com diferentes materiais, e até 10.000 fronteiras. Suas sub-rotinas e funções são responsáveis por orientar a simulação das histórias das partículas quanto ao meio material na qual estão inseridas e realizar ajustes quando se cruza uma fronteira e se adentra a um corpo e/ou material diferente.

3.3.3 Tabela de materiais.

Os arquivos com as tabelas de propriedades físicas e seções transversais dos materiais são criados por meio do programa material.exe, que é obtido pela compilação do arquivo de programa material.f, executado dentro da pasta pdfiles. O PENELOPE-2014 traz em sua base de dados um total de 280 materiais já preparados divididos em 99 elementos da tabela periódica, identificados pelo seu respectivo número atômico, e 181 compostos e misturas que são lidos através do arquivo pdcompos.pen.

3.3.4 Redução da variância.

PENELOPE-2014 possui um arquivo de programa chamado penvared.f que contém sub-rotinas para a redução de variância com a finalidade de acelerar o cálculo das quantidades locais (por exemplo, energia média depositada ou fluência média das partículas) que são obtidas por meio da pontuação de contribuições de eventos de interação individual. O uso das técnicas como de divisão de partículas, roleta russa e força de interação promovem a redução do número total de histórias necessárias para alcançar uma determinada incerteza (WANG *et al.*, 2016). No entanto, a utilização de técnicas de redução de variância geralmente reduz a incerteza estatística de uma determinada quantidade em detrimento do aumento da incerteza de outras quantidades, sendo assim não recomendadas quando se busca uma descrição global do processo de transporte. (SALVAT, 2015).

3.3.5 Gerador de Números Pseudoaleatórios RANECU

O pacote de distribuição PENELOPE-2014 utiliza em suas simulações Monte Carlo o algoritmo gerador de números pseudoaleatórios chamado RANECU que foi codificado em Fortran por James na década de 90 (JAMES, 1990). O arquivo de programa rita.f possui a função RAND que tem como valores de entrada duas sementes de números inteiros, ISEED1 e ISEED2, que determinam a sequência de números aleatórios utilizados durante a simulação. Como saída, a função RAND retorna um único número aleatório no intervalo entre zero e um

com um período da ordem de 10^{18}, o que é praticamente inesgotável em simulações práticas conferindo ao algoritmo um status de um gerador de número pseudoaleatório de alta qualidade.

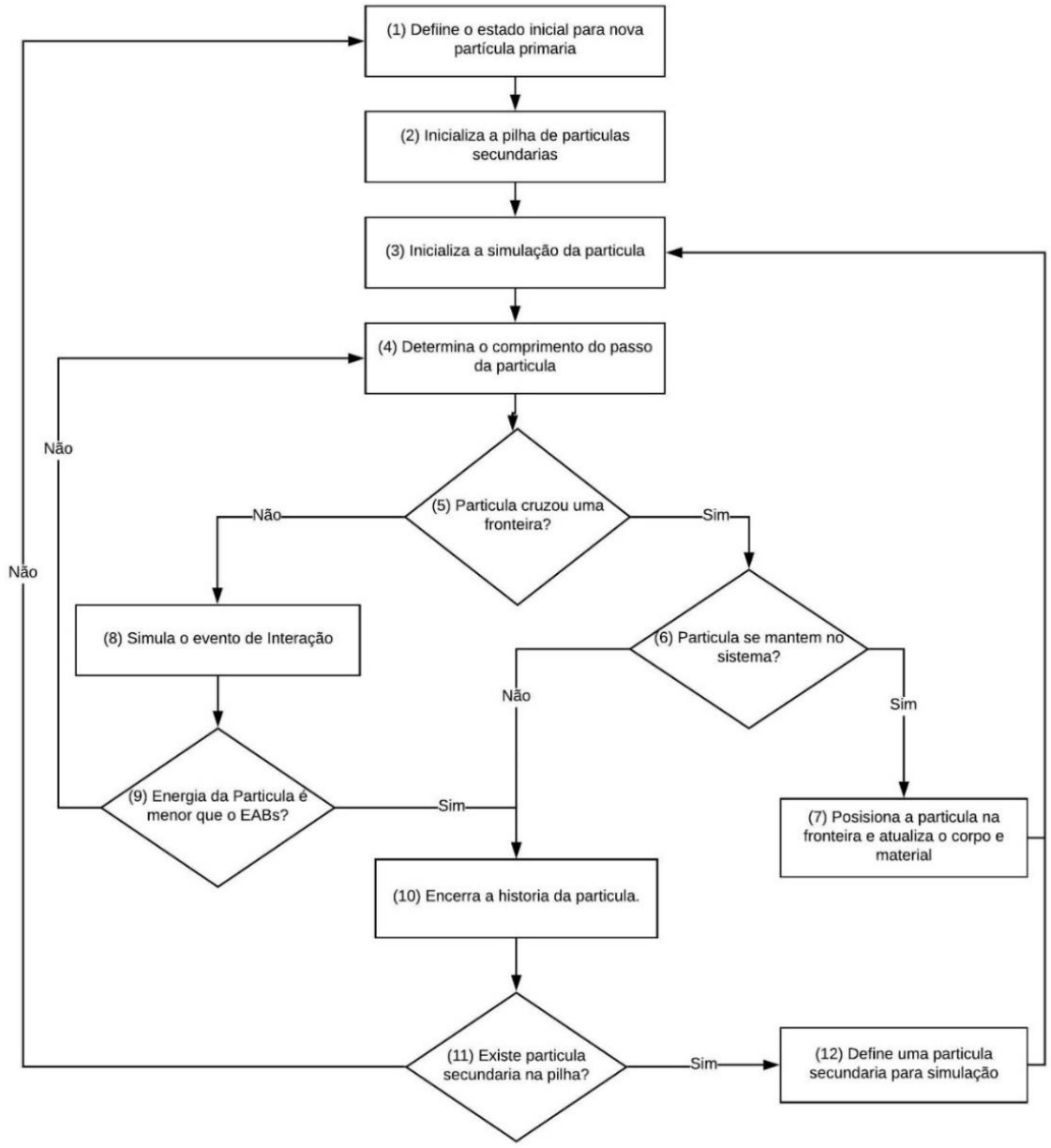
Em situações de cálculos paralelos é necessário certificar que os diferentes processadores gerem sequências verdadeiramente independentes uma das outras, ou seja, sem que haja repetição de números pseudoaleatórios. PENEPLOE-2014 traz em seu arquivo de programa rita.f a subrotina RAND0, que ao ser chamada para execução com um argumento de número inteiro entre 0 a 1000 inicia a função RAND com duas sementes diferentes que garantem a geração de uma sequência de números pseudoaleatórios da ordem de 10^{14}, sem que haja repetição e que por vezes é suficiente para execução de uma simulação Monte Carlo.

Durante toda a simulação os valores do comprimento do caminho percorrido pela partícula até a próxima interação, o mecanismo de interação envolvido, a mudança de direção e a perda de energia são variáveis aleatórias amostradas a partir de funções de densidade de probabilidade que dependem de números aleatórios uniformemente distribuídos no intervalo entre zero e um.

3.3.6 Fluxograma de execução

A figura 9 mostra o fluxograma de execução da simulação Monte Carlo no PENELOPE-2014 que possibilita a realização do transporte da partícula na matéria.

Figura 9 - Fluxograma de execução da simulação do transporte de partícula na matéria



Fonte: Desenvolvido pelo próprio autor com base em (SALVAT, 2015).

A etapa (1) do fluxograma define o estado inicial da nova partícula atribuindo os valores como o tipo de partícula, sua energia, coordenadas de posição, coordenadas de direção e movimento, o corpo e o material na qual está inserida. Na etapa (2) é chamada a função CLEANS para inicialização da pilha secundária. No passo (3) inicia-se a simulação da partícula com a chamada da função START. Em (4) é determinado o comprimento do passo que a partícula percorrerá com a chamada da função JUMP. No passo (5) é verificado se a partícula cruzou uma fronteira de material. Caso tenha cruzado uma fronteira a etapa (6) verifica se a

partícula continua no sistema de geometria delimitado. A partícula permanecendo no sistema de geometria, em (7) ela é posicionada na borda da fronteira do novo material e o processo retorna ao passo (3) para iniciar novamente a simulação da partícula. Na situação em que a partícula não cruzou uma fronteira, a etapa (8) realiza os eventos de interação da partícula com a matéria por meio da chamada da função KNOCK. O passo (9) verifica se após a interação ocorrida a energia da partícula se mantém maior do que a energia de absorção local do material. Em (10) ocorre o encerramento da partícula nas situações em que ela escapa do sistema de geometria ou no caso em a sua energia é menor do que a energia de absorção determinada para aquele material. Na etapa (11) é verificado se existe alguma partícula na pilha de partículas secundárias. Caso exista, a etapa (12) define a partícula secundária a ser simulada e retorna ao passo (3). Caso não existam partículas secundárias a serem simuladas, a etapa (12) direciona ao passo (1) para criação de uma nova partícula. Esse processo se repete até que todas as partículas estipuladas sejam simuladas ou o tempo determinado da simulação seja alcançado.

Esse fluxograma de execução do PENELOPE-2014 foi desenvolvido para execução das histórias das partículas primárias e secundárias de modo sequencial não estando preparado para uma simulação paralela.

3.4 Programação Paralela em GPU

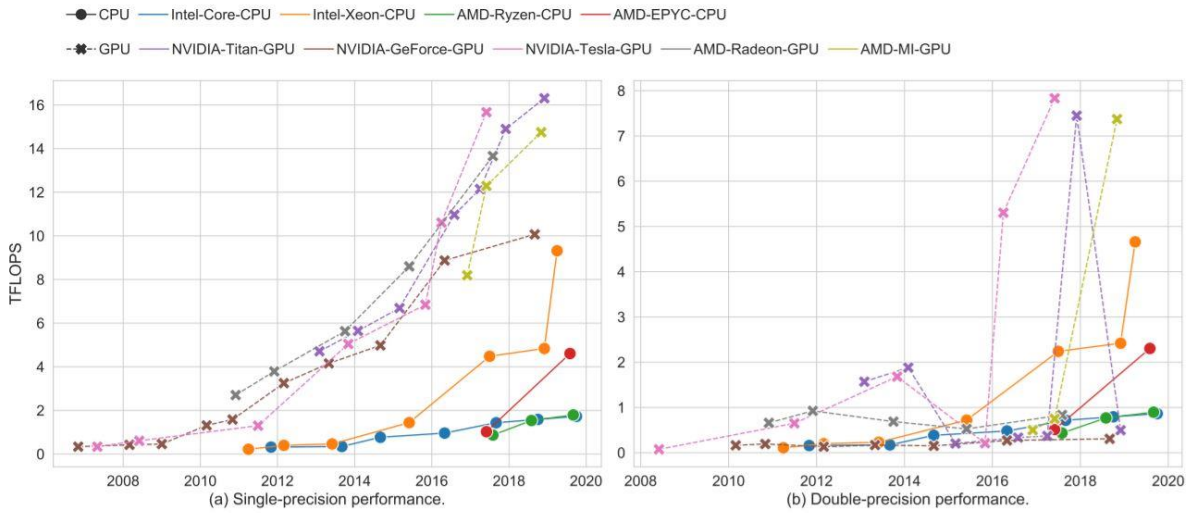
A computação paralela consiste na realização de muitos cálculos simultâneos com o princípio de que grandes problemas podem ser divididos em problemas menores e então resolvidos ao mesmo tempo em unidades de processamento diferentes. De modo a conseguir realizar esse tipo de operação, a utilização das GPUs (*Graphics Processing Unit*) ao invés de CPUs (*Central Processing Unit*) tornou-se uma opção interessante a ser considerada, pois a CPU é construída com alguns poucos núcleos otimizados para o processamento sequencial de tarefas enquanto a GPU é construída com milhares de núcleos menores e mais eficientes, projetados para lidar com múltiplas tarefas simultaneamente (SOSUTHA; MOHANA, 2015).

A título de comparação, um modelo básico de CPU, o Intel Core i7 1165G7, possui 4 núcleos de processamento, podendo ser virtualizados e operar com até 8 threads para execução de tarefas simultâneas (INTEL, 2021). Enquanto um modelo básico de GPU, como por exemplo a NVIDIA Geforce MX350, possui 640 núcleos de processamento para execução de tarefas de modo simultâneo (TECHPOWERUP, 2021).

Apesar do número superior de núcleos disponíveis na GPU é importante ressaltar que a sua utilização não, necessariamente, substitui por completo a utilização da CPU. Para a execução de programas, seja ele todo ou em parte, que utilizam poucos ou mesmo um único

thread, a latência de operação menor confere às CPUs um melhor desempenho. Por outro lado, quando o problema puder ser divido e modelado para ser executado por muitos *threads*, a maior taxa de transferência de execução das GPUs pode resultar em um maior poder de processamento (KIRK; WEN-MEI, 2016). A figura 10 mostra uma evolução e um comparativo das capacidades teóricas entre GPUs vs CPUs medido em FLOPS (*Floating-point Operations Per Second*).

Figura 10 - Comparativo entre GPUs e CPUs em operações de Precisão Simples e Dupla Precisão



Fonte: (SUN; AGOSTINI; DONG; KAELI, 2019)

Nota-se na Figura 10 que as GPUs mantêm uma boa vantagem frente às CPUs tanto em operações de precisão simples (*Float*) quanto de operações de dupla precisão (*Double*), atingindo valores de 16 *TeraFlops* o que colocaria uma única GPU na lista dos 500 melhores supercomputadores no ano de 2008 (SUN *et al.*, 2019).

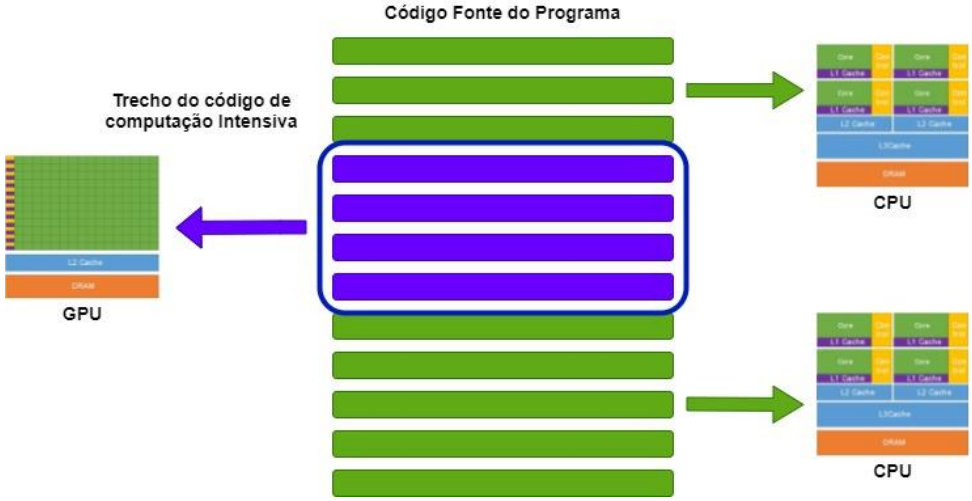
De forma a explorar a potencialidade do processamento paralelo das GPUs, a NVIDIA lançou em 2007 a plataforma de programação CUDA (*Compute Unified Device Architecture*) possibilitando que desenvolvedores utilizem a linguagem C++ para resolver problemas computacionais complexos de forma mais eficiente do que em uma CPU (NVIDIA, 2021).

3.4.1 Plataforma de programação paralela CUDA.

Os programas desenvolvidos na plataforma CUDA são escritos em modo heterogêneo, possuindo uma parte do código em modo sequencial, executada na CPU que é chamada de *host*, e uma parte do código em modo paralelo, executado na GPU que é chamada de *device*. No *host*

é executada a função principal do programa, o gerenciamento e transferências de dados entre memórias e a realização da configuração para a utilização da GPU. No *device* são executadas funções especiais, chamadas de *kernels*, que geralmente ficam responsáveis pela execução de trechos do código que possuem uma computação intensiva e que podem obter ganhos de processamento com o uso de muitos núcleos. (MISIC; DURDEVIC; TOMASEVIC, 2012). A figura 11 demostra o processo de execução de um programa na plataforma CUDA.

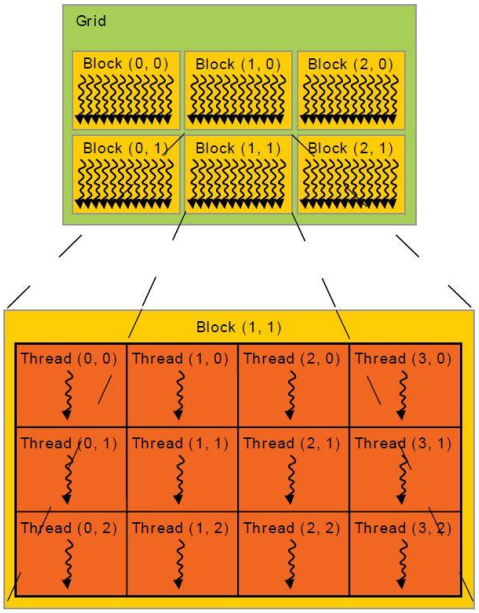
Figura 11 - Execução de uma aplicação na plataforma de programação CUDA



Fonte: Desenvolvido pelo próprio autor com base em (MISIC; DURDEVIC; TOMASEVIC, 2012) e (NVIDIA, 2021).

Para execução de um programa em CUDA precisamos determinar o conjunto de threads que serão executados pelo *kernel*. Os threads são organizados em uma hierarquia de dois níveis, conforme figura 12. No nível inferior, os threads são organizados em blocos (*blocks*) e podem ter uma, duas ou três dimensões. Os blocos, por sua vez, são organizados em uma grade (*grid*) e podem ter também uma, duas ou três dimensões. A quantidade de blocos e o tamanho da grade são limitados pelos recursos disponíveis do dispositivo utilizado (BARLAS, 2022).

Figura 12 - Hierarquia de *Grid*, *Blocks* e *Threads*



Fonte: (NVIDIA, 2021).

Na figura 13 podemos ver alguns exemplos de chamadas do *kernels* com os respectivos conjuntos de *threads*:

Figura 13 - Chamadas de kernels e seus respectivos conjuntos de *threads*

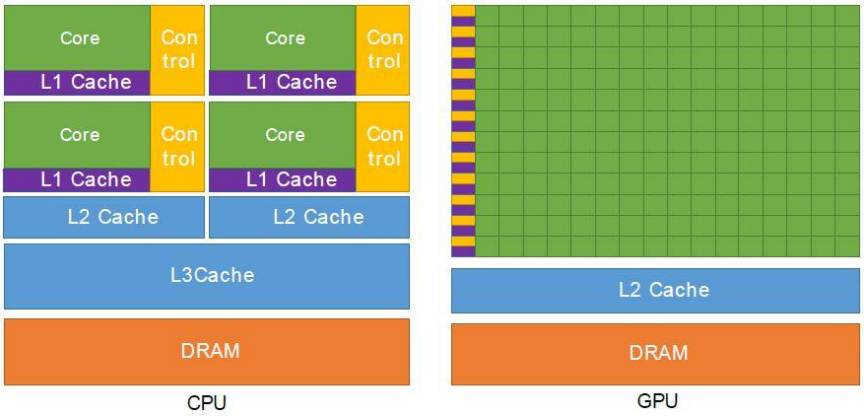
```
dim3 block(3,3,3);
dim3 grid(20,100);
kernel<<grid, block>>>(); //kernel de 20x100 blocks com 3x3x3 threads por block.
kernel<<<24,128>>>(); //kernel de 24 blocks com 128 threads por block.
dim3 b(3,4);
dim3 g(2,3);
kernel<<g, b>>>();//kernel de 2x3 blocks com 3x4 threads por block. Exemplo da Figura 12.
```

Fonte: Desenvolvido pelo próprio autor com base em (BARLAS, 2022).

3.4.1.1 Arquitetura de uma GPU

Ao se analisar a arquitetura da GPU (figura 12) nota-se que ela foi projetada para que a maior parte dos seus transistores sejam dedicados ao processamento, priorizando a existência de um maior número de núcleos em detrimento ao armazenamento em cache ou controle de fluxo de dados, quando comparado à arquitetura da CPU. Dedicar mais transistores ao processamento de dados, como por exemplo, cálculos de ponto flutuante, é extremamente benéfico para execução de operações paralelas (NVIDIA, 2021).

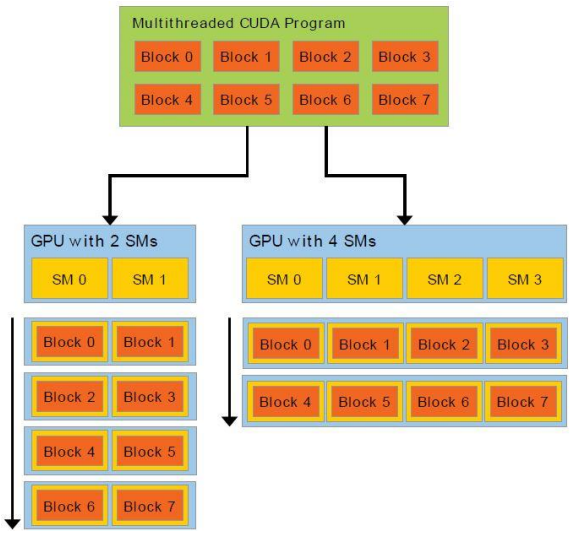
Figura 14 - Arquitetura da CPU e da GPU



Fonte: (NVIDIA, 2021).

A arquitetura das GPUs NVIDIA foi construída em torno de uma matriz escalonável de *Stream Multiprocessors* (SMs). Quando um programa CUDA invoca um *kernel* para execução, os blocos, que são formados por *threads*, são enumerados e distribuídos para os SMs que estão atualmente disponíveis para execução. Conforme podemos visualizar na figura 13, ao se utilizar uma GPU com maior poder de processamento, consequentemente um maior número de SMs disponíveis, a GPU se encarrega de direcionar corretamente os blocos de *threads* para a execução nos SMs de modo a diminuir a fila de execução tornando a plataforma CUDA altamente escalável. (NVIDIA, 2021).

Figura 15 - Escalabilidade da Arquitetura da GPU NVIDIA



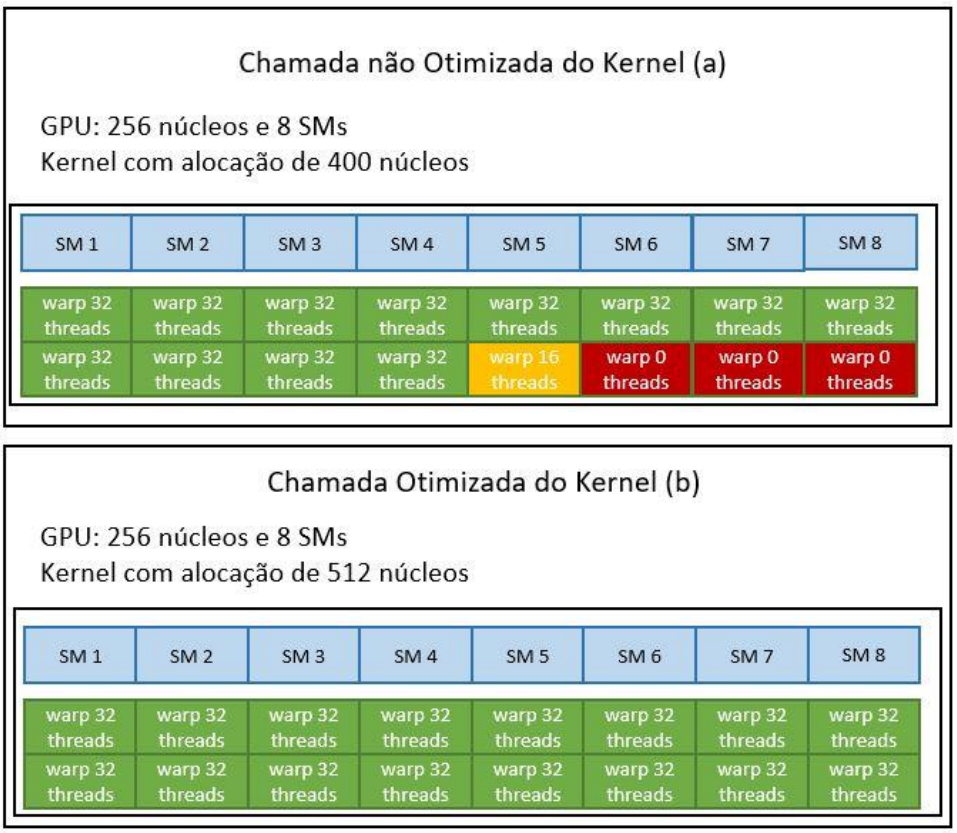
Fonte (NVIDIA, 2021).

3.4.1.2 Warps

Os SMs criam, gerenciam, agendam e executam os *threads* em grupos de 32 *threads* chamadas de *warps*. A plataforma CUDA segue o modelo SIMT (*Single Instruction, Multiple Thread*) onde todos os *threads* em um *warp* buscam e executam a mesma instrução por vez, de modo que, a eficiência máxima é obtida quando todos os 32 threads convergem em seu caminho de execução. Caso um dos *threads* percorra um caminho diferente dos outros, todo o conjunto dos 31 *threads* restantes do *warp* ficarão inativos aguardando a conclusão do processo do *thread* ativo divergente (NVIDIA, 2021).

O poder computacional de uma GPU é consideravelmente determinado pela quantidade de núcleos e SMs disponíveis. As novas GPUs são lançadas escalonando a quantidade desses atributos como o principal fator de ganho de desempenho. Portanto, para se obter uma melhor eficiência na chamada de um kernel deve-se levar em consideração a alocação de *threads* sempre múltiplos de 32 para que a GPU organize a execução dos *warps* em seus respectivos SMs de modo a obter uma máxima ocupação do hardware disponível, conforme podemos visualizar na figura 14 (BARLAS, 2022).

Figura 16 - Chamada de Kernel Otimizada



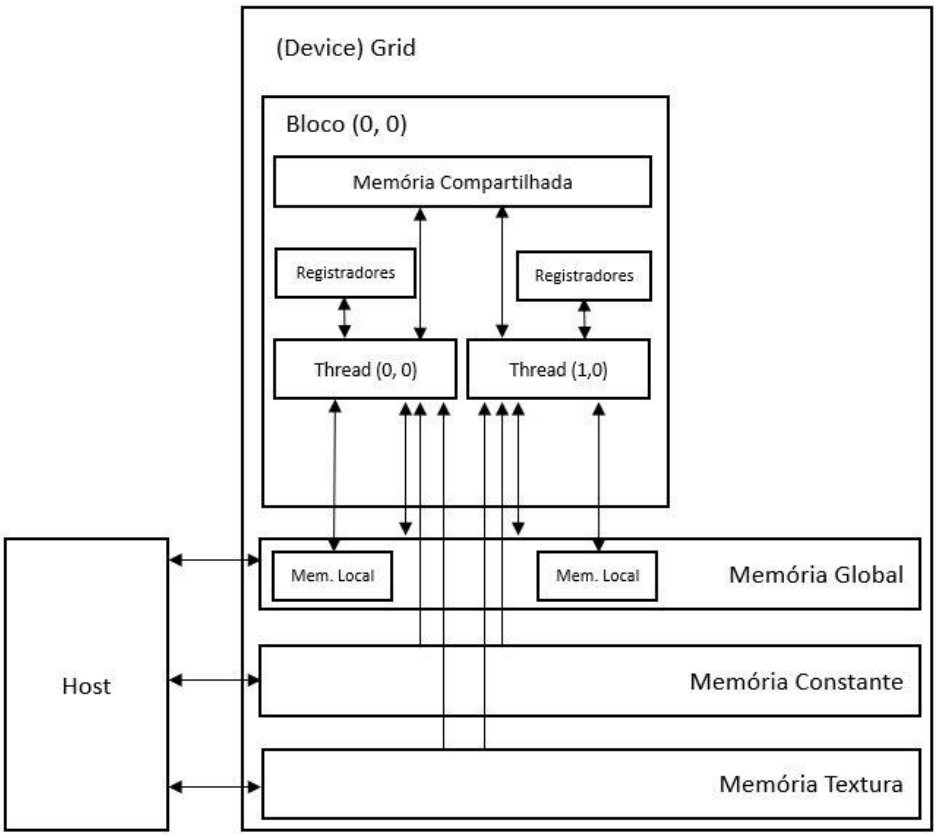
Fonte: Desenvolvido pelo próprio autor com base em (BARLAS, 2022).

Na figura 14 podemos visualizar uma chamada de kernel não otimizada em (a) e uma chamada de kernel otimizada em (b). Quando se realizada uma chamada de kernel alocando núcleos em uma quantidade não múltipla de 32, inevitavelmente teremos um *warp* com uma quantidade menor de threads do que a sua capacidade e poderemos ter também *warps* inativos durante sua execução promovendo uma perda de desempenho. Ao passo que, se for realizada uma alocação múltipla de 32, de modo a preencher todos os *warps* possíveis nos SMs, teremos um ganho de desempenho na execução do kernel.

3.4.1.3 Tipos de Memória da GPU

O acesso e o gerenciamento adequado da memória são partes importantes de qualquer linguagem de programação tendo um impacto particularmente grande na computação de alto desempenho. Na figura 15 podemos visualizar que o modelo de memória da plataforma CUDA oferece diferentes tipos de memórias programáveis tendo uma hierarquia de acesso com diferentes latências, larguras de banda e capacidades. Em geral, à medida que a latência de acesso para a memória aumenta, a sua capacidade também aumenta (KIRK; WEN-MEI, 2016).

Figura 17 - Visão geral do modelo de memória da GPU NVIDIA

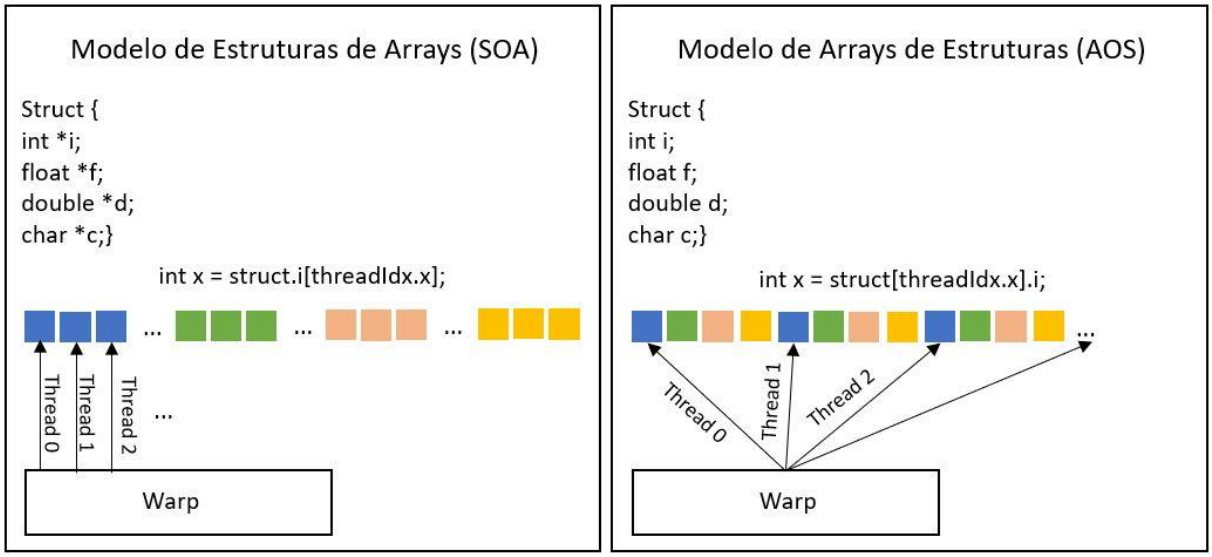


Fonte: Adaptado pelo próprio autor com base em (KIRK; WEN-MEI, 2016).

a) Registrador: é o tipo de memória que possui a menor latência e consequentemente o menor tamanho na GPU. A variáveis de registro são exclusivas para cada *thread* e possuem o tempo de vida do *kernel*. Devido à pequena quantidade disponível para utilização e de ter que ser dividida entre os *threads* em execução, o uso de poucos registrados por kernel possibilita a execução de mais blocos de *threads* simultâneos aumentando a ocupação e melhorando o desempenho da aplicação;
b) Memória Local: quando as variáveis de registros de um *kernel* extrapolam a sua capacidade elas são alocadas para a memória local que reside no mesmo chip físico da memória global possuindo uma alta latência e baixa largura de banda;
c) Memória Compartilhada: possui uma largura de banda maior e uma latência menor do que a memória global. Está para GPU assim como o cache L1 está para a CPU, com uma diferença de poder ser gerenciada possuindo o tempo de vida do bloco de *threads*. Assim, como o uso dos registrados limita a ocupação de um kernel, cada SM tem uma quantidade limitada de memória compartilhada que é particionada para uso entre os blocos de threads e extrapolar a alocação dessa memória limita o número de *warps* ativos;
d) Memória Constante: os valores guardados nessa memória devem ser previamente declarados podendo ser apenas acessados pelo kernel. Possuem um melhor desempenho quando todos os *threads,* em um *warp,* leem o mesmo endereço de memória pois em um único acesso, à informação é entregue a todos os *threads*. Uma informação interessante para guardar na memória constante são coeficientes de fórmulas matemáticas;
e) Memória Textura: é um tipo de memória global somente leitura otimizada para trabalhar com dados em matrizes 2D. Para alguns aplicativos pode oferecer um melhor desempenho do que a memória global, no entanto, em alguns casos pode ser mais lento;
f) Memória Global: é a maior memória disponível na GPU, da ordem de *Gigabytes*, e consequentemente a de maior latência e menor largura de banda. Alocada e liberada pelo *host*, possui o tempo de vida da aplicação e pode ser acessada por qualquer *thread* do *kernel*, o que ocasiona uma concorrência de acesso e nos confere alguns cuidados especiais na forma de organizar os seus dados e no momento de acessá-los.

Um ponto importante a se destacar trata-se da forma de organização e de acesso aos dados na memória global. Para se obter o melhor desempenho deste tipo de memória deve-se garantir que o acesso sequencial aos dados seja feito de modo adjacente. O barramento da GPU, que faz acesso à memória global, tem uma largura de banda de 32 bytes por ciclo de *clock* possibilitando que um *warp*, consumindo dados do tipo *float* (32 threads x 4 bytes), seja completamente atendido com apenas 4 ciclos de acesso à memória. De modo a conseguir esse acesso sequencial adjacente, o modelo SIMT, presente no CUDA, exige que os dados sejam organizados em estruturas de *arrays* (SOA) ao invés de *arrays* de estruturas (AOS) comumente utilizada em código sequencial na CPU. A figura 16 ilustra a diferença de organização e a forma de acesso aos dados na memória global (HAN; SHARMA, 2019).

Figura 18 - Acesso a memória global de modo adjacente x modo não adjacente

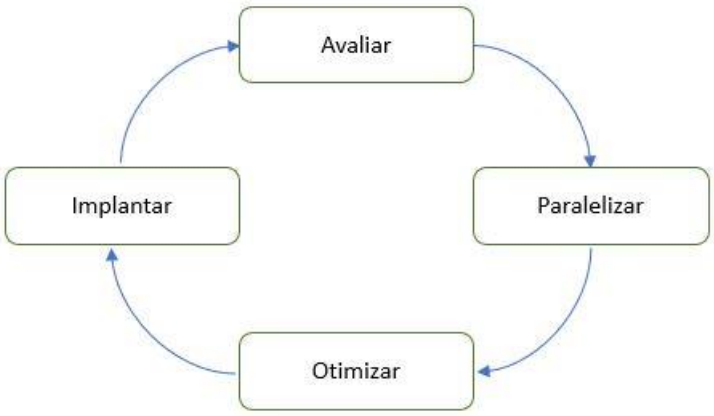


Fonte: Adaptado pelo próprio autor com base em (HAN; SHARMA, 2019).

3.4.1.4 Avaliar, Paralelizar, Otimizar e Implantar

O processo de desenvolvimento de uma aplicação na plataforma de programação CUDA se beneficia do ciclo de *design* Avaliar, Paralelizar, Otimizar e Implantar. Esse ciclo ajuda os desenvolvedores a identificar rapidamente partes do código fonte que possam se beneficiar mais prontamente da aceleração da GPU, possibilita perceber rapidamente esse benefício e começar a aproveitar as acelerações resultantes da implementação o mais breve possível (NVIDIA, 2021). A figura 17 demonstra como esse ciclo deve ser abordado.

Figura 19 - Ciclo de design para desenvolvimento na plataforma CUDA



Fonte: Adaptado pelo próprio autor com base em (NVIDIA, 2021).

a) Avaliar: A primeira etapa consiste em estudar o código fonte na busca por partes que possuam um maior tempo de execução e que possam obter ganhos de desempenho com a paralisação;
b) Paralelizar: Implementação dos trechos de códigos que foram identificados como candidatos a paralização;
c) Otimizar: A otimização é feita de maneira incremental à medida que são identificados gargalos de execução no código fonte ou disponibilizadas novas técnicas e estratégias de paralelização;
d) Implantar: É indicado que a disponibilização para utilização do software paralelizado seja feita o mais rápido possível para que se possa usufruir dos ganhos de performance, uma vez que é um processo cíclico e poderá ser aperfeiçoado.

3.5 Revisão de literatura

Foi realizada uma busca por trabalhos correlatos visando auxiliar e entender se a implementação de um algoritmo de Monte Carlo aplicado ao transporte de radiação, por meio de execução em unidades de processamento gráfico, pode ser capaz de produzir uma diminuição no tempo computacional necessário para se obter resultados como a distribuição de dose no corpo.

A apresentação dos trabalhos encontrados é organizada por ano de publicação em ordem crescente. Foram analisados nos trabalhos o desenvolvimento ou adaptação de um algoritmo Monte Carlo aplicado ao transporte de radiação, o cálculo de distribuição de dose, a plataforma

de programação utilizada, os ambientes de tratamento utilizados para a simulação, os recursos de hardware disponíveis em GPU e CPU e o fator de desempenho entre os dois tipos processadores.

Em 2014, (BELLEZZO; DO NASCIMENTO; YORIYAZ, 2014) implementaram o algoritmo CUBMC, baseado no PENELOPE, utilizando a plataforma de programação CUDA com estrutura de geometria modelada em *voxels* e uso do método de redução de variância WoodCook. Realizaram simulações em quatro ambientes distintos compostos por água, osso, pulmão e osso + vácuo. Utilizando uma GPU Nvidia GTX 560-TI alcançaram um fator desempenho entre 50 e 100 vezes mais rápido do que a mesma simulação executada em uma CPU Intel Xeon 2.7 GHZ utilizando o algoritmo PENELOPE.

Também em 2014, (HENDERSON *et al.*, 2014) publicaram a implementação do algoritmo G4CU, baseado no Geant4, desenvolvido na plataforma CUDA. Aplicaram a simulação em objetos simuladores baseados em *voxels* formados de água homogênea e com três placas formadas por água, pulmão e osso. Executando o algoritmo G4CU em uma GPU Nvidia Tesla K20, obtiveram um fator de desempenho da ordem de 40 vezes mais rápido comparado a mesma simulação executada pelo algoritmo Geant4 em uma CPU Intel Xeon X5680 3.33 GHz.

Em 2016, (WANG *et al.*, 2016) implementaram o algoritmo GPENELOPE, baseado no PENELOPE, na plataforma de programação CUDA com modificações para incluir transporte voxelizado e rastreamento com redução de variância pelo método Woodcock. Realizaram a simulação em 16 planos de tratamento com materiais que possibilitam simular os tecidos do estômago, pulmão, fígado, glândula adrenal, pâncreas, baco e mama. Executando o algoritmo desenvolvido em uma GPU Nvidia Tesla K80, alcançaram um fator de desempenho de 152 vezes mais rápido comparado a mesma simulação realizada utilizando o algoritmo PENELOPE em uma CPU Intel Xeon E5 2630 v3.

No ano de 2020, (MIRZAPOUR *et al.*, 2020) realizaram a publicação do algoritmo DOSXYZgpu, baseado no EGSnrc, desenvolvido na plataforma CUDA. As simulações foram realizadas em Fantômas voxelizados multicamadas simulando água-osso-água e água-pulmão-água e em Fantômas representando caso real de imagem de tomografia computadorizada. O algoritmo desenvolvido foi executado em uma GPU Nvidia GTX 1080 Ti 11 GB e comparado com o desempenho do algoritmo EGSnrc executado em uma CPU Intel i7 4790K 4.0 GHz e 16 GB de memória RAM DDR4. A execução da simulação na GPU foi até 205 vezes mais rápida em relação à execução em CPU.

4 MATERIAIS E MÉTODOS

Este capítulo está dividido em cinco etapas: na primeira etapa foi realizado um estudo com a finalidade de elencar todos parâmetros e sub-rotinas, contidos nos arquivos do pacote PENELOPE-2014, que são necessárias para realização da simulação Monte Carlo aplicada ao transporte de radiação e que possibilite obter como resultados o mapa de distribuição de dose e a dose depositada em cada órgão da geometria estudada; na segunda etapa foram criadas três geometrias e programas principais de modo a possibilitar a realização da simulação de um tumor presente no cérebro, no pulmão e na próstata e realizada a alteração do programa responsável pela geração do arquivo plotdose.dat que permite a plotagem do mapa de distribuição de dose; na terceira etapa foi realizada a transcrição de partes do código fonte do PENELOPE-2014 para a linguagem de programação C++ tomando por base as informações colhidas da primeira etapa e tomados cuidados especiais com a diferenciação entre a linguagem de programação Fortran e C++; na quarta etapa foi realizada a adaptação da versão sequencial em C++ para uma versão paralela em CUDA C++, com implementações otimizações inerentes da nova plataforma de programação; na quinta e última etapa são apresentados os ambientes de simulação com configuração de hardwares e softwares para comparação de exatidão e desempenho.

4.1 Pacote PENELOPE-2014

Nesta etapa foi realizado um estudo do documento oficial e do código fonte dos arquivos penmain.f, penelope.f, pengeom.f, timer.f e rita.f . Esse estudo visou elencar os parâmetros, sub-rotinas e funções necessárias para realização da simulação Monte Carlo aplicado ao transporte de radiação de modo a obter como resultados o mapa de distribuição de dose e a dose depositada em cada órgão da geometria estudada.

4.1.1 Análise do arquivo Penmain.f

O primeiro arquivo analisado foi o programa principal Penmain.f. Nele foram encontrados os principais parâmetros e sub-rotinas necessários para se criar e dar início à execução da simulação. No quadro 1 são demonstrados os parâmetros do programa principal Penmain.f

Quadro 1 - Parâmetros do arquivo Penmain.f

Parâmetro	Descrição
TITLE	Título da Simulação
SKPAR	Tipo da partícula primária: (1:Eletron; 2: Fóton; 3:Positron)
SENERG	Energia do feixe Mono Energético
SPECTR	Espectro de Energia do Feixe
SPOSIT	Coordenadas do centro de origem da fonte. (SX0, SY0, SZ0)
SCONE	Feixe cônico/circular (Theta, Phi, Alpha)
SRECTA	Feixe retangular quadrático (ThetaL, ThetaU, PhiL, PhiU)
RSEED	Sementes do gerador de número pseudoaleatório (ISeed1, ISeed2)
GRIDX	Coordenadas X da caixa de Dosagem e o número de *voxels*
GRIDY	Coordenadas Y da caixa de Dosagem e o número de *voxels*
GRIDZ	Coordenadas Z da caixa de Dosagem e o número de *voxels*
RESUME	Arquivo de restauração
DUMPTO	Nome do arquivo de Saída
DUMPP	Período de gravação em segundos
NSIMSH	Número de partículas simuladas
TIME	Tempo da simulação em segundos
C1	Deflexão angular média
C2	Perda fracionária máxima de energia
WCC	Perda de energia para colisões inelásticas duras
WCR	Perda de energia para emissão de bremmstrahlung
EABS	Energia de absorção no material

Fonte: (SALVAT, 2015)

No quadro 2 são elencadas as sub-rotinas necessárias para dar início à execução da simulação e a obtenção dos resultados:

Quadro 2 - Sub-rotinas do arquivo Penmain.f

Sub-rotina	Descrição
PMRDR	Lê o arquivo de entrada e inicializa o Penelope.f e o Pengeom.f
SHOWER	Inicia a simulação de uma nova história de partícula e registra a pontuação de quantidades relevantes.
SDOSE	Escreve os resultados do mapa da distribuição de dose e a dose nos corpos

Fonte: (SALVAT, 2015)

4.1.2 Análise do arquivo Penelope.f

A análise do código fonte do arquivo Penelope.f possibilitou determinar as funções sequenciais decorrentes da chamada das sub-rotinas PMRDR e SHOWER, além dos demais parâmetros necessários para realização da simulação e dos atributos responsáveis pelo armazenamento dos dados que possibilitam acompanhar o trajeto das partículas dentro do material conforme exposto no Quadro 3.

Quadro 3 - Parâmetros para armazenamento dos dados das partículas

Parâmetro	Descrição
E	Energia da partícula
POSITION	Coordenadas de posição (x, y, z)
DIRECTION	Cossenos diretores (u, v, w)
WGHT	Peso da partícula
KPAR	Tipo de partícula
IBODY	Corpo atual
MAT	Material atual
ILB	Origem da partícula

Fonte: (SALVAT, 2015)

Para realizar a simulação da interação das partículas com a matéria são necessárias as sub-rotinas apresentadas no Quadro 4.

Quadro 4 - Sub-rotinas de interação das partículas com a matéria

Elétron	**Descrição**
EELa	Colisão Elástica
EINa	Colisão Inelástica
EBRa	Efeito Bremmstrahlung
Fóton	**Descrição**
GCOa	Espalhamento Compton (Incoerente)
GRAa	Espalhamento Raylegh (Coerente)
GPHa	Efeito Fotoelétrico
GPPa	Produção de Pares
Pósitron	**Descrição**
PELa	Colisão Elástica
PINa	Colisão Inelástica
PANa	Aniquilação de Pósitrons

Fonte: (SALVAT, 2015)

Além das sub-rotinas que permitem simular a interação da partícula com a matéria, o arquivo possui sub-rotinas que possibilitam simular o trajeto dessas partículas no material conforme disposto no quadro 5.

Quadro 5 - Sub-rotinas para simular o trajeto das partículas

PEINIT	Entrada de dados de materiais e inicialização de rotinas de simulação
CLEANS	Inicializa a pilha de partículas secundárias onde seus estados iniciais são armazenados.
START	É chamada antes de iniciar a simulação de uma nova partícula (primária ou secundária) e quando uma partícula cruza uma fronteira
JUMP	Determina o espaço percorrido por uma partícula até o próximo evento de interação
KNOCK	Simula o evento de interação, calcula novos valores de energia, direção de movimento e armazena os estados iniciais das partículas secundárias geradas, se houver.
DIRECT	Calcula os novos cossenos diretores da partícula após uma determinada colisão com espalhamento em ângulos polares e azimutal.
STORES	Armazena o estado inicial de uma nova partícula secundária
SECPAR	Fornece o estado inicial de uma partícula secundária e a remove da pilha.

Fonte: (SALVAT, 2015)

4.1.3 Análise do arquivo Pengeom.f

Para possibilitar a construção das geometrias da simulação foi analisado o arquivo Pengeom.f. Nele foram encontrados os parâmetros que permitem descrever as superfícies, os corpos e os módulos da geometria conforme os quadros 6, 7 e 8, respectivamente.

Quadro 6 - Parâmetros que descrevem as superfícies

Parâmetro	Descrição
AO	Forma explicita. Termo independente
A1	Forma explicita. AX, AY, AZ
A2	Forma explicita. AXX, AYY, AZZ
INDICES	Forma implícita. I1, I2, I3, I4, I5
SHIFTS	Vetor de deslocamento. X-Shift, Y-Shift, Z-Shift
SCALES	Fatores de escala. X-Scale, Y-Scale, Z-Scale
ANGULES	Ângulos Euler: Omega, Theta, Phi

Fonte: (SALVAT, 2015)

Quadro 7 - Parâmetros que descrevem os corpos

Parâmetro	Descrição
ID	Identificação do corpo
DSMAX	Comprimento máximo do passo no corpo
EABSB	Energia de absorção no corpo

Fonte: (SALVAT, 2015)

Quadro 8 - Parâmetros que descrevem os módulos

ID	Identificação do módulo
SHIFTS	Fatores de escala. X-Scale, Y-Scale, Z-Scale
ANGULES	Ângulos Euler: Omega, Theta, Phi

Fonte: (SALVAT, 2015)

Ainda no arquivo Pengeom.f foram identificadas sub-rotinas necessárias para obtenção dos resultados desejados conforme o quadro 9.

Quadro 9 - Sub-rotinas das geometrias do material

CLONE	Possibilita a clonagem de um módulo
LOCATE	Determina o corpo e o material que contém as coordenadas de posição X, Y, Z.
STEP	Lida com a parte geométrica da simulação e o caminho percorrido pela partícula

Fonte: (SALVAT, 2015)

4.1.4 Análise do arquivo Timer.f

O arquivo Timer.f possui as sub-rotinas que fornecem o tempo de execução da simulação em segundos. Ao término da leitura do arquivo de geometria e do arquivo do programa principal é realizado a chamada a sub-rotina TIMER que inicia a contagem do tempo e após transcorrida uma quantidade determinada de partículas simuladas ou um tempo pré-determinado executa-se a sub-rotina CPUTIM que retorna a diferença do tempo inicial para o tempo atual em que está a simulação. Essas funções foram adaptadas para a linguagem C++ seguindo a mesma lógica de execução na versão original.

4.1.5 Análise do arquivo Rita.f

A análise do arquivo Rita.f possibilitou identificar as funções RAND e RAND0,

necessárias para geração de números pseudoaleatórios que são fundamentais para a execução da simulação Monte Carlo. O PENELOPE-2014, executado em modo sequencial, em uma única CPU, faz utilização da função RAND com as sementes, ISEED1 e ISSED2, declaradas no programa principal. A Função RAND0 possibilita e execução da função RAND garantindo que sejam produzidas sequências verdadeiramente independentes de números pseudoaleatórios em modo de execução em paralelo por diversos CPUs. Isso é feito alimentando cada processador com pares de sementes diferentes que resultam na geração de várias sequências de números pseudoaleatórios, mas que estão distantes o suficiente uma do outra para evitar a sobreposição.

O código existente no arquivo traz uma lista de 1000 pares de sementes que possibilitam à função RAND a geração de uma sequência longa da ordem de 10^{14} números pseudoaleatórios.

4.2 Preparação do ambiente de simulação

Nesta etapa foram criados arquivos para permitir a simulação de geometrias com tecidos tumorais no cérebro, pulmão e próstata; arquivos de programas principais com os parâmetros de entrada, parâmetros de controle e parâmetros para geração dos resultados esperados e a geração dos arquivos de materiais utilizados na simulação.

4.2.1 Arquivos de geometrias.

As geometrias referentes à cabeça, ao pulmão e à próstata foram criadas utilizando como referência o arquivo phantom.geo que é distribuído junto com o pacote PENELOE-2014. Adicionalmente a isso, foram inseridos nódulos com dimensões adequadas de modo a simular um câncer na cabeça e no pulmão. Para o câncer de próstata, foi adicionado um corpo próximo à parte inferior da bexiga com dimensões ligeiramente aumentadas do padrão de uma próstata considerada normal para um homem adulto. O material utilizado para os tecidos tumorais foi o *TissueSoft*, identificação 263, disponibilizado pelo pacote PENELOPE-2014 em sua base de dados de materiais.

O quadro 10 traz a composição dos materiais formadores das três geometrias criadas.

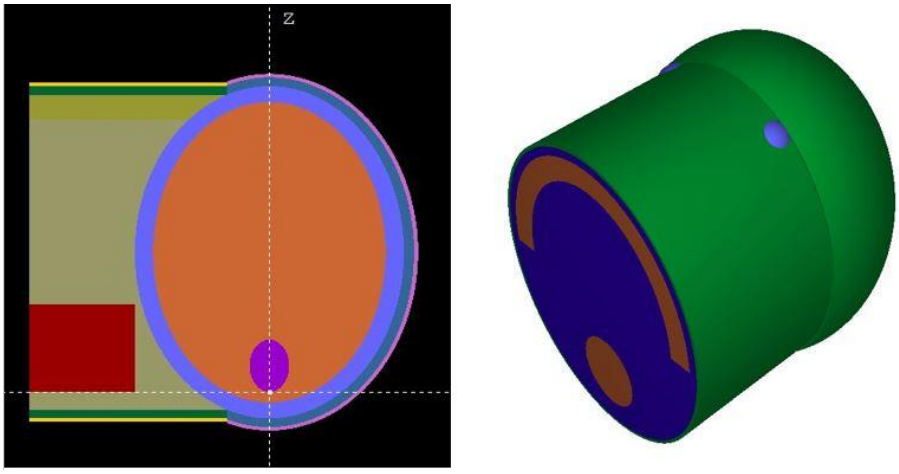
Quadro 10 - Materiais que compõem as geometrias

Cabeça		**Pulmão**		**Próstata**	
Corpo	**Material**	**Corpo**	**Material**	**Corpo**	**Material**
Ar	*Air* (104)	Ar	*Air* (104)	Água	*Water* (278)
Crânio	*Bone* (119)	Costelas	*Bone* (119)	Ar	*Air* (104)
Tumor	*Tissue* (263)	Tumor	*Tissue* (263)	Pelvis	*Bone* (119)
Pele	*Skin* (251)	Pele	*Skin* (251)	Próstata	*Tissue* (263)
Cérebro	*Brain* (123)	Coração	*Muscle* (203)	Bexiga, reto	*Muscle* (203)
Olhos	*Eye* (156)	Pulmão	*Lung* (191)	Pele	*Skin* (251)

Fonte: Desenvolvido pelo próprio autor com base em (SALVAT, 2015)

As figuras 18, 19 e 20 mostram as três geometrias em 2D e 3D geradas pelos programas gview2d.exe e gview3d.exe disponibilizados pelo pacote PENELOPE-2014.

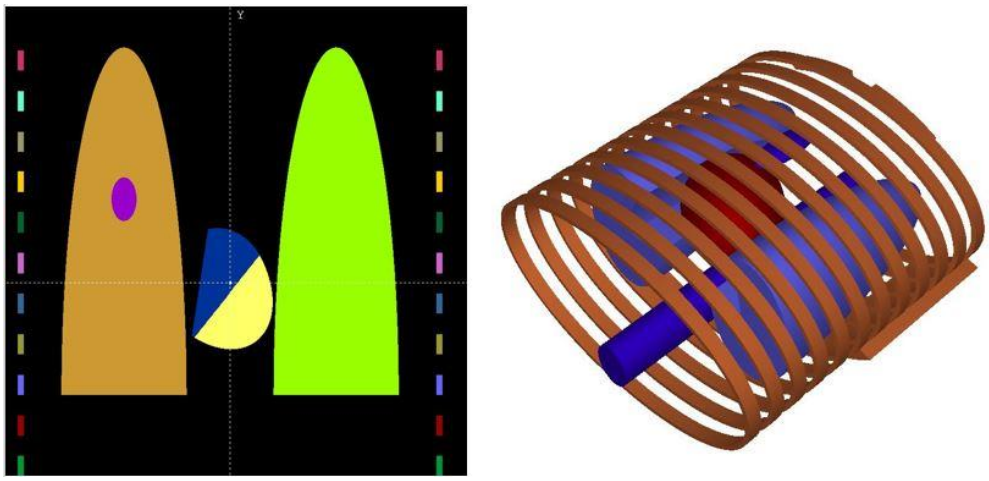
Figura 20 - Geometria da Cabeça em 2D e 3D



Fonte: Desenvolvido pelo próprio autor

Na figura 18 o tumor está representado pele elipse, na cor roxa, dentro do cérebro.

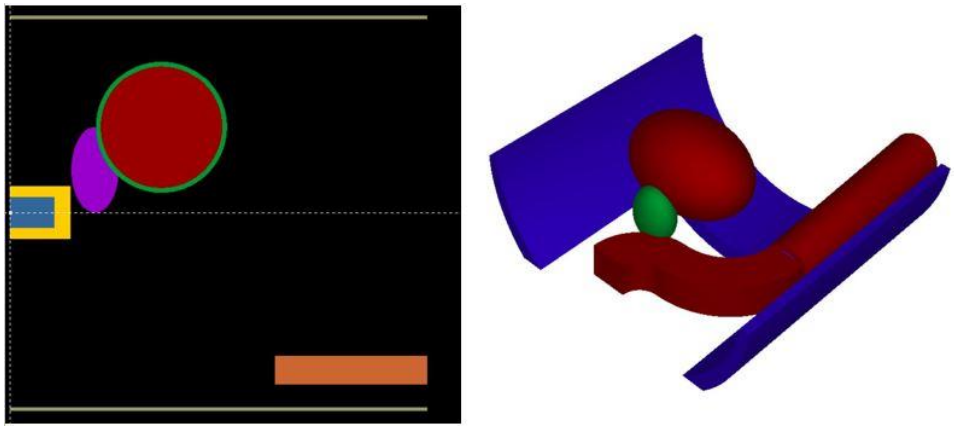
Figura 21 - Geometria do Pulmão em 2D e 3D



Fonte: Desenvolvido pelo próprio autor

Na figura 19 o tumor está representado pela elipse, na cor roxa, dentro do pulmão direito.

Figura 22 - Geometria da Próstata em 2D e 3D



Fonte: Desenvolvido pelo próprio autor

Na figura 20 podemos visualizar a próstata, representada pela elipse na cor roxa em 2D e na cor verde em 3D, abaixo da bexiga.

4.2.2 Arquivos de programas principais.

Os arquivos dos programas principais foram criados com parâmetros de entrada, parâmetros de controle e com os parâmetros para geração dos resultados que se espera obter com a execução das simulações das três geometrias, variando as sementes iniciais para geração de números pseudoaleatórios e o número de partículas simuladas.

A partícula primária utilizada na simulação foi o fóton com um espectro de energia do feixe de 6 MeV e variação de 0,250 MeV, conforme o acelerador linear da Varian encontrado no trabalho realizado por (SHEIKH-BAGHERI; ROGERS, 2002). A fonte de emissão foi colocada a 100 cm do isocentro do tecido alvo e um ângulo alpha com abertura de 2,86 graus gerou um feixe cônico/circular com raio de 5 centímetros. Os parâmetros de controle das partículas percorrendo os materiais foram estipulados com um EABS de 10^3 eV para elétrons, fótons e pósitrons; C1 e C2 com valores de 0,05 e WCC e WCR com valores de 10^3 eV. Esses valores são considerados conservadores para realização da simulação de modo a garantir a exatidão dos resultados. Para a geração do mapa de distribuição de dose, foi determinada uma resolução de 1.000.000 voxels nos eixos X, Y e Z com caixa de distribuição de dose com volumes 17x15x21 centímetros para cabeça, de 41x35x21 centímetros para pulmão e de 41x22x21 centímetros para próstata.

4.2.3 Mapa de Distribuição de Dose

Como resultado da simulação os programas principais geram um arquivo chamado 3d-dose-map.dat com informações da posição central e da dose depositada no voxel, o que possibilita plotar o mapa de distribuição de dose. O pacote PENELOPE-2014 traz um programa, escrito sob o código fonte do arquivo plotdose.f, que utiliza o arquivo 3d-dose-map.dat para gerar um novo arquivo chamado plotdose.dat com informações sobre qual material é formado o *voxel* que recebeu o depósito de dose.

De modo a possibilitar a plotagem de corpos específicos, como o tumor, a próstata e o pulmão, o programa plotdose.f foi alterado para obter como resultado em qual corpo o *voxel* está localizado. Isso foi possível utilizando a função LOCATE que retorna o corpo no qual as coordenadas X, Y Z estão situadas. De posse desse novo arquivo, foi criado um script na linguagem de programação do gnuplot, que possibilitou gerar um mapa de distribuição de dose em 3D de corpos específicos.

4.3 Desenvolvimento da versão C++ da simulação

O desenvolvimento de uma versão C++ do PENELOPE-2014 tomou como base os estudos realizados nas etapas 4.1 e 4.2 e é considerado um passo importante pois a migração para uma linguagem sequencial, que serve de base para a linguagem paralela CUDA, possibilitou uma implementação do código fonte de modo escalar, onde cada nova rotina implementada pode ser testada e validada com a execução híbrida do programa contendo partes do código original PENELOPE-2014 e partes da nova versão em C++. Para possibilitar essa

combinação e execução simultânea das duas versões foram necessárias algumas adaptações no código devidas particularidades de cada linguagem de programação.

4.3.1 Particularidades entre as linguagens de programação Fortran e C++

A linguagem de programação Fortran é bem difundida no ambiente acadêmico e científico enquanto o C++ se destaca também no ambiente empresarial. As duas linguagens possuem algumas particularidades importantes que devem ser levadas em consideração quando há necessidade de utilização de código híbrido ou realização de uma portabilidade.

Uma das diferenças diz respeito aos tipos de dados existentes entre as duas linguagens. O quadro 11 mostra os principais tipos compatíveis e que foram utilizados para armazenar dados com a mesma finalidade.

Quadro 11 - Principais tipos de dados compatíveis entre Fortran e C++

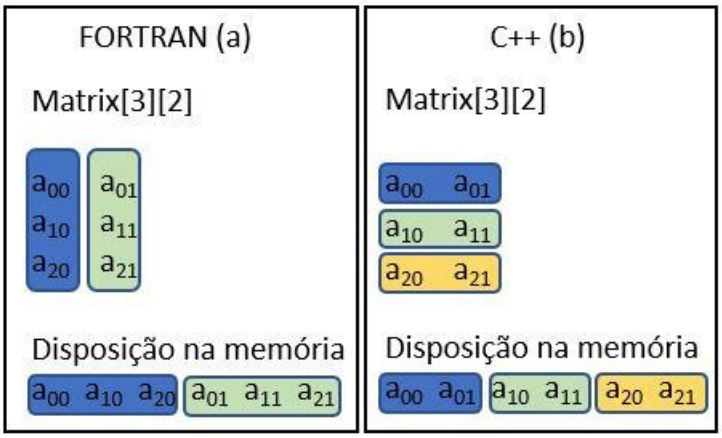
Fortran	**C++**
INTEGER	int
LOGICAL	bool
REAL	float
DOUBLE PRECISION	double

Fonte: (ARNHOLM, 2022).

Além dos tipos de dados também se verifica uma diferença na disposição dos dados em um *array*. O Fortran trata como primeiro elemento de um *array* o dado de posição 1 (*array*[1]), enquanto o C++ trata como o primeiro elemento de um *array* o dado de posição 0 (*array*[0]). Essa diferenciação foi levada em conta ao acessar os dados ou percorrer um *array* por meio de laços de repetição para que não seja acessado ou modificado um dado em posição equivocada.

A disposição dos dados de uma matriz na memória é uma outra diferença entre as duas linguagens. O Fortran dispõe a alocação de uma matriz 2D acomodando coluna na frente de coluna enquanto o C++ dispõe a alocação da mesma matriz acomodando linha na frente de linha. A figura 21 exemplifica essa particularidade.

Figura 23 - Disposição de uma matriz 2D na memória



Fonte: Desenvolvido pelo próprio autor com base em (ARNHOLM, 2022).

A disposição diferente dos dados de uma matriz na memória faz com que cuidados especiais precisem sem tomados ao alocá-las na memória RAM. Em suma, uma matriz 3x2 em Fortran precisa ser considera como uma matriz 2x3 em C++. Dessa forma, os dados de matrizes do PENELOPE-2014 foram mapeados por meio de ponteiros em matrizes C++ invertendo a sua dimensão para que o acesso e modificações fossem realizados de modo correto.

4.3.2 Combinação do código Fortran com código C++

O pacote PENELOPE-2014 possui dezenas de funções e sub-rotinas distribuídas em mais de trinta mil linhas de código fonte. Realizar a escrita de um novo programa totalmente em linguagem C++ dificultaria garantir a exatidão dos resultados, pois todo o código deveria ser implementado para realização dos devidos testes de validação. Dessa forma, foi adotada uma estratégia de execução de um programa híbrido, permitindo a combinação de partes do código fonte em Fortran e partes do código fonte em C++. Essa abordagem possibilitou a escrita progressiva das funções e a devida realização dos testes de comparação de resultados a cada novo trecho de código implementado.

Para que essa abordagem fosse possível, foram implementadas sub-rotinas dentro do código Fortran que mapearam toda a sua estrutura de dados por meio de ponteiros em um programa C++. Dessa forma, todos os dados foram apontados para o mesmo endereço na memória RAM nas diferentes linguagens e qualquer alteração efetuada em C++ também representaria uma alteração realizada em Fortran. Isso garantiu a integridade dos dados e possibilitou a execução individual das funções.

4.3.3 Funcionalidades da versão C++

A nova versão em C++ realiza a leitura do arquivo de geometria, do arquivo principal e dos arquivos de materiais com os mesmos padrões de layout da versão do PENELOPE-2014. Dessa forma, todas as sub-rotinas de leituras dos arquivos foram implementadas com suas respectivas estruturas de dados visando garantir essa compatibilidade. A priori, foram implementadas todas as sub-rotinas que possibilitam a realização de uma simulação que possua fótons como partícula primária, um espectro de energia, feixe cônico, e com simulações de todas as partículas secundárias geradas sejam elas, fótons, elétrons ou pósitrons garantindo o mapeamento de toda a história das partículas durante a simulação Monte Carlo.

Como resultado é gerado um arquivo no mesmo modelo da versão original e com todas as informações do arquivo penmain-res.dat além de um arquivo chamado plotdose.dat que possibilita a plotagem em 3D do mapa de distribuição de dose. A função que possibilita gerar o plotdose.dat segue as mesmas alterações realizadas na versão do PENELOPE-2014 que permitiram, por meio da utilização da função LOCATE, a localização do corpo de depósito de dose mediante a informação das coordenadas de posição.

4.4 Adaptação da versão C++ para versão PRISMATIC

A plataforma de programação CUDA tem como linguagem de programação nativa o C++. Portanto, possuir uma versão já implementada nessa linguagem facilitou a adaptação de um código com execução sequencial para um código com execução paralela. A versão final do código, que permite a simulação em GPUs, recebeu o nome de PRISMATIC, acrônimo de *PaRtIcle SiMulATIon in Cuda* (Simulação de Partículas em Cuda).

4.4.1 Código Sequencial e Código Paralelo.

Desenvolver um código paralelo executado em uma GPU não necessariamente significa o fim do código sequencial executado em uma CPU. A plataforma de programação CUDA possui uma característica heterogênea, onde parte do seu código é executado no *host* e outra parte executada no *device*. Respeitando essa característica elegemos alguns trechos do código para execução em ambiente sequencial e paralelo conforme abaixo:

a) Código Sequencial: Funções que realizam a leitura do arquivo de geometria, do arquivo de programa principal, dos arquivos de materiais, que geram e ordenam as partículas primárias que são enviadas para execução na GPU, que controlam o tempo de execução e que criam os arquivos que contêm os resultados simulação.

b) Código Paralelo: Funções que executam a simulação de partículas primárias, partículas secundárias e que contabilizam os resultados da simulação.

Ao se realizar essa separação dos códigos fonte a CPU ficou com um maior trecho de código a ser executado, porém a GPU ficou encarregada de executar os trechos de código de maior uso computacional o que deve conferir um ganho de desempenho devido à possibilidade de execução paralela.

4.4.2 Sementes para os geradores de números pseudoaleatórios.

Em algumas situações a computação paralela requer cuidados especiais para que os mesmos dados não sejam acessados e sobrescritos por *threads* diferentes. Este é o caso das sementes dos geradores de números pseudoaleatórios. O PENELOPE-2014 faz uso de um algoritmo chamado RANECU que utiliza como base duas sementes independentes para a geração de um número pseudoaleatório. Se utilizarmos as mesmas duas sementes no ambiente de computação paralela, corre-se o risco de gerar números pseudoaleatórios iguais prejudicando a essência da simulação Monte Carlo.

O PENELOPE-2014 traz em seu código um conjunto 1000 duplas de sementes para serem utilizadas em ambientes de simulação paralela. No entanto, as novas GPUs ultrapassam facilmente essa quantidade de núcleos tornando esse conjunto de sementes insuficiente para extrair o máximo de desempenho e impedir a repetição de números pseudoaleatórios. Diante disso, foi utilizado o programa seedsMLCG, desenvolvido por BADAL e SEMPAU em 2005, que é o mesmo programa utilizado pelo PENELOPE-2014 para gerar as suas 1000 duplas de sementes, para a geração de 10.000 duplas de sementes com capacidade de gerar até 10^{13} números pseudoaleatórios diferentes. Isso possibilitou que cada *thread* possua um gerador de número pseudoaleatório exclusivo e de alta qualidade.

4.4.3 Estratégias de desenvolvimento do código Paralelo

A programação paralela é um paradigma recente e que possui diferenças consideráveis quando comparada à programação sequencial que estamos habituados. Estratégias para aumentar a eficiência do código devem ser adotadas tomando-se cuidados especiais como a criação da estrutura dos dados, a utilização dos diferentes tipos de memórias, o tráfego de dados entre *host* e *device* e a alocação de núcleos disponíveis para execução de um *kernel*.

4.4.3.1 Divergência de Threads

Com base no modelo SIMT, utilizado por CUDA, quando um *thread* possui um caminho de execução diferente dos demais, ocorre uma perda de eficiência que é chamada de divergência de *thread*. Na simulação Monte Carlo, aplicada ao transporte de radiação, essa divergência pode ocorrer em várias situações devido à natureza da simulação que tem de lidar com diferentes tipos de partículas, com diferentes energias e sofrendo diferentes tipos de interações. Para minimizar essa problemática foram adotadas as seguintes estratégias:

a) Ordenação das partículas primárias por energia: A função responsável por gerar uma partícula primária faz uso de um número pseudoaleatório para determinar a sua energia inicial. Executar *warps* com partículas primárias possuindo energias com valores discrepantes acentua a incidência de divergência de thread devido à quantidade superior de interações que uma partícula de alta energia pode sofrer frente a uma partícula de baixa energia.

b) *Warps* homogêneos: Cada tipo de partícula na simulação sofre diferentes tipos de interações com a matéria. Adotar a execução de um kernel com apenas um tipo de partícula diminui a divergência de thread pois várias partículas tendem a sofrer o mesmo tipo de interação dentro de um *warp*. Dessa forma, foram criados kernels separados para execução das interações dos fótons, elétrons e pósitrons.

4.4.3.2 Tráfego de dados entre memórias da CPU e da GPU.

Para execução de uma aplicação paralela faz-se necessária a transferência de dados entre as memórias do *host* e do *device*, seja para enviar os dados para execução (do *host* para o *device*) ou para colher os resultados obtidos (do *device* para o *host*). A transferência de grandes volumes de dados entres as memórias da CPU e da GPU tende a ser um gargalo na execução de um *kernel*. De modo a reduzir esse tráfego, foi realizado envio para o *device* apenas das partículas primárias, já ordenadas por energia, e optado pelo armazenamento das partículas secundárias diretamente na memória do device. Assim, para traçar a história das partículas secundárias, que são a maior quantidade de partículas em uma simulação, bastou apenas realizar a chamada do *kernel* responsável por executar as interações dos fótons, elétrons ou pósitrons secundários.

4.4.3.3 Redução do uso de registradores.

O código fonte para execução das interações da partícula com a matéria é extenso e faz

uso de diversas funções matemáticas o que acaba gerando um *kernel* com um alto número de registradores. A GPU possui uma memória pequena para armazenamento dos registradores e o número de *threads* ativas depende diretamente da intensidade do seu uso. O código fonte responsável pela execução das interações da partícula com a matéria possuía inicialmente um total de 200 registradores. Um número considerado alto, pois, em uma GPU de capacidade 6.0, proporcionaria uma ocupação da ordem de 0.125. Nessa situação, se a GPU possuir 1000 núcleos disponíveis, apenas 125 estariam aptos a executar o *kernel*. Diante disso, três estratégias foram adotadas para reduzir o número de registradores utilizados:

a) Quebra da função SHOWER em funções menores: A função SHOWER inicia todo o processo de interações das partículas com a matéria realizando a chamada de diversas outras funções. Seu código foi quebrado em funções menores e realizado a chamada sequencial dessas funções diretamente do host;
b) Substituição da função matemática *pow*(): As funções matemáticas fazem uso de registradores internos para realização dos cálculos. Em especial, a função *pow*(), responsável por realizar o cálculo de um número elevado a uma potência, foi substituída pelo seu equivalente explícito;
c) Uso de memória constante: Variáveis declaradas no início de diversas funções foram realocadas para a memória constante.

4.4.3.4 Acesso Adjacente à Memória Global

O maior fluxo de dados ocorre na memória global da GPU. Nela são armazenadas as estruturas de dados que guardam a maior parte das informações necessárias para realização da simulação. Visando realizar um uso otimizado desse tipo de memória, com o objetivo de aproveitar toda a largura de banda do barramento a cada acesso realizado, as estruturas de dados foram reestruturadas para seguirem o padrão SOA. A figura 22 demostra uma estrutura de dados do código fonte que é responsável por armazenar o estado de uma partícula durante o processo de simulação.

Figura 24 - Estrutura de dados para estado das partículas

```
static const int pilhaPart = 3072

typedef struct {
    double E[pilhaPart],
           X[pilhaPart], Y[pilhaPart], Z[pilhaPart],
           U[pilhaPart], V[pilhaPart], W[pilhaPart],
           WGHT[pilhaPart],
    int KPAR[pilhaPart],
        IBODY[pilhaPart],
        MAT[pilhaPart],
        ILB[5][pilhaPart],
} TRACK_MOD;
```

Fonte: Desenvolvido pelo próprio autor

Na figura 22 visualizamos a respectiva estrutura de dados segue o modelo SOA onde temos uma estrutura com vários *arrays* do tamanho da variável constante pilhaPart que é múltiplo de 32 garantindo uma ocupação ideal dos *warps* para execução nos SMs.

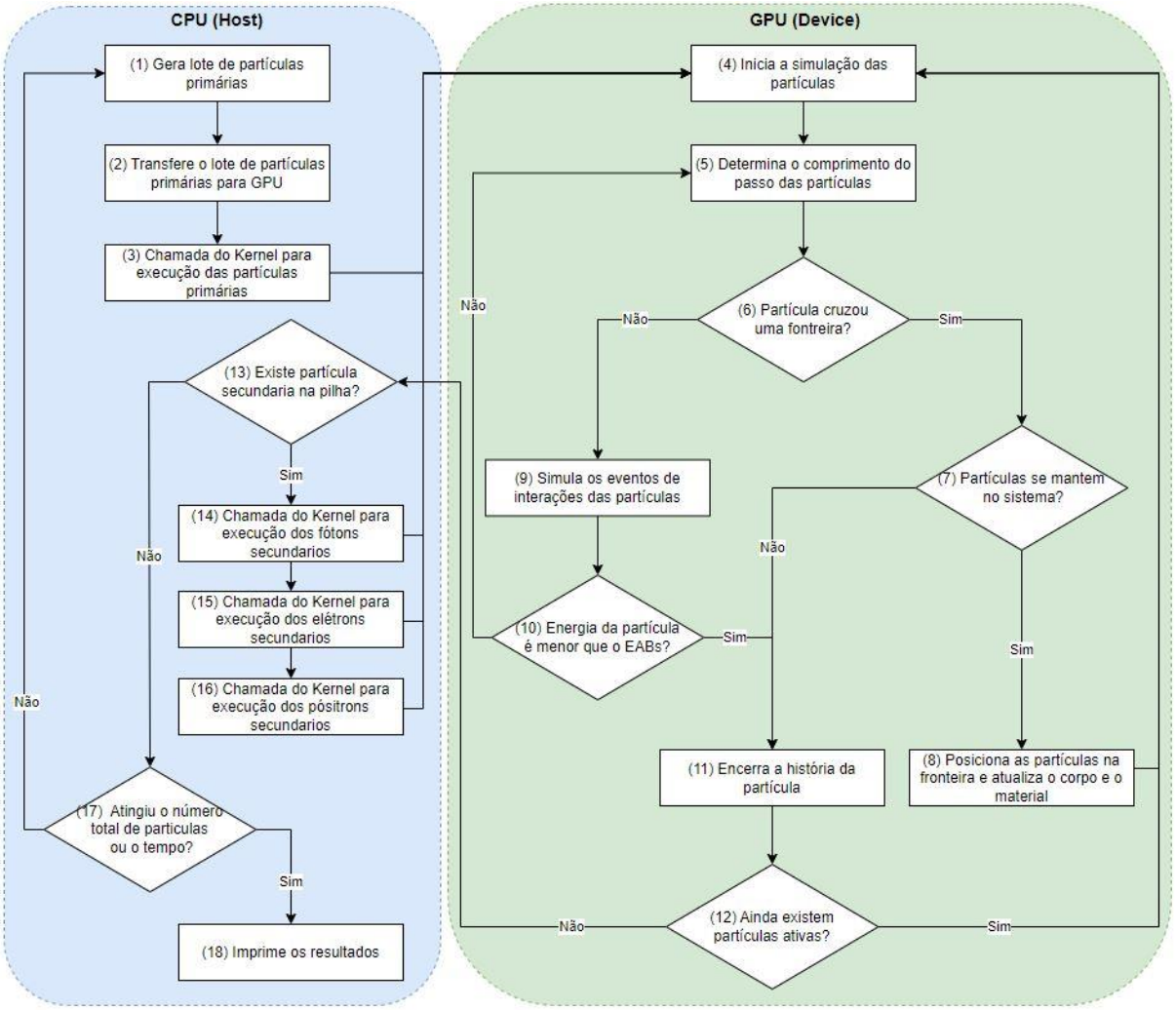
4.4.3.5 Uso de Memória Compartilhada

Existem dados que precisam ser acessados diversas vezes durante uma simulação. Este é o caso das sementes utilizadas para gerar um número pseudoaleatório. Na simulação Monte Carlo, aplicada ao transporte de radiação, um novo número pseudoaleatório é gerado para definir situações como o tipo de interação sofrida pela partícula, a perda de energia, o espaço percorrido na matéria e a sua mudança de direção. Portanto, durante a história de uma partícula essas sementes precisam ser acessadas e modificas inúmeras vezes e armazenar essa informação na memória compartilhada promove um ganho de desempenho em comparação com o armazenamento na memória global.

4.4.3.6 Fluxograma heterogêneo

Na figura 23 visualizamos o fluxograma que tratam das simulações das partículas primárias e secundárias no PRISMATIC.

Figura 25 - Fluxograma da simulação de partículas de modo heterogêneo entre *host* e *device*



Fonte: Desenvolvido pelo próprio autor

A figura 23 demostra a parte do processo de simulação que é executada no *host* e a parte, de computação intensiva, que é executada no *device*. Na etapa (1) do fluxograma ocorre a geração de um lote de partículas primárias, ordenadas por energia, onde é definido o seu estado inicial como valores de energia, coordenadas de posição, coordenadas de direção, o corpo e o material no qual está inserida. Na etapa (2) ocorre a transferência dessas partículas alocadas na memória do *host* para a memória do *device*. No passo (3) é realizada a chamada do kernel dimensionado de forma compatível com o tamanho do lote gerado para realização da simulação das partículas primárias. No passo (4) inicia-se a simulação das partículas com a chamada da função START. Em (5) é determinado o comprimento do passo que as partículas irão percorrer com a chamada da função JUMP. No passo (6) verifica-se quais partículas cruzaram uma fronteira de material. Para as partículas que cruzaram uma fronteira a etapa (7) verifica se as partículas ainda continuam no sistema de geometria delimitado. As partículas que

permanecerem no sistema de geometria a etapa (8) irá posicioná-las na borda da fronteira do novo material e o processo retorna ao passo (4) para iniciar novamente a simulação dessas partículas. Para as partículas que não cruzaram uma fronteira, a etapa (9) realiza os eventos de interação dessas partículas com a matéria por meio da chamada da função KNOCK. O passo (10) verifica quais partículas possuem uma energia maior do que a energia de absorção local do material. Em (11) ocorre o encerramento das histórias das partículas nas situações em que elas escapam do sistema de geometria ou no caso em que suas energias são menores do que a energia de absorção determinada para aquele material. Na etapa (12) é verificado se ainda existe alguma partícula ativa, caso exista, o processo retorna ao passo (4) para continuar a simulação das partículas restantes, do contrário o kernel é encerrado e processo retorna para a CPU. No passo (13) é verificado se existem partículas secundárias nas pilhas e, caso existam, é realizada a chamada das etapas (14), (15) e (16) que realizam a simulação das partículas secundárias dos fótons, elétrons e pósitrons, respectivamente. Após o término da execução da simulação de um lote de partículas primárias e de suas partículas secundárias a etapa (17) verifica se o fluxo de execução deve retornar para a criação de um novo lote de partículas, na etapa (1), caso não tenha atingido o número de partículas total ou o tempo determinado para a simulação, ou realizar a impressão os resultados, na etapa (18). Detalhes dos códigos fontes de preparação da simulação e da simulação das partículas primárias podem ser consultados nos Apêndices A e B

4.4.3.7 Processo de aperfeiçoamento constante

O processo de desenvolvimento de um código paralelo se beneficia da abordagem cíclica Avaliar, Paralelizar, Otimizar e Implantar. Seguindo a abordagem, durante o estudo do código fonte em Fortran e C++ foram feitas avaliações dos trechos de códigos elegíveis para realização da paralelização. Com a devida identificação desses códigos, foram realizadas adaptações para que pudessem ser executados de modo paralelo na GPU. Com a aplicação funcionando de modo paralelizada foram estudadas e implementadas otimizações visando melhorar o desempenho da simulação. Com isso, chegamos a uma versão que julgamos apta a publicação, porém passível de ser melhorada devido ao constante aperfeiçoamento que implementações em código paralelo na GPU proporcionam.

4.5 Simulação, validação e comparação de desempenho

As simulações no PENELOPE-2014, na versão em linguagem C++ e no PRISMATIC foram realizadas utilizando-se dos mesmos arquivos de geometria, de programas principais e de materiais. Para cada uma das três geometrias foram realizadas simulações para 10^4, 10^5, 10^6

e 10^7 partículas com três diferentes pares de sementes geradoras de números pseudoaleatórios, para cada ordem de grandeza, totalizando 36 simulações para cada versão estudada.

O objetivo da simulação na linguagem em C++ é validar o código fonte desenvolvido comparando o comportamento da simulação, os resultados de depósito de dose no tecido alvo e o mapa de distribuição de dose com os equivalentes obtidos na simulação realizada no PENELOPE-2014. Essa validação nos dá uma tranquilidade para avaliar os desempenhos das simulações, em termos de tempo de execução, realizadas no PRISMATIC frente às simulações realizadas no PENELOPE-2014.

As execuções das simulações do PENELOEPE-2014 e da versão em linguagem C++ foram realizadas no hardware composto por uma CPU Intel Xeon 6130 Gold 2.1 GHz (32 núcleos), 512 GB de memória RAM (*Random Acess Memory*) DDR4 2400 MHz e dois SSDs (*Solid State Drive*) de 240 GB.

A execução da simulação do PRISMATIC foi realizada em dois ambientes distintos. O primeiro ambiente é composto pelo mesmo hardware utilizado na simulação do PENELOPE-2014 e da versão em linguagem C++ acrescentado da utilização de uma GPU NVidia Quadro Pascal P5000 com 16 GB de memória que possui 2560 núcleos CUDA. O segundo ambiente é composto de uma CPU Intel i7 7500u 2.7 GHz, 16 GB de memória RAM DDR4 2100 MHz, um SSD 240 GB e uma GPU Nvidia 920MX 2 GB de memória.

5 RESULTADOS E DISCUSSÃO

Os resultados e a discussão são apresentados em cinco tópicos. Nos três primeiros tópicos são apresentados os resultados de depósito de dose, o mapa de distribuição de dose e realizada uma discussão sobre o comportamento da simulação e validação dos resultados. O quarto tópico realiza uma comparação dos resultados obtidos nas três simulações e na quinta e última parte são realizadas a apresentação e comparação dos tempos de simulação.

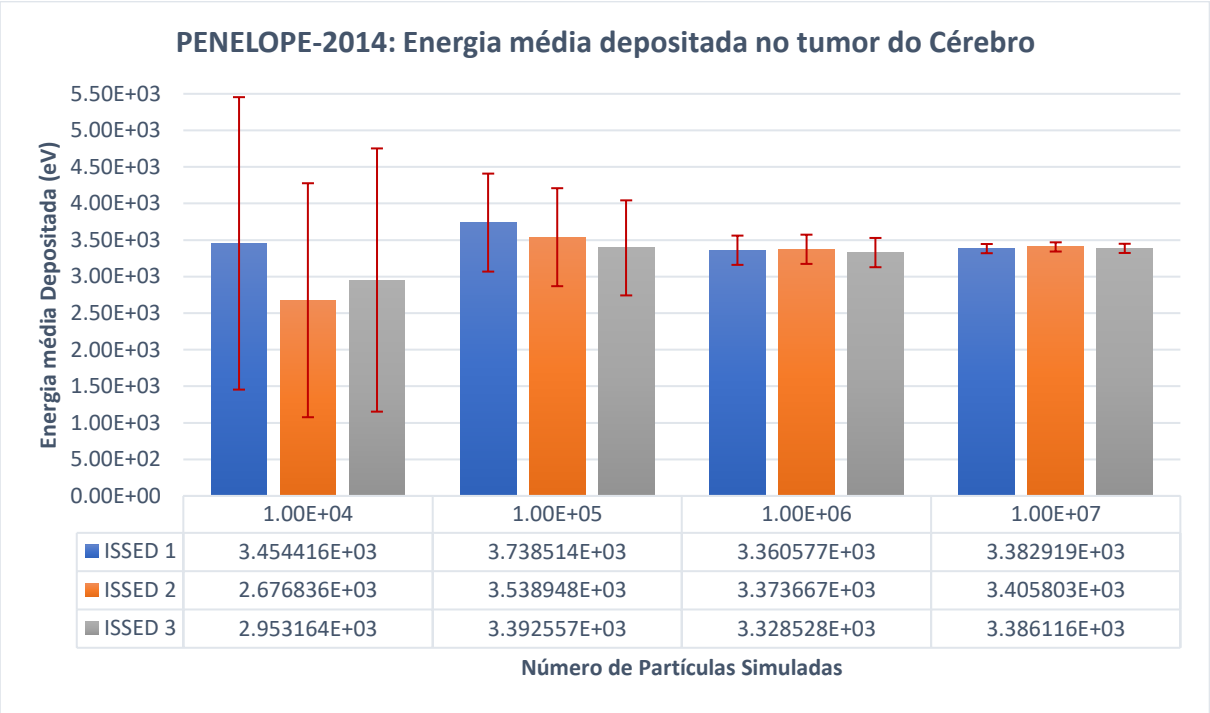
5.1 Resultados da simulação no PENELOPE-2014

Neste tópico são apresentados e discutidos os resultados dos depósitos de dose e mapas de distribuição de dose obtidos pelas simulações executadas no PENELOPE-2014.

5.1.1 Depósito de dose

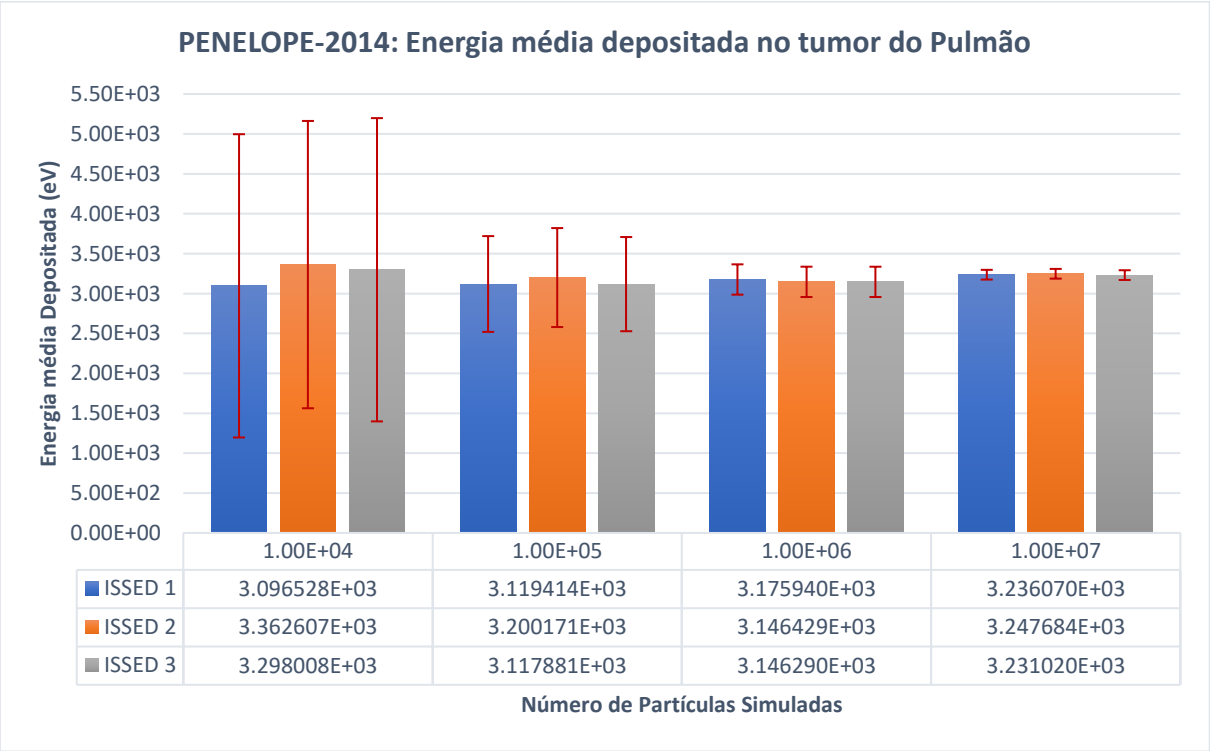
Nos gráficos 1, 2 e 3 é possível visualizar os depósitos de doses nos tumores do cérebro, pulmão e próstata obtidos pela simulação no PENELOPE-2014. Foram executadas três simulações com três duplas diferentes de sementes iniciais para analisar o comportamento das simulações. Verificamos em todos os gráficos uma diferença no depósito de dose quando se realiza uma simulação com um número pequeno de partículas e que os valores tendem a convergir, diminuindo o erro estatístico, à medida que o número de partículas aumenta, evidenciando o comportamento padrão de uma simulação Monte Carlo.

Gráfico 1 - Energia média depositada no tumor do Cérebro – Simulação PENELOPE-2014



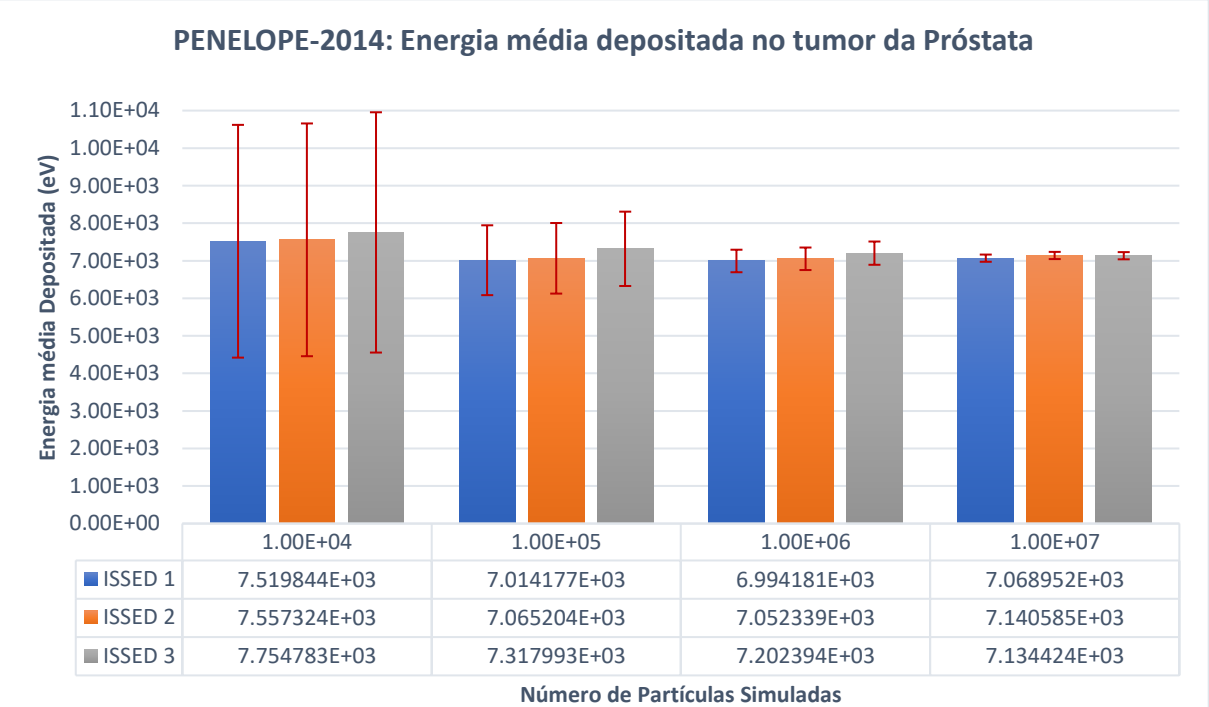
Fonte: Desenvolvido pelo próprio autor

Gráfico 2 - Energia média depositada no tumor do Pulmão – Simulação PENELOPE-2014



Fonte: Desenvolvido pelo próprio autor

Gráfico 3 - Energia média depositada no tumor da Próstata – Simulação PENELOPE-2014

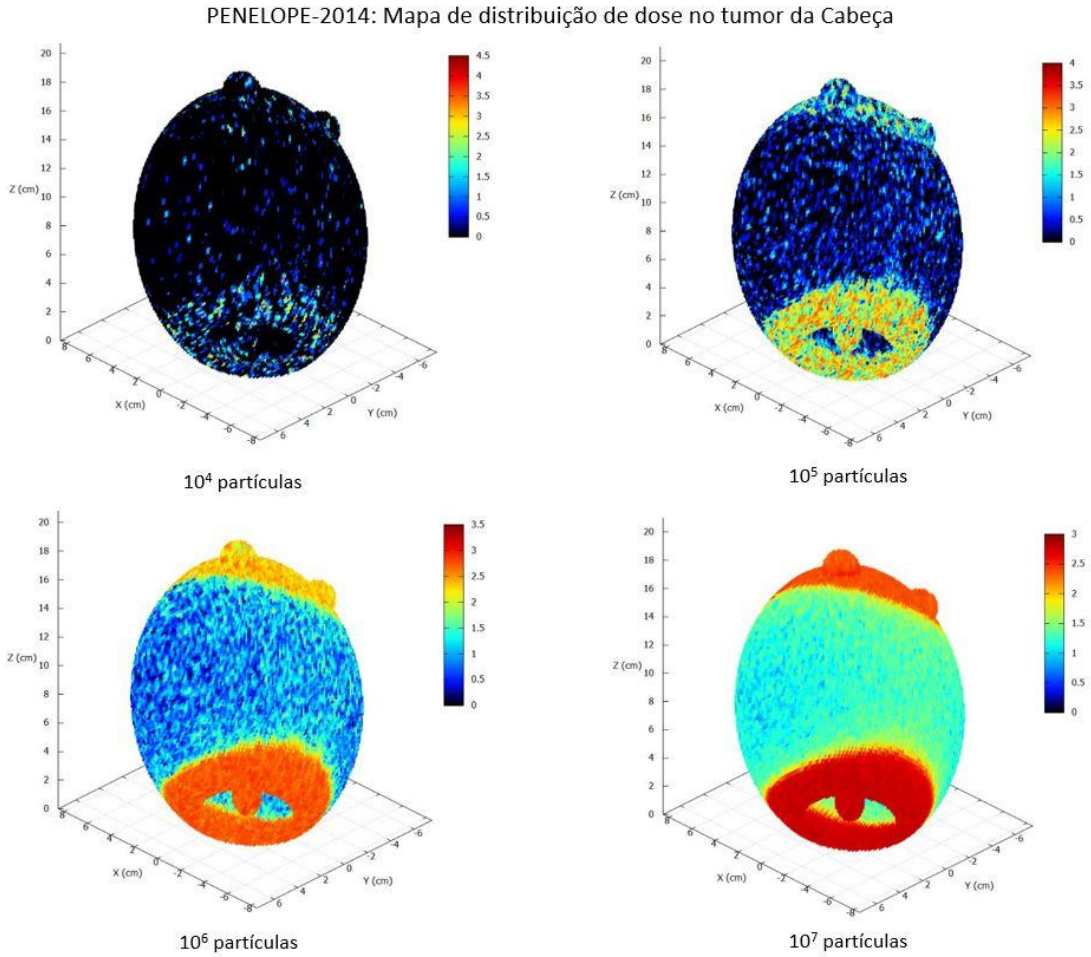


Fonte: Desenvolvido pelo próprio autor

5.1.2 Mapa de distribuição de dose

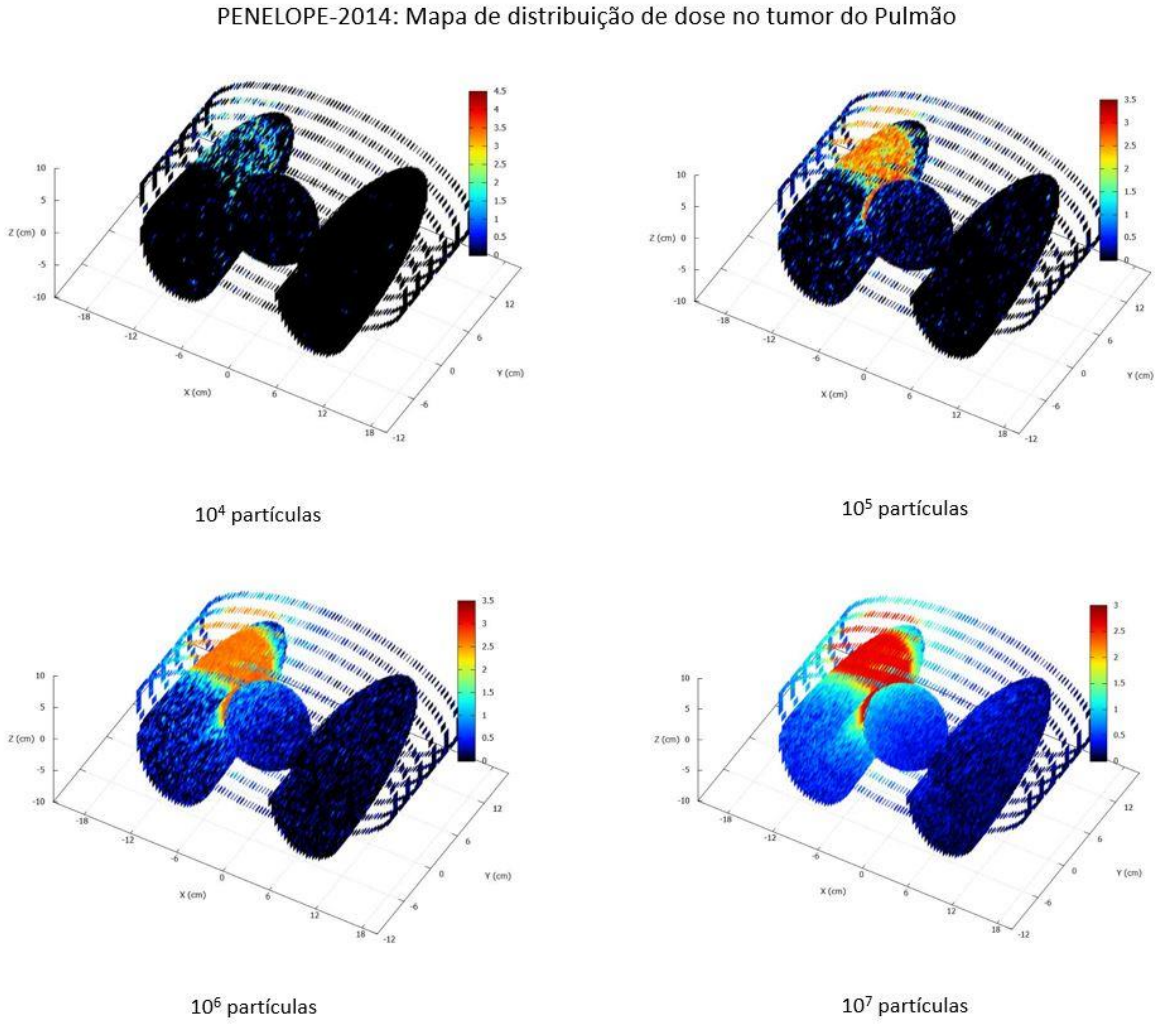
As figuras 24, 25 e 26 mostram os mapas de distribuições de doses nas geometrias da cabeça, do pulmão e da próstata obtidos pela simulação no PENELOPE-2014. É possível notar uma melhora na visualização da dose depositada à medida que se aumenta o número de partículas simuladas.

Figura 26 - Mapa de distribuição de dose na Cabeça: Simulação PENELOPE-2014



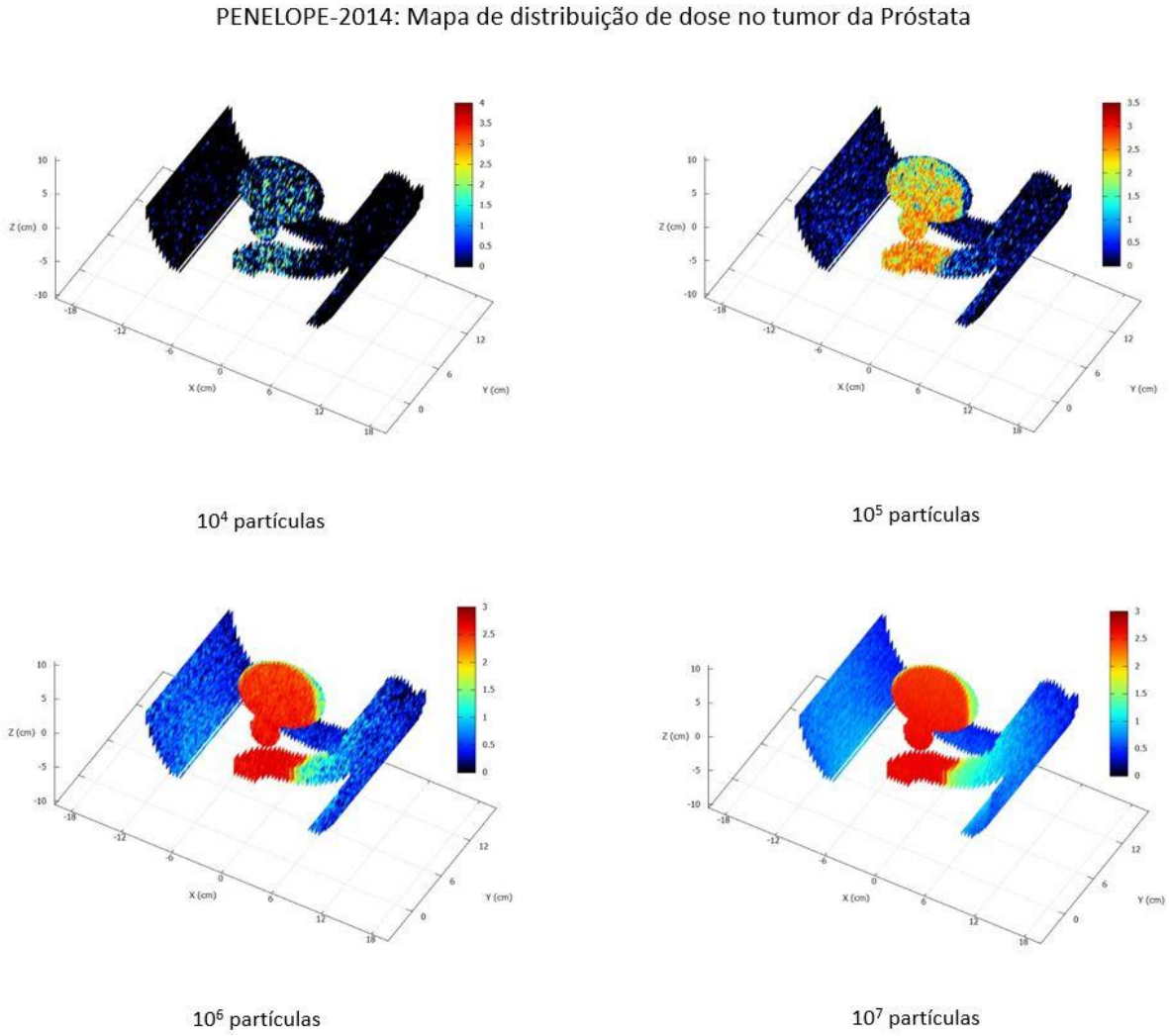
Fonte: Desenvolvido pelo próprio autor

Figura 27 - Mapa de distribuição de dose no Pulmão: Simulação PENELOPE-2014



Fonte: Desenvolvido pelo próprio autor

Figura 28 - Mapa de distribuição de dose na Próstata: Simulação PENELOPE-2014



Fonte: Desenvolvido pelo próprio autor

Podemos notar nas figuras 24, 25 e 26 os locais de incidência do feixe cônico/circular de 5 cm que resulta em uma área da geometria representada pela intensidade da dose que se destaca, em cor vermelha, ao aumentar o número de partículas simuladas. Na figura 25, a região de trás da cabeça recebe uma dose maior de radiação atingindo o tumor do cérebro. Na figura 26 o pulmão direito recebe a dosagem maior de radiação, porém as costelas também são atingidas. Na figura 27 a próstata recebe a dose maior de radiação com uma parte da bexiga e do reto também recebendo doses menores de radiação.

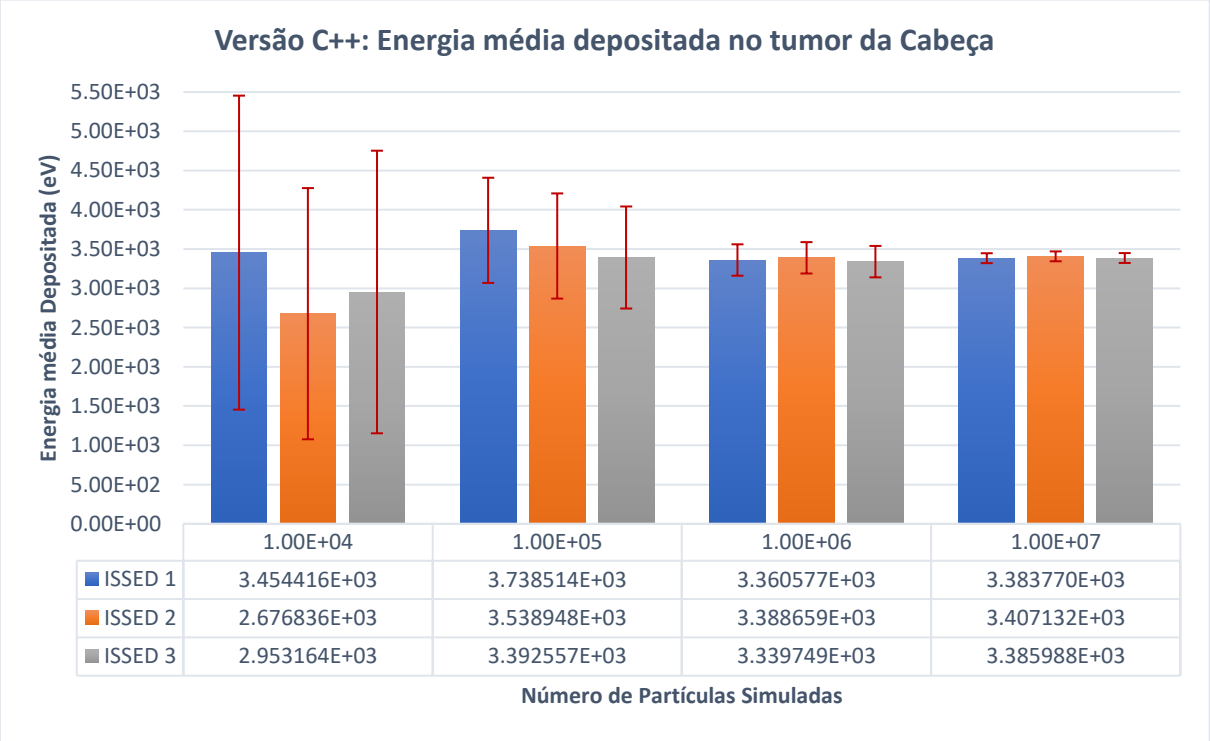
5.2 Resultados da simulação na versão C++

Neste tópico são apresentados e discutidos os resultados dos depósitos de dose e mapas de distribuição de dose obtidos pelas simulações executadas na versão C++.

5.2.1 Depósito de dose

Nos gráficos 4, 5 e 6 é possível visualizar os depósitos de doses nos tumores do cérebro, pulmão e próstata obtidos pela simulação da versão C++. Foram executadas três simulações com as mesmas três duplas de sementes iniciais utilizadas na simulação no PENELOPE-2014. Verificamos em todos os gráficos o mesmo comportamento da simulação no PENELOPE-2014 onde nota-se uma diferença no depósito de dose quando se realiza uma simulação com um número pequeno de partículas e que os valores tendem a convergir, diminuindo o erro estatístico, à medida que o número de partículas aumenta. Esse resultado se mostra satisfatório pois a versão C++ apresenta o comportamento esperado para uma simulação Monte Carlo e similar ao obtido pelo PENELOPE-2014.

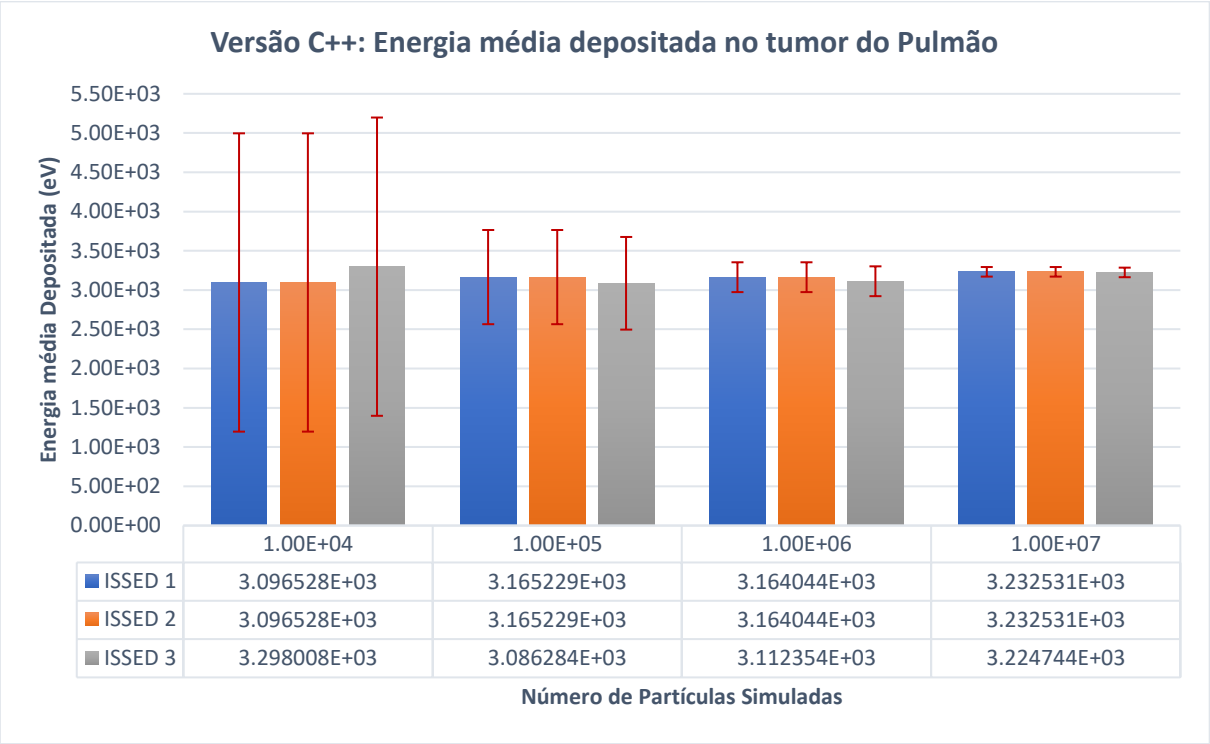
Gráfico 4 - Energia média depositada no tumor do Cérebro: Simulação Versão C++



	1.00E+04	1.00E+05	1.00E+06	1.00E+07
ISSED 1	3.454416E+03	3.738514E+03	3.360577E+03	3.383770E+03
ISSED 2	2.676836E+03	3.538948E+03	3.388659E+03	3.407132E+03
ISSED 3	2.953164E+03	3.392557E+03	3.339749E+03	3.385988E+03

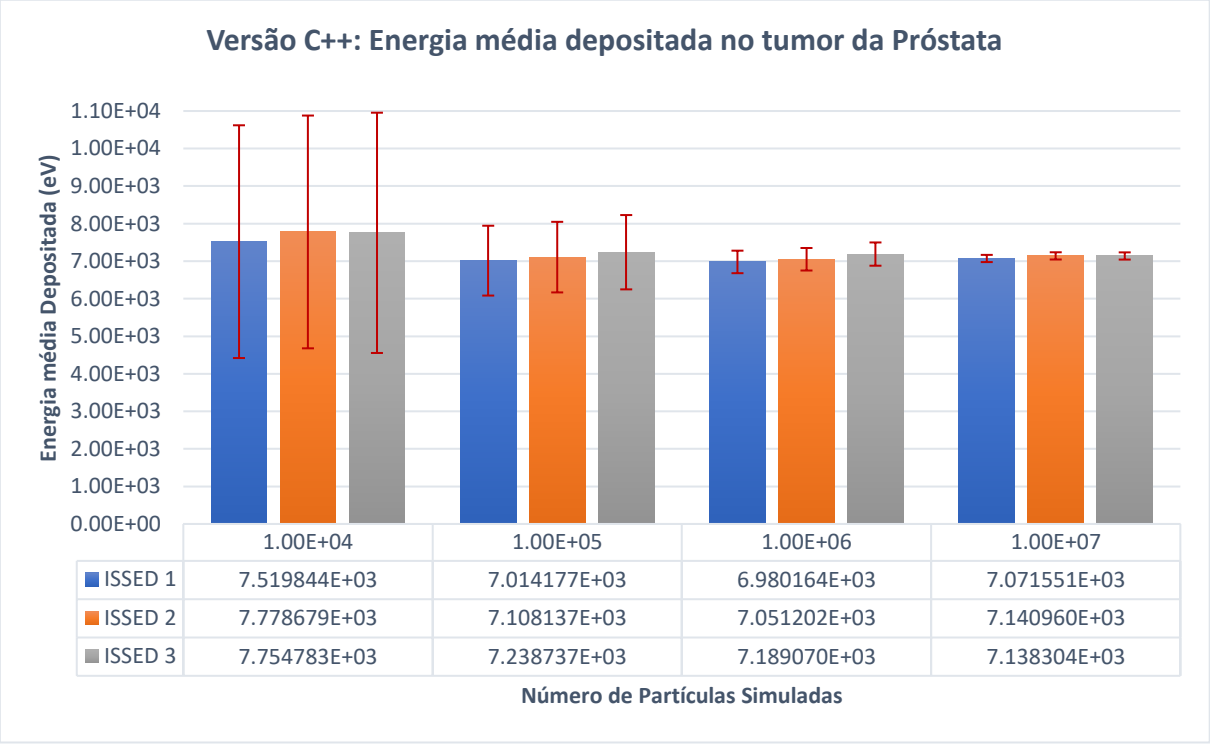
Fonte: Desenvolvido pelo próprio autor

Gráfico 5 - Energia média depositada no tumor do Pulmão: Simulação Versão C++



Fonte: Desenvolvido pelo próprio autor

Gráfico 6 - Energia média depositada no tumor da Próstata: Simulação Versão C++

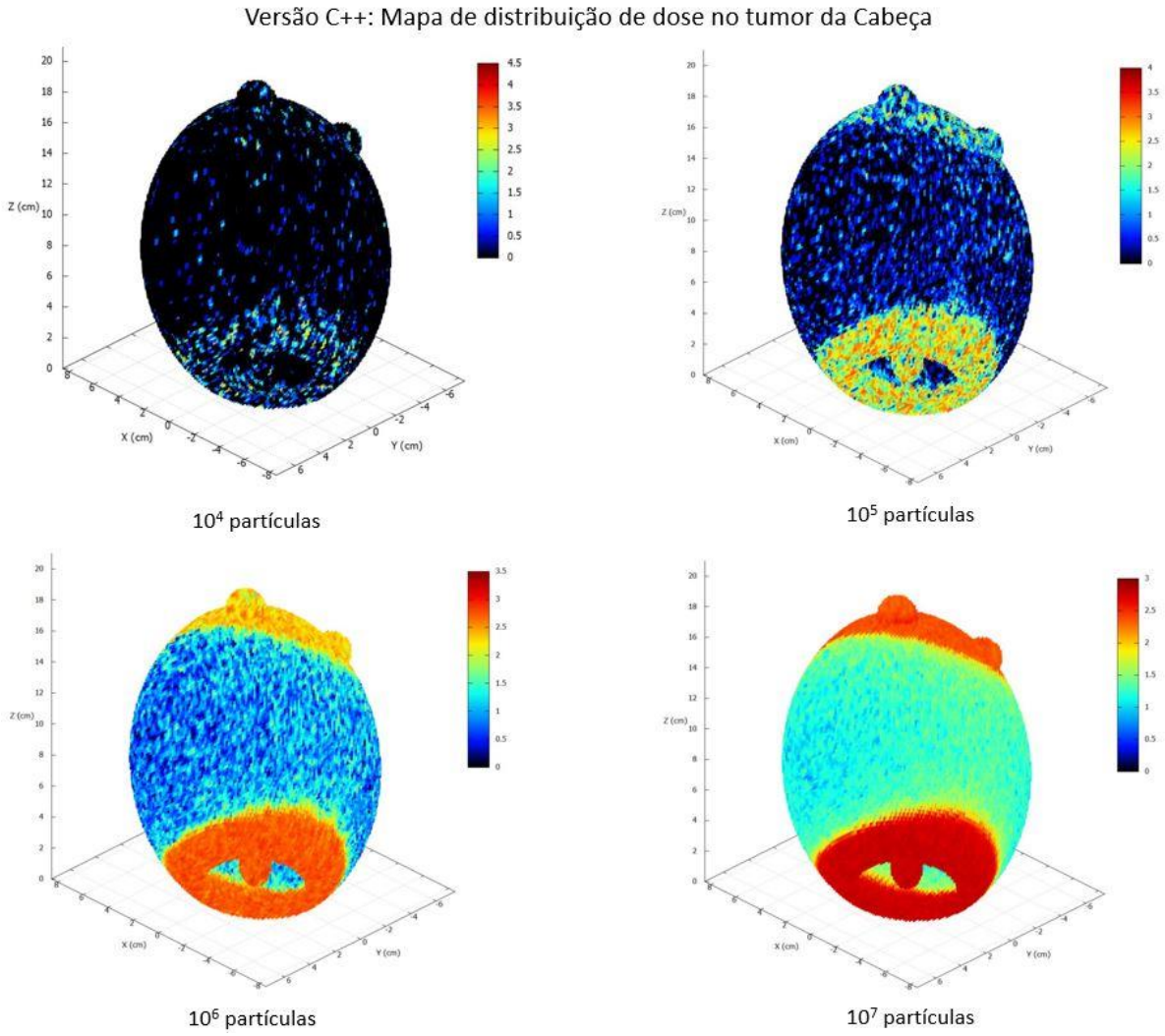


Fonte: Desenvolvido pelo próprio autor

5.2.2 Mapa de distribuição de dose

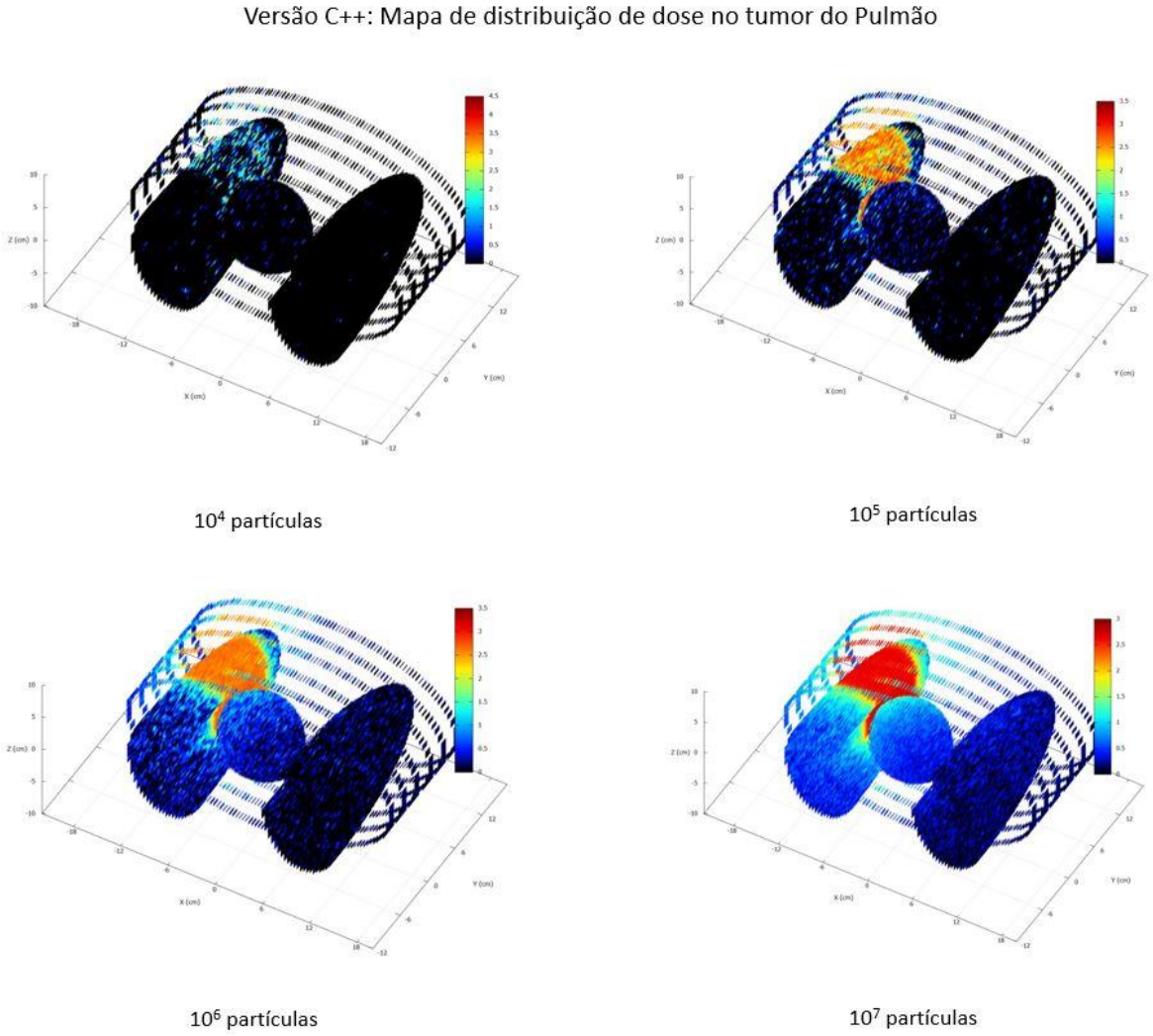
As figuras 27, 28 e 29 mostram os mapas de distribuições de doses nas geometrias da cabeça, do pulmão e da próstata obtidas pela simulação na versão C++. Assim como na versão PENELOPE-2014 é possível notar uma melhora na visualização da dose depositada à medida que se aumenta o número de partículas simuladas.

Figura 29 - Mapa de distribuição de dose na Cabeça: Simulação C++



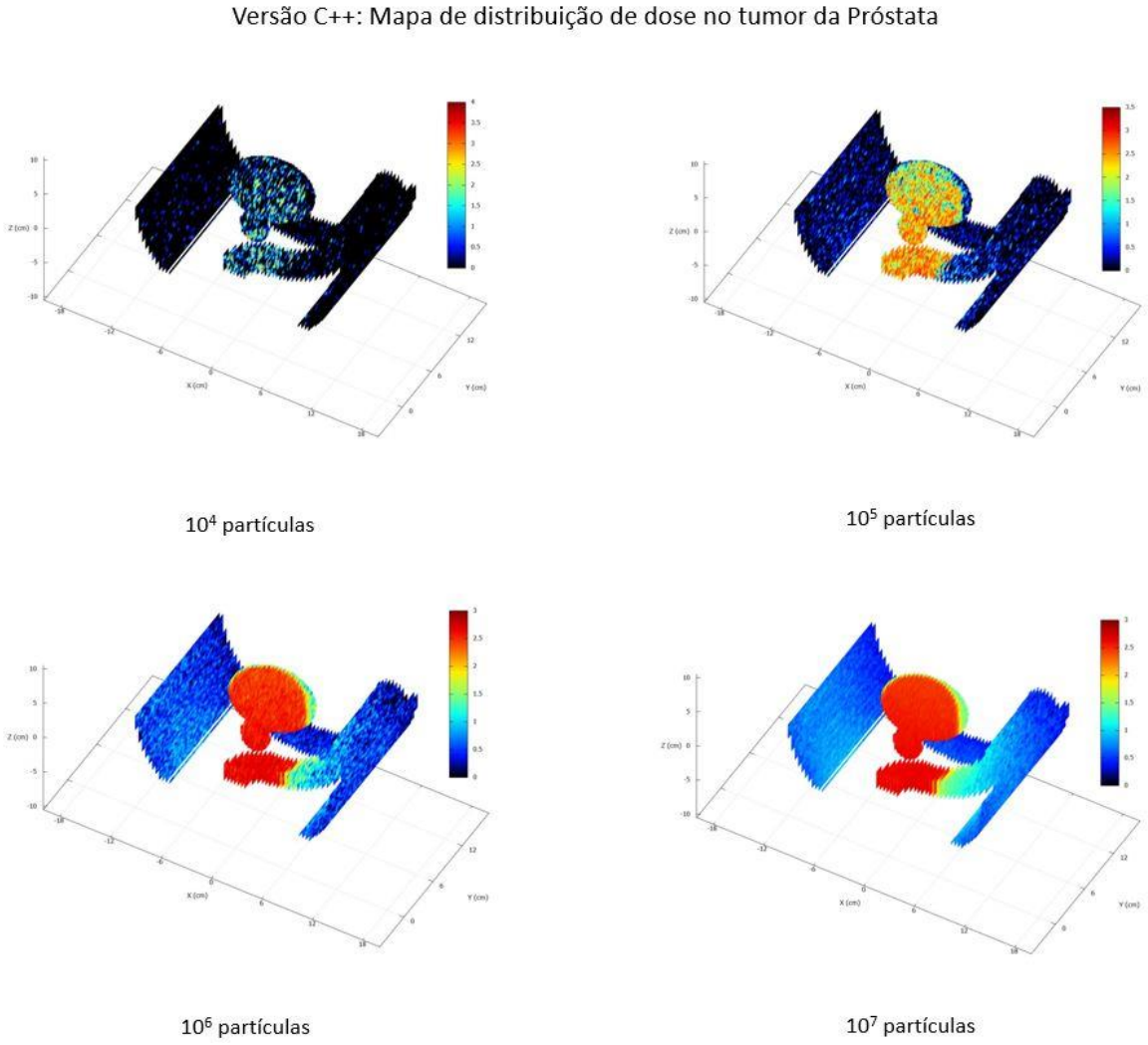
Fonte: Desenvolvido pelo próprio autor

Figura 30 - Mapa de distribuição de dose no Pulmão: Simulação C++



Fonte: Desenvolvido pelo próprio autor

Figura 31 - Mapa de distribuição de dose na Próstata: Simulação C++



Fonte: Desenvolvido pelo próprio autor

5.3 Resultados da simulação no PRISMATIC

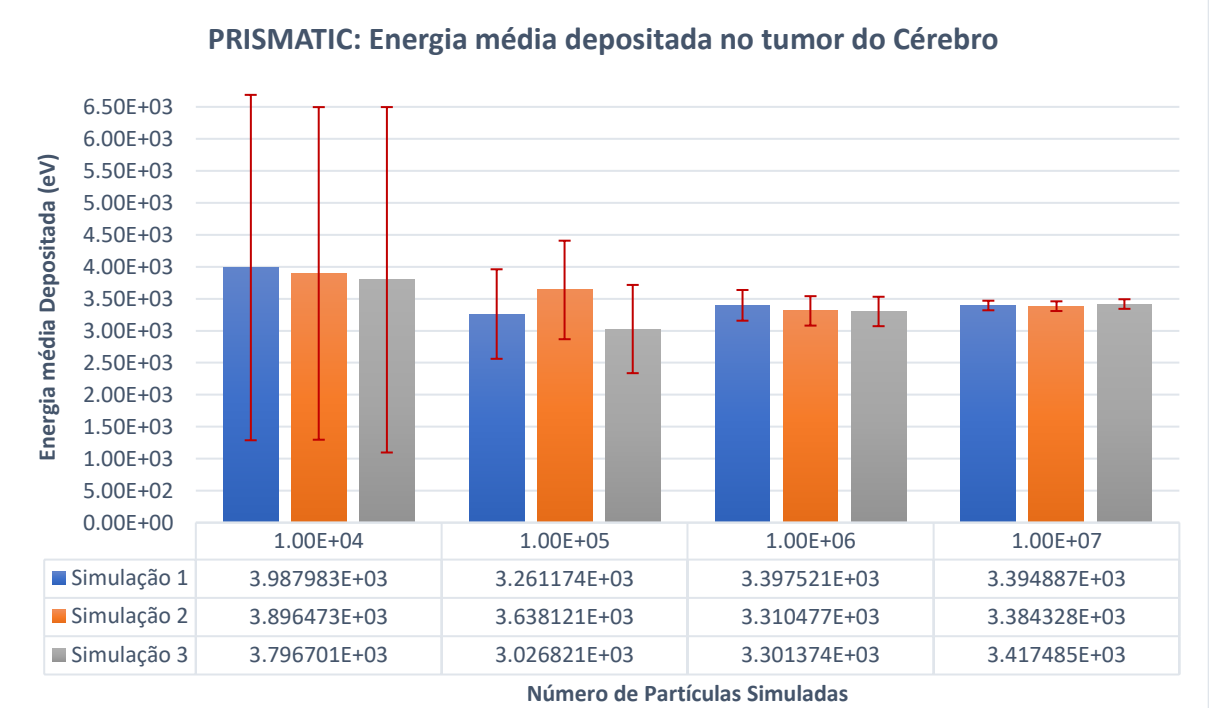
Neste tópico são apresentados e discutidos os resultados dos depósitos de dose e mapas de distribuição de dose obtidos pelas simulações executadas no PRISMATIC.

5.3.1 Depósito de dose

Nos gráficos 7, 8 e 9 é possível visualizar os depósitos de doses nos tumores do cérebro, pulmão e próstata obtidos pela simulação PRISMATIC. Foram executadas três simulações que promovem diferenças de resultados, quando comparados com as simulações executas no PENELOPE-2014 e na versão C++, devido a cada núcleo da GPU possuir um gerador de número pseudoaleatório diferente. Verificamos em todos os gráficos o mesmo comportamento das simulações no PENELOPE-2014 e na versão C++ onde nota-se uma diferença no de

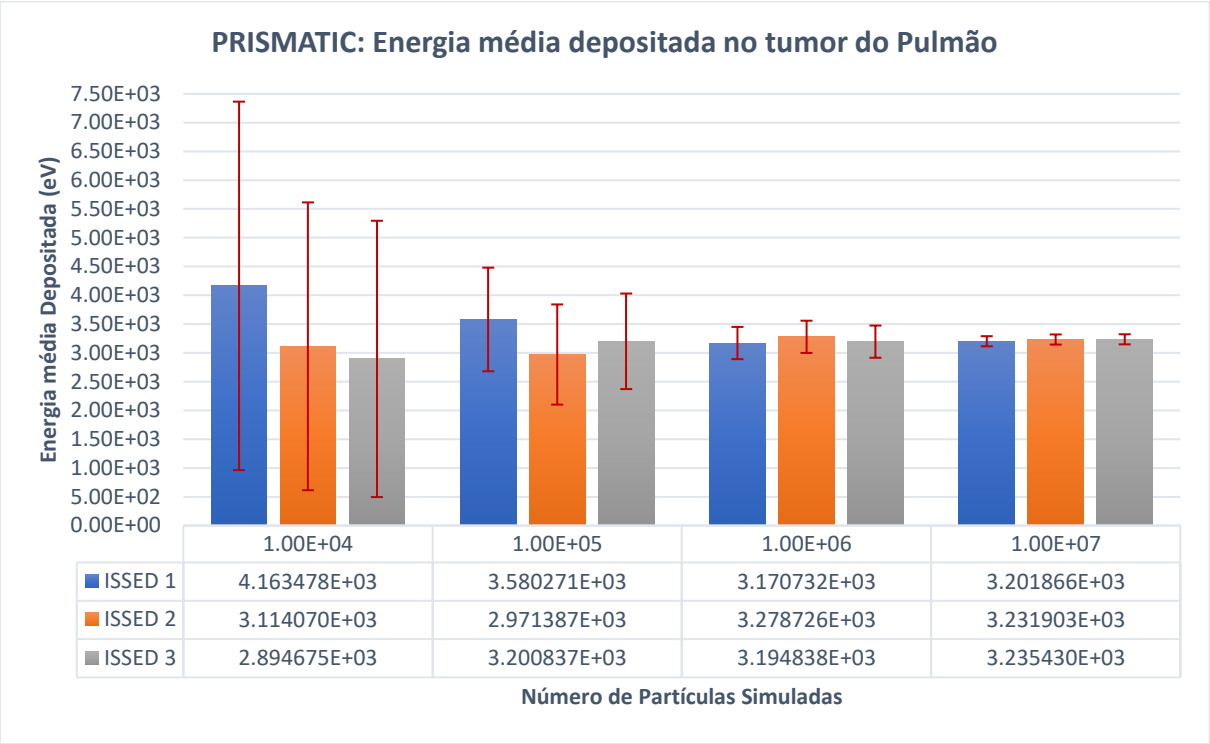
depósito de dose quando se realiza uma simulação com um número pequeno de partículas e que os valores tendem a convergir, diminuindo a incerteza estatística, à medida que o número de partículas aumenta. Assim, como na versão C++, esse resultado se mostra satisfatório pois evidência o comportamento esperado para uma simulação Monte Carlo.

Gráfico 7 - Energia média depositada no tumor da Cérebro: Simulação PRISMATIC



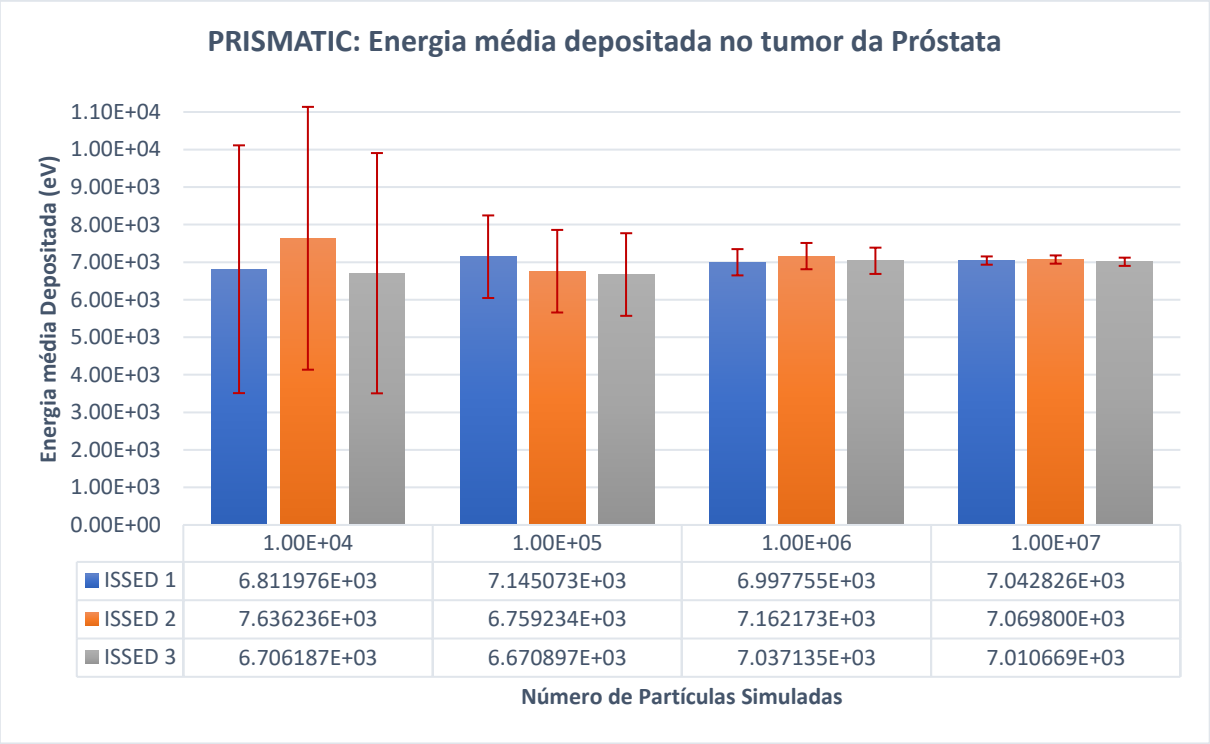
Fonte: Desenvolvido pelo próprio autor

Gráfico 8 - Energia média depositada no tumor do Pulmão: Simulação PRISMATIC



Fonte: Desenvolvido pelo próprio autor

Gráfico 9 - Energia média depositada no tumor da Próstata: Simulação PRISMATIC

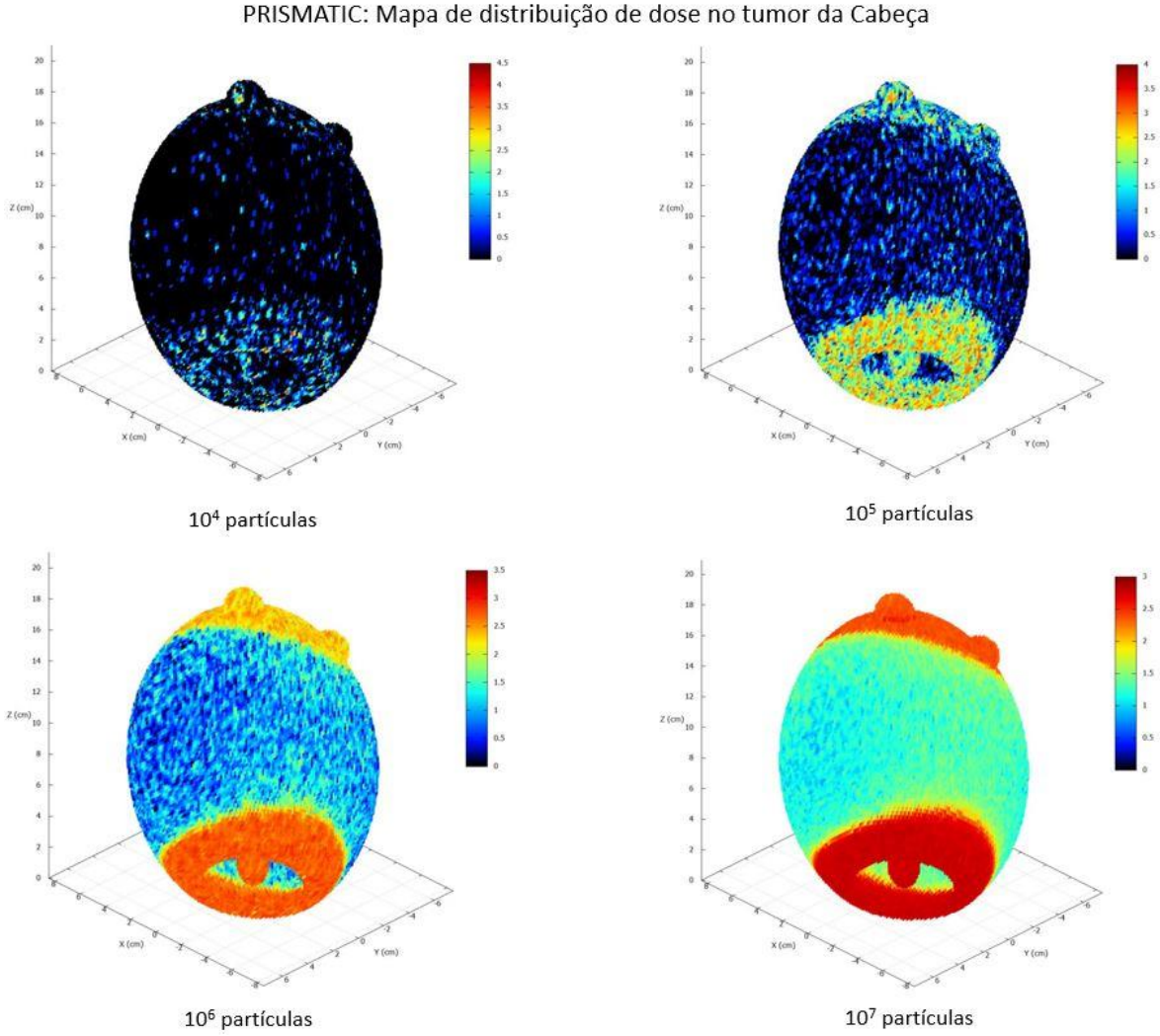


Fonte: Desenvolvido pelo próprio autor

5.3.2 Mapa de distribuição de dose

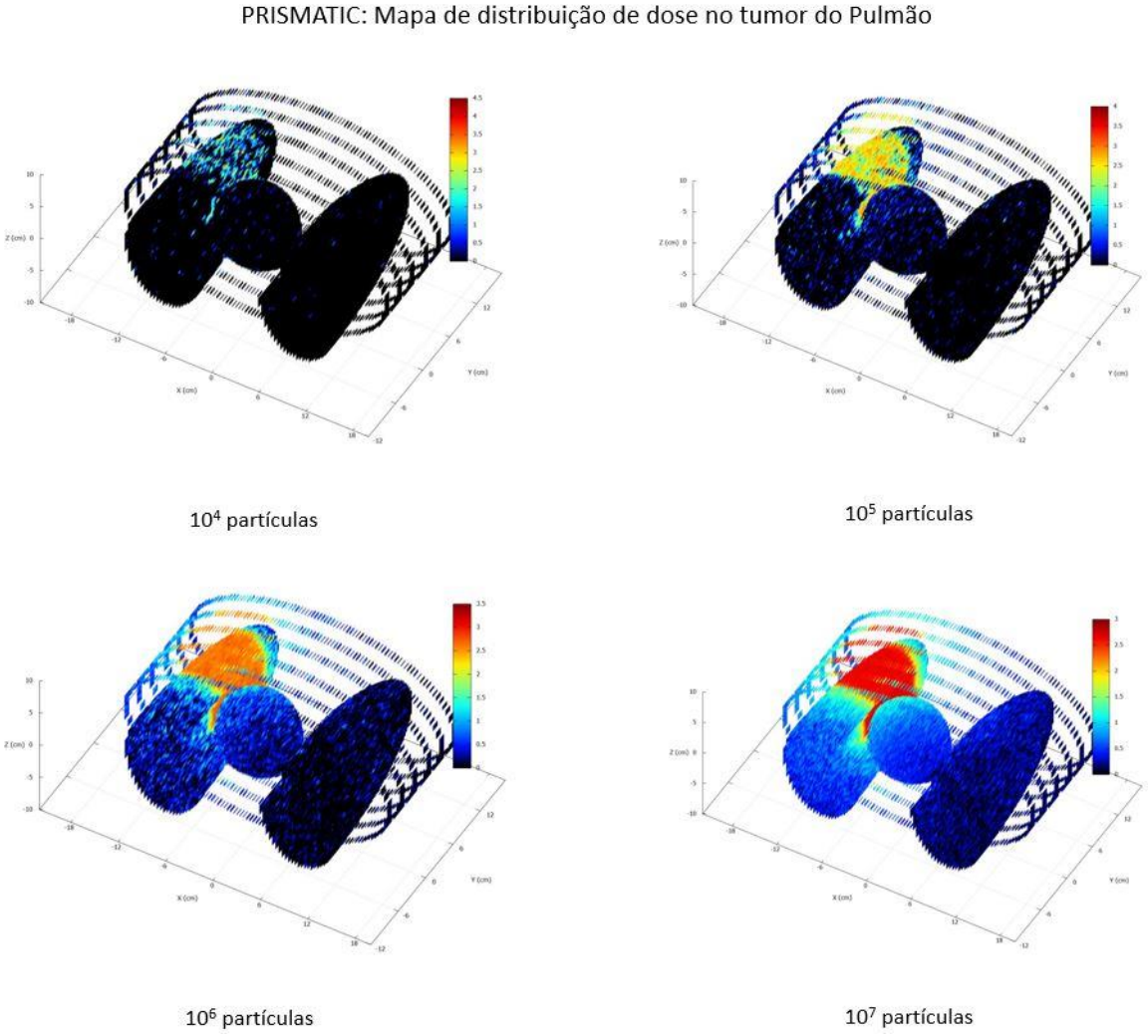
As figuras 30, 31 e 32 mostram os mapas de distribuições de doses nas geometrias da cabeça, pulmão e próstata obtidas pela simulação PRISMATIC. Assim como nas simulações no PENELOPE-2014 e na versão C++ é possível notar uma melhora na visualização da dose depositada à medida que se aumenta o número de partículas simuladas.

Figura 32 - Mapa de distribuição de dose na cabeça: Simulação PRISMATIC



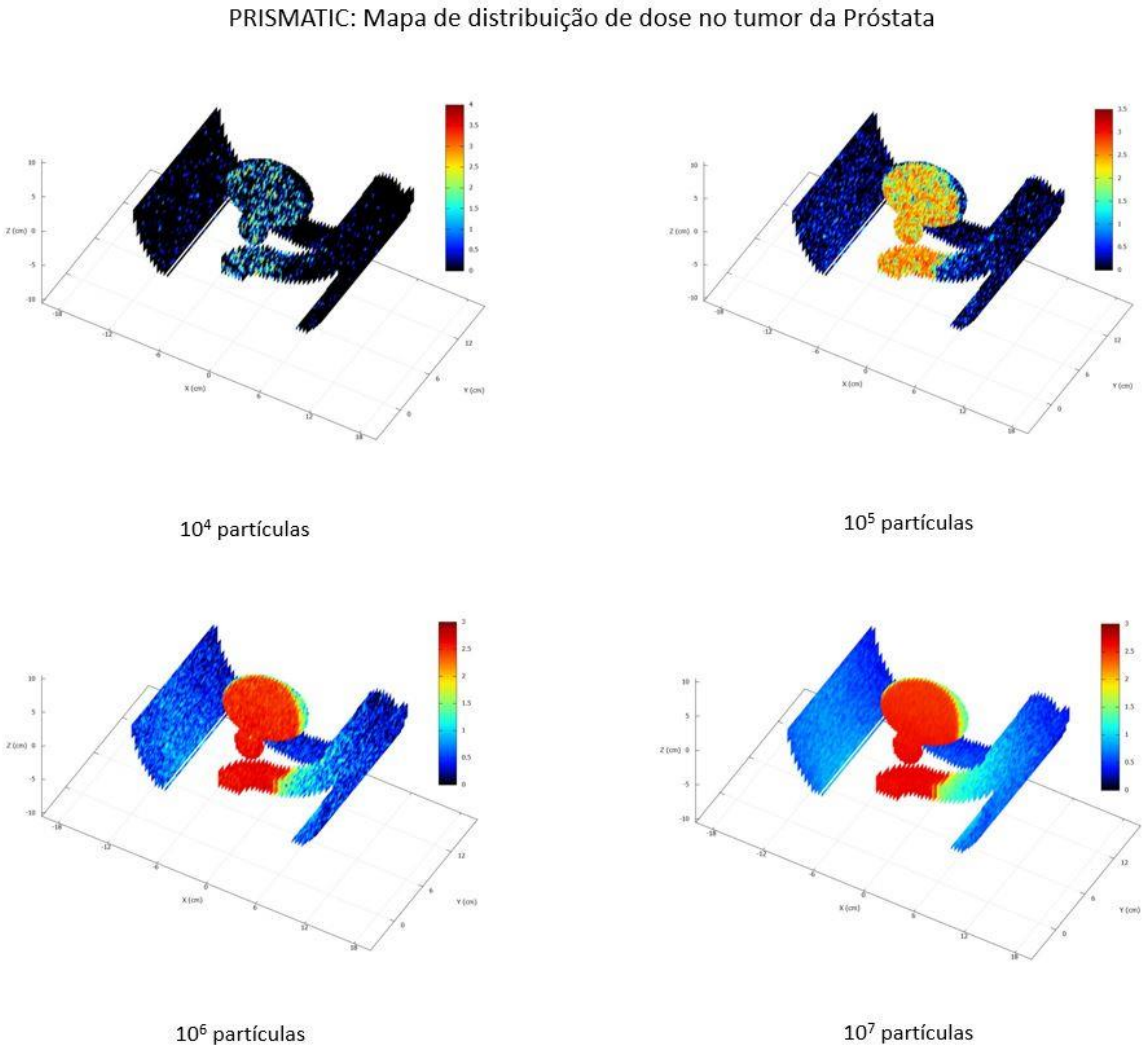
Fonte: Desenvolvido pelo próprio autor

Figura 33 - Mapa de distribuição de dose no Pulmão: Simulação PRISMATIC



Fonte: Desenvolvido pelo próprio autor

Figura 34 - Mapa de distribuição de dose na Próstata: Simulação PRISMATIC



Fonte: Desenvolvido pelo próprio autor

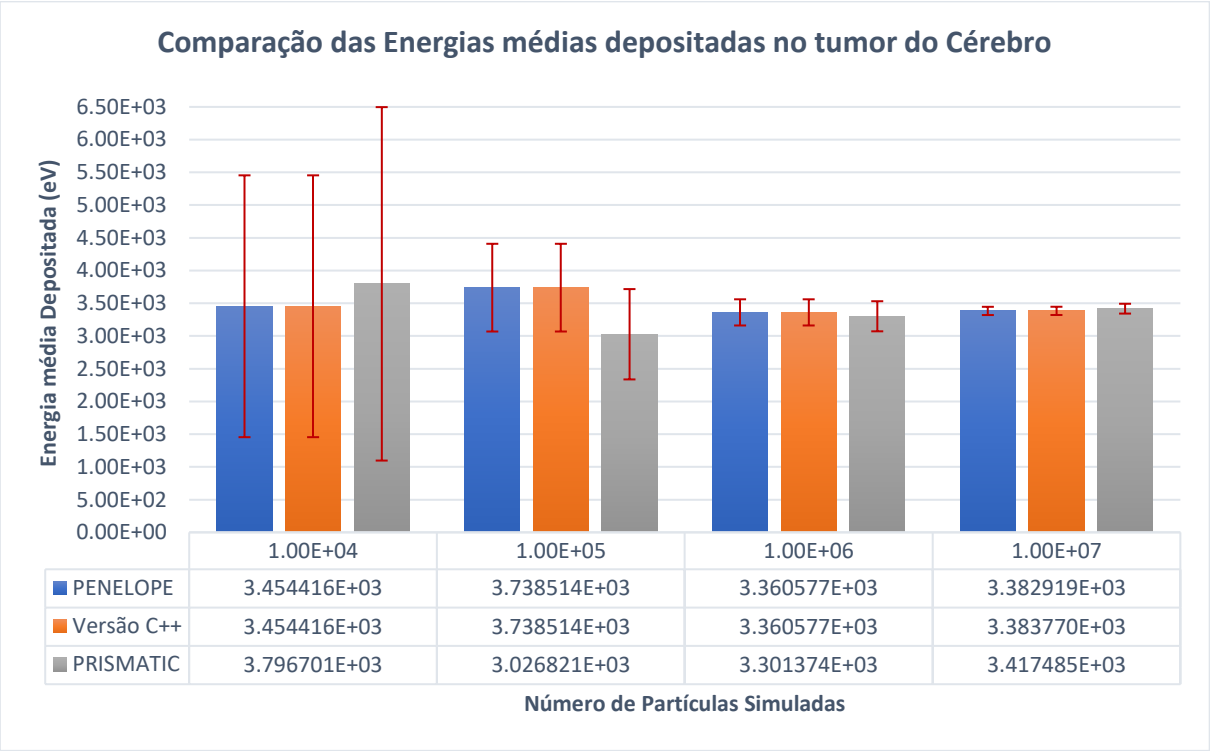
5.4 Comparação dos resultados das simulações PENELOPE-2014, versão C++ e PRISMATIC

Neste tópico são comparados os resultados de depósito de dose e mapa de distribuição de dose entre as três simulações.

5.4.1 Depósito de dose

Para fins de comparação foram selecionados os resultados obtidos com a primeira dupla de sementes iniciais de números pseudoaleatórios para as simulações executadas no PENELOPE-2014 e na versão C++ junto com a simulação executada pelo PRISMATIC que apresentou maior diferença de resultado. Nos gráficos 10, 11 e 12 são apresentados os depósitos de dose para cada tipo de tumor.

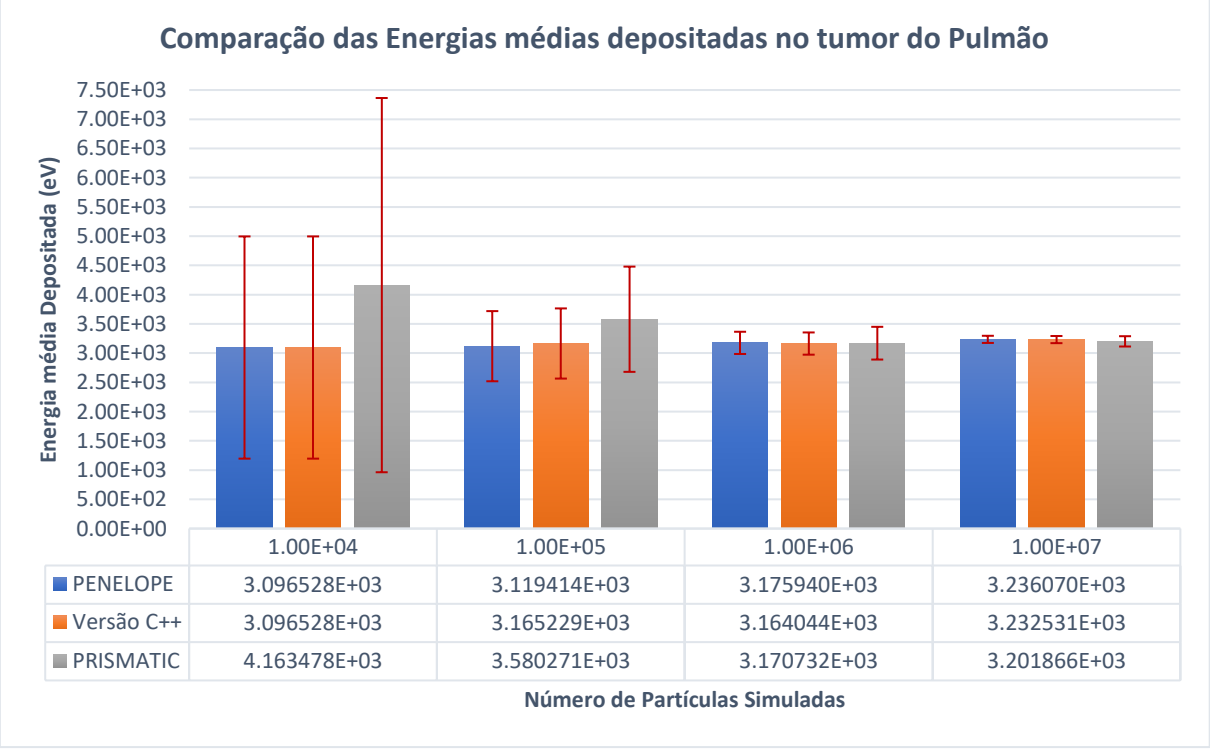
Gráfico 10 - Energia média depositada no tumor do Cérebro: Comparação



	1.00E+04	1.00E+05	1.00E+06	1.00E+07
PENELOPE	3.454416E+03	3.738514E+03	3.360577E+03	3.382919E+03
Versão C++	3.454416E+03	3.738514E+03	3.360577E+03	3.383770E+03
PRISMATIC	3.796701E+03	3.026821E+03	3.301374E+03	3.417485E+03

Fonte: Desenvolvido pelo próprio autor

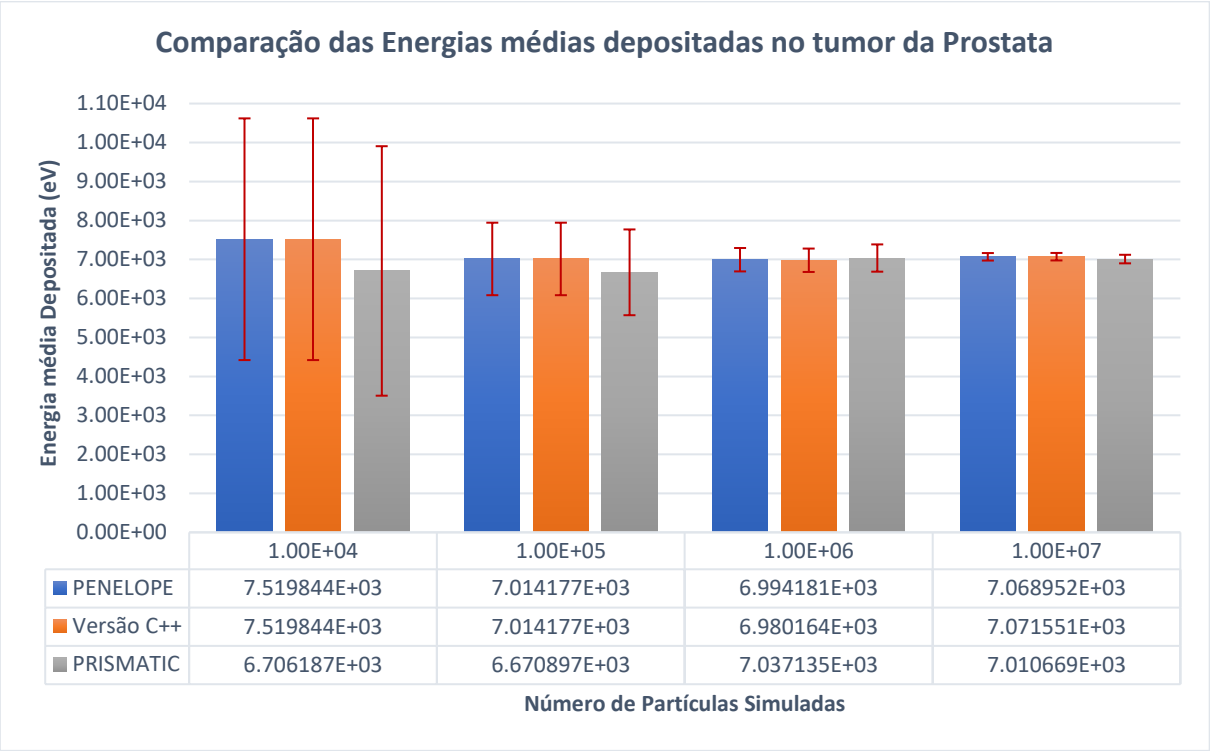
Gráfico 11 - Energia média depositada no tumor do Pulmão: Comparação



	1.00E+04	1.00E+05	1.00E+06	1.00E+07
PENELOPE	3.096528E+03	3.119414E+03	3.175940E+03	3.236070E+03
Versão C++	3.096528E+03	3.165229E+03	3.164044E+03	3.232531E+03
PRISMATIC	4.163478E+03	3.580271E+03	3.170732E+03	3.201866E+03

Fonte: Desenvolvido pelo próprio autor

Gráfico 12 - Energia média depositada no tumor da Próstata: Comparação



Fonte: Desenvolvido pelo próprio autor

Analisando os três gráficos é possível verificar uma alta similaridade entre os resultados das simulações executadas no PENELOPE-2014 e na versão C++. Os resultados variaram de 98,5% a 100% de exatidão conforme aumenta ou diminui o número de partículas simuladas até chegar a um valor de 99,9% com a carga máxima executada de 10^7 partículas. Os resultados de depósito de dose apresentando uma alta taxa de exatidão, mesmo com poucas partículas simuladas, nos possibilitam afirmar que a versão C++ é altamente compatível com o PENELOPE-2014. Atribuímos as pequenas variações de resultado a arredondamentos realizados dentro do código fonte do PENELOPE-2014, principalmente em situações de cruzamento de uma fronteira entre os corpos que formam a geometria, e por diferentes otimizações existentes entre os compiladores Fortran e C++.

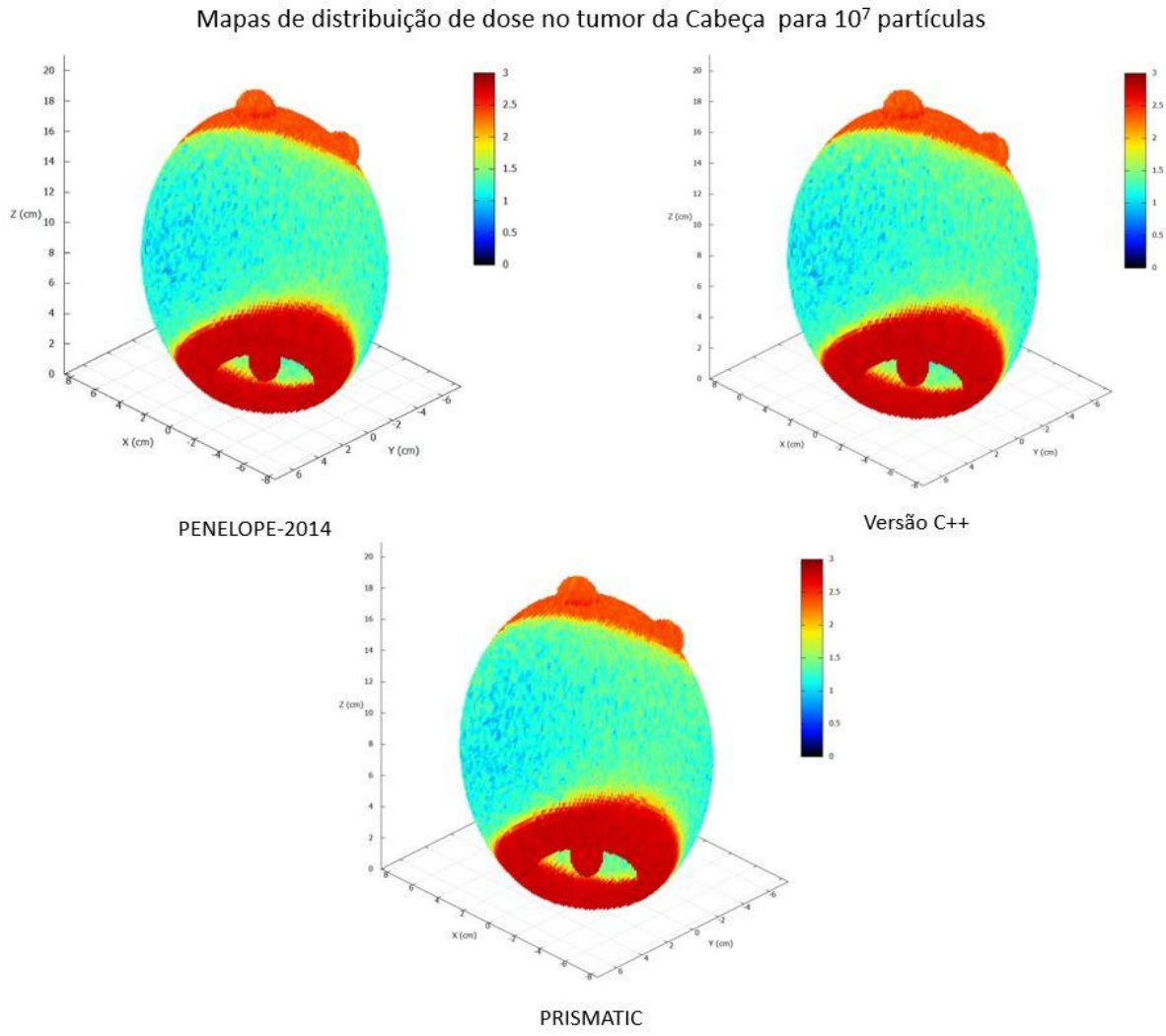
Ao se comparar a versão do PENELOPE-2014 com o PRISMATIC verifica-se que o resultado da simulação, com baixa quantidade de partículas, alcançou 74,37% de exatidão no tumor do pulmão. Essa diferença se deve aos vários números pseudoaleatórios gerados pelos diversos núcleos em execução na GPU reforçando a aleatoriedade da simulação Monte Carlo. No entanto, ao se aumentar o número de simulações até 10^7 partículas o resultado com maior diferença, observado no tumor da próstata, obteve uma exatidão de 99,17%. Quando levamos em consideração o erro estatístico para cada simulação, as versões em C++ e PRISMATIC

apresentam resultados dentro da margem de erro para todos os cenários estudados e comparados com o PENELOPE-2014.

5.4.2 Mapa de distribuição de dose

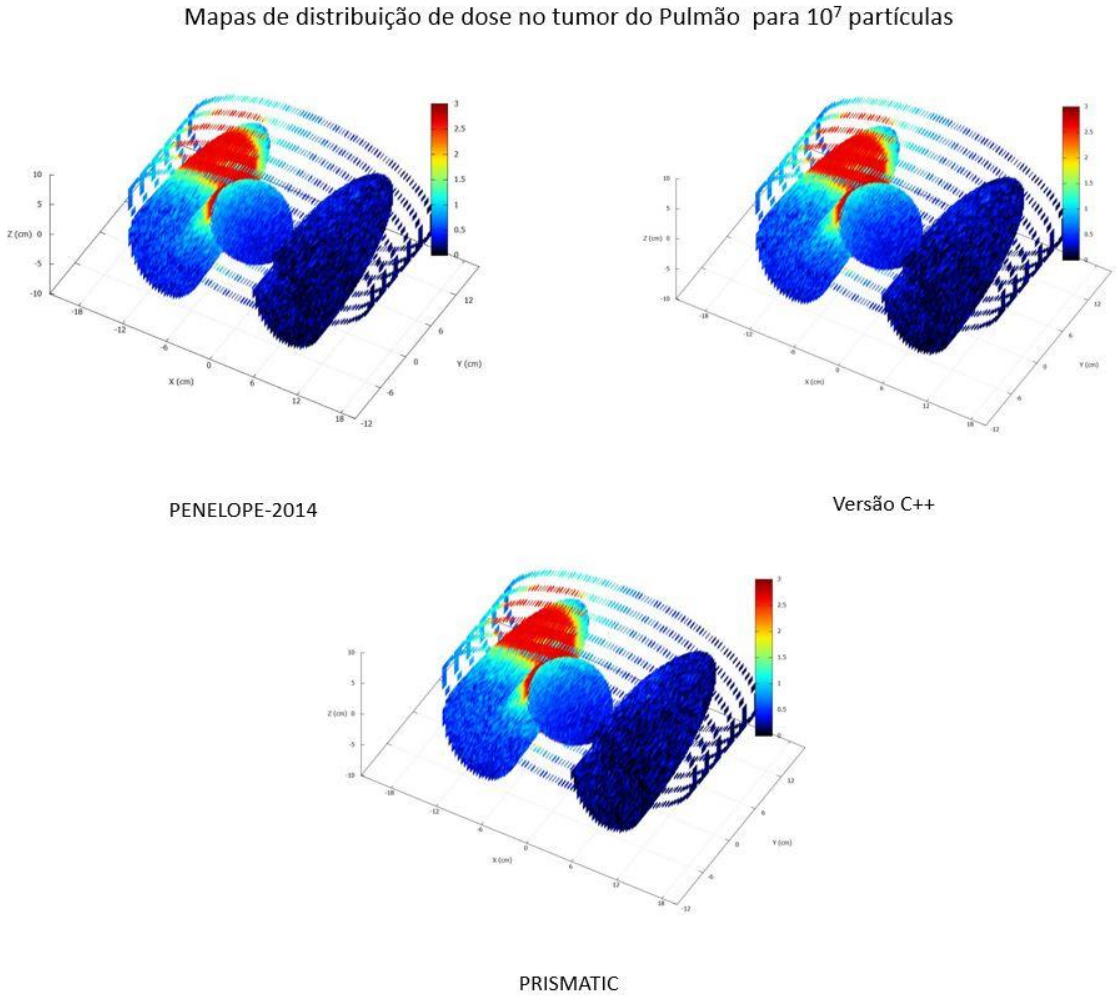
Para fins de comparação são apresentados nas figuras 33, 34 e 35 os mapas de distribuição de dose obtidos pelo PENELOPE-2014, pela versão C++ e pelo PRISMATIC com simulações de 10^7 partículas.

Figura 35 - Mapas de distribuições de dose na Cabeça: Comparação



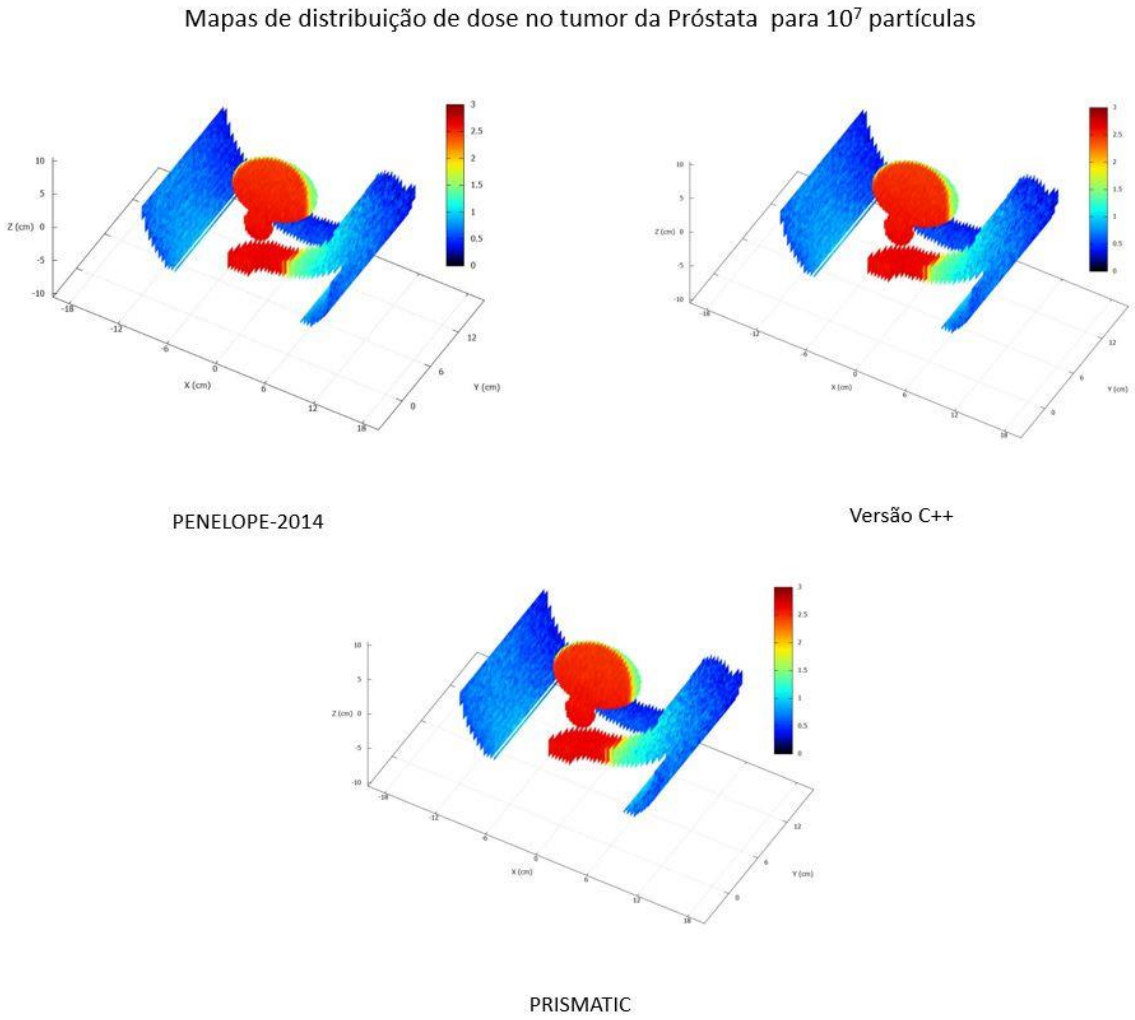
Fonte: Desenvolvido pelo próprio autor

Figura 36 - Mapas de distribuições de dose no Pulmão: Comparação



Fonte: Desenvolvido pelo próprio autor

Figura 37 - Mapas de distribuições de dose na Próstata: Comparação



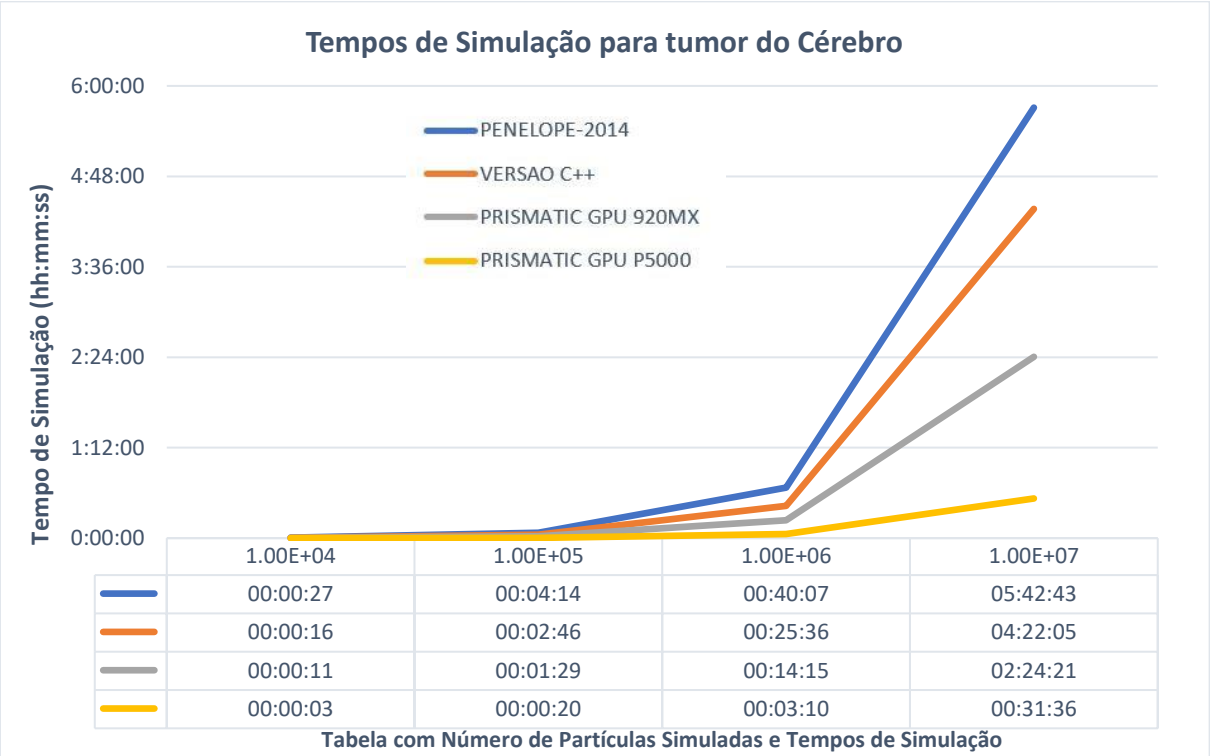
Fonte: Desenvolvido pelo próprio autor

Ao analisar as imagens das figuras 33, 34 e 35 nota-se que não é possível observar diferenças entre os mapas de distribuição de dose obtidos pelas versões C++ e PRISMATIC quando comparados com o PENELOPE-2014. Essa alta semelhança entre as imagens confere aos mapas um resultado satisfatório para visualização do depósito de dose nos órgãos estudados.

5.5 Tempo de Simulação

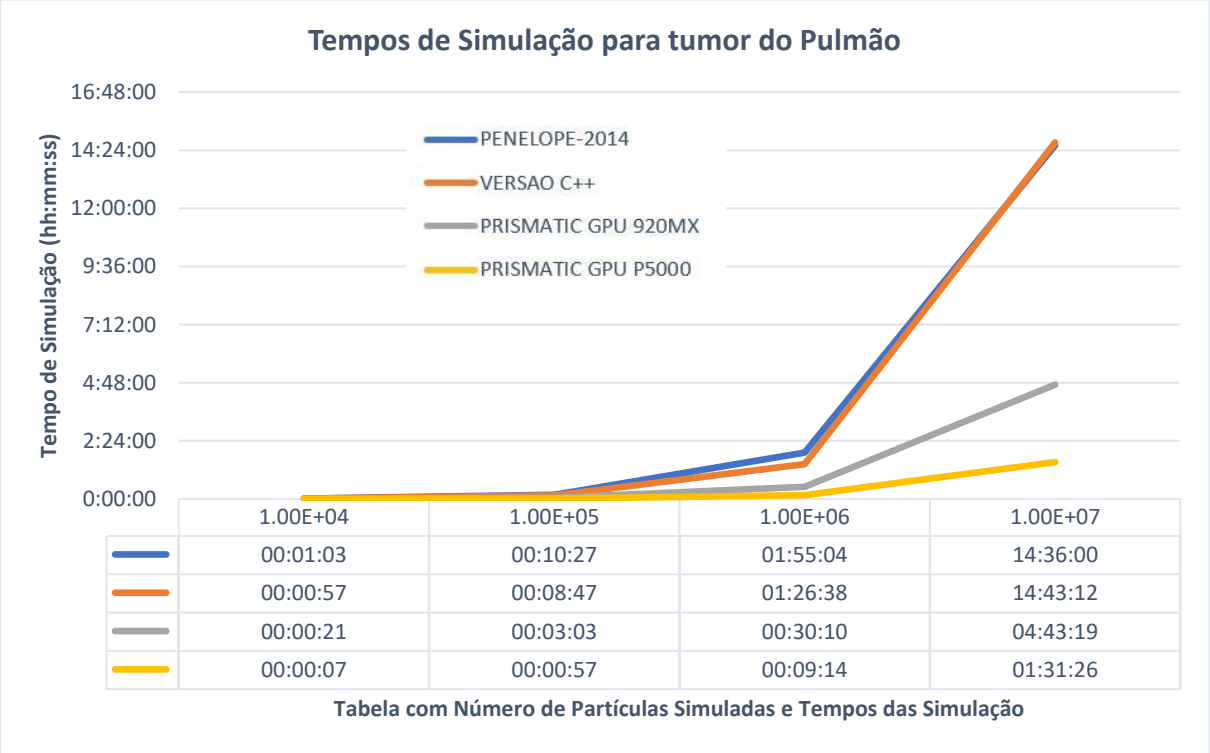
Nos gráficos 13, 14 e 15 são apresentados os tempos necessários para realizar as simulações de modo a obter os resultados de depósito de dose e o mapa de distribuição de dose nos tumores do cérebro, pulmão e próstata**.** Cada gráfico apresenta informações sobre a evolução do tempo de execução conforme aumenta-se o número de partículas simuladas em quatro ambientes de simulação, no PENELOPE-2014, na Versão C++, no PRISMATIC fazendo uso de uma GPU Nvidia 920MX e no PRISMATIC utilizando uma GPU Nvidia P5000.

Gráfico 13 - Comparação entre os tempos das simulações para o tumor no Cérebro



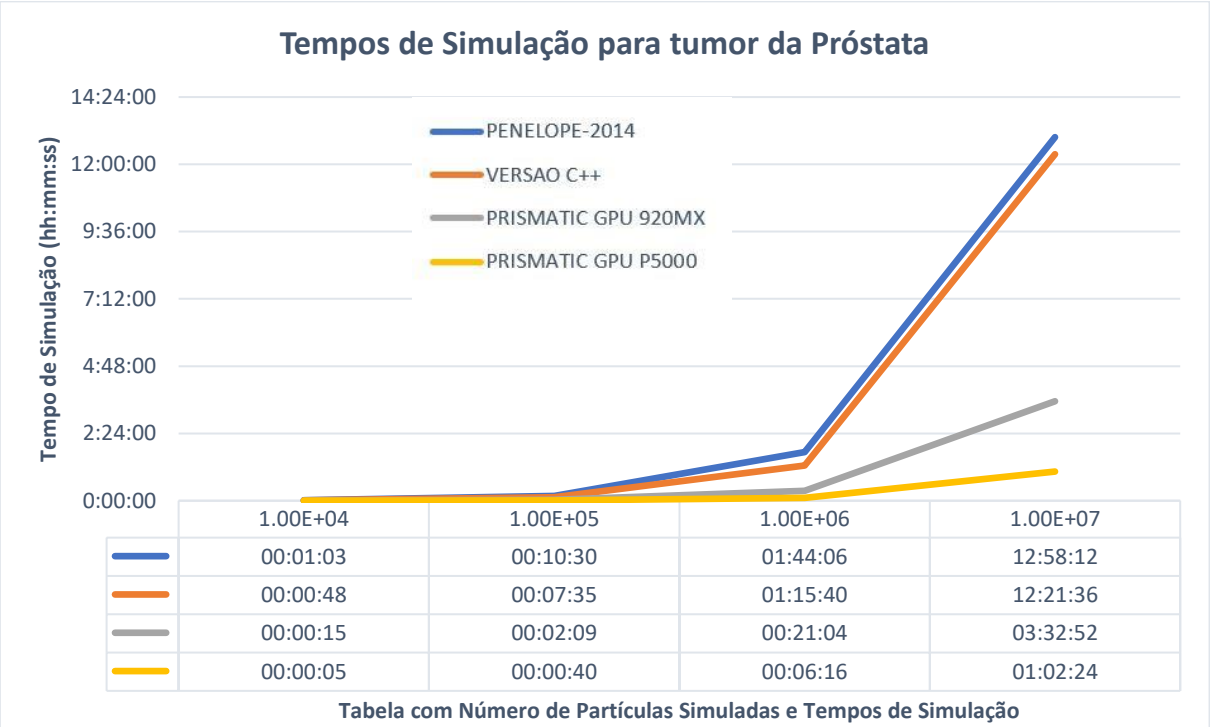
Fonte: Desenvolvido pelo próprio autor

Gráfico 14 - Comparação entre os tempos das simulações para o tumor no Pulmão



Fonte: Desenvolvido pelo próprio autor

Gráfico 15 - Comparação entre os tempos das simulações para o tumor na Próstata



Fonte: Desenvolvido pelo próprio autor

Realizando uma comparação de tempos, para simulação de 10^7 partículas, notamos que em uma geometria mais simples, como é a geometria da cabeça, a versão C++ apresentou um desempenho aproximado de 30% superior à simulação executada no PENELOPE-2014. No entanto, em geometrias mais complexas, esse ganho de desempenho não se repetiu chegando ao ponto da versão C++ ser um pouco mais lenta na geométrica do pulmão. A elaboração da versão C++ mostrou-se ser um passo importante para o desenvolvimento e validação do modelo, porém não apresenta ganhos substanciais de desempenho frente ao PENELOPE-2014.

Debruçando o olhar sobre os tempos de simulação obtidos com o PRISMATIC, nota-se um ganho de desempenho considerável mesmo com a utilização de uma GPU de modelo simples como a Nvidia 920MX. As simulações realizadas nessa GPU apresentaram um fator de desempenho entre 2 e 3 vezes mais rápido do que as simulações executadas no PENELOPE-2014. Um ganho de desempenho mais acentuado é verificado ao utilizar uma GPU mais robusta conforme observamos ao executar as simulações com o modelo Nvidia P5000. O fator de desempenho observado nesse modelo de GPU ficou entre 9 e 13 vezes mais rápido do que as simulações executadas no PENELOPE-2014.

6 CONCLUSÃO

Nesse estudo, avaliou-se a utilização da programação paralela em GPU com intuito de acelerar a execução de uma simulação Monte Carlo, aplicada ao transporte de radiação, a fim de determinar a possibilidade de tornar o método uma opção clinicamente viável de ser utilizada no planejamento radioterápico.

A metodologia empregada possibilitou a determinação dos parâmetros, sub-rotinas e funções necessárias para execução de uma simulação no pacote PENELOPE-2014 e que foram essenciais para o desenvolvimento de uma nova versão em C++. Essa nova versão teve sua implementação validada levando-se em consideração a exatidão dos resultados, o que possibilitou a abertura do caminho para que, com o emprego de otimizações inerentes ao paradigma de programação paralela na plataforma CUDA, fosse desenvolvido o código PRISMATIC. Esse, por sua vez, é capaz de realizar uma execução paralela, em GPU, de uma simulação Monte Carlo aplicada ao transporte de radiação, utilizando-se de fótons como partícula primária e estruturas de corpos em geometria quadrática, para obtenção de resultados como o depósito de dose no tecido alvo e o mapa de distribuição de dose na geometria estudada.

Comparando os resultados obtidos entre as diferentes simulações executadas nos códigos PENELOPE-2014 e PRISMATIC, nota-se uma alta exatidão no depósito de dose, com diferenças inferiores a 1%, e imagens dos mapas de distribuições de doses com diferenças imperceptíveis. Em relação a performance, a execução do PRISMATIC, em diferentes modelos de GPUs, apresentou uma redução do tempo necessário de execução em todos os cenários simulados tendo como pico a execução realizada na GPU Nvidia P5000, que apresentou um fator de desempenho de aproximadamente 13 vezes superior à execução da mesma simulação no PENELOPE-2014 para o câncer de próstata.

A combinação do ganho de desempenho com a manutenção da exatidão da simulação, demonstra que o código PRISMATIC apresenta um potencial para ser utilizado como uma opção para a execução da simulação Monte Carlo aplicada ao transporte de radiação, possibilitando a redução do tempo de planejamento do tratamento em radioterapia e tornando a sua utilização viável na clínica médica de rotina.

O código PRISMATIC é passível de ser melhorado com a implementação de novas técnicas de programação paralela onde podem ser exploradas mais a fundo o uso de memórias de rápido acesso, como a memória compartilhada e a memória constante, identificação de gargalos inerentes a divergências de *threads*, recorrentes em códigos que utilizam o Método Monte Carlo, e remodelagem da estrutura de dados implementada que, por compatibilidade e fácil acesso a documentação existente, seguiu o modelo padrão do PENELOPE-2014.

Como proposta de trabalhos futuros e visando aperfeiçoar o código PRISMATIC, podemos citar a utilização de simulação em geometrias *voxelizadas* em substituição ao modelo padrão de simulação em geometrias quadráticas. Essa abordagem tem o potencial de proporcionar uma maior realidade e de reduzir o tempo de simulação, pois simplifica o deslocamento das partículas dentro do material, e abre a possibilidade de utilização de imagens de tomografia computadorizada ou ressonância magnética possibilitando aproximar o modelo de simulação ao de um sistema de planejamento radioterápico.

7 REFERÊNCIAS

ARNHOLM. **Mixed language programming using C++ and FORTRAN 77**. 2022. Disponível em: http://arnholm.org/software/cppf77/cppf77.htm. Acesso em: 08/04/2022.

BARLAS, G. **Multicore and GPU Programming**. Second Edition ed. Philadelphia: Morgan Kaufmann, 2022. 983 p. 978-0-12-814120-5.

BELLEZZO, M.; DO NASCIMENTO, E.; YORIYAZ, H. A GPU-based Monte Carlo dose calculation code for photon transport in a voxel phantom. **International Symposium on Solid State Dosimetry, Mexico: Sociedad Mexicana de Irradiacion y Dosimetria.**, 2014.

BVSMS. **Glossário Temático: Controle de Câncer**. 2013. Disponível em: https://bvsms.saude.gov.br/bvs/publicacoes/glossario_tematico_controle_cancer.pdf. Acesso em: 08/04/2021.

CHEN, W.-Z.; XIAO, Y.; LI, J. Impact of dose calculation algorithm on radiation therapy. **World journal of Radiology**, 6, n. 11, p. 874, 2014.

CINTRA, F. B. D. **Avaliação da metodologia de cálculo de dose em microdosimetria com fontes de elétrons com o uso de código MCNP5**. Universidade de São Paulo, 2010.

COSTA, G. D. M. P. D. **Estudos espectrais aplicados à radioterapia utilizando o método de Monte Carlo**. 2013. - Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto, Universidade de São Paulo.

ELCIM, Y.; DIRICAN, B.; YAVAS, O. Dosimetric comparison of pencil beam and Monte Carlo algorithms in conformal lung radiotherapy. **Journal of Applied Clinical Medical Physics**, 19, n. 5, p. 616-624, 2018.

HAN, J.; SHARMA, B. **Learn CUDA Programming: A beginner's guide to GPU programming and parallel computing with CUDA 10. x and C/C++**. Packt Publishing Ltd, 2019. 478 p. 178899129X.

HENDERSON, N. *et al.* A CUDA Monte Carlo simulator for radiation therapy dosimetry based on Geant4. **SNA+ MC 2013-Joint International Conference on Supercomputing in Nuclear Applications+ Monte Carlo**, 2014.

ICRU. Report 85: Fundamental Quantities and Units for Ionizing Radiation **Journal of the ICRU**, 11, n. 1, 2011.

INTEL. **Processador Intel Core i7-1165G7**. 2021. Disponível em: https://www.intel.com.br/content/www/br/pt/products/processors/core/i7-processors/i7-10510u.html . Acesso em: 13/04/2021.

JAMES, F. A review of pseudorandom number generators. **Computer physics communications**, 60, n. 3, p. 329-344, 1990.

JIA, X. *et al.* GPU-based fast Monte Carlo simulation for radiotherapy dose calculation. **Physics in Medicine & Biology**, 56, n. 22, p. 7017, 2011.

KIRK, D. B.; WEN-MEI, W. H. **Programming massively parallel processors: a hands-on approach**. Morgan kaufmann, 2016. 012811987X.

KRAMER, G. H.; CROWLEY, P.; BURNS, L. C. Investigating the impossible: Monte Carlo simulations. **Radiation protection dosimetry**, 89, n. 3-4, p. 259-262, 2000.

MEDEIROS, M. Modelagem Computacional de um Acelerador Linear e da Sala de Radioterapia para Cálculo da Dose Efetiva em Pacientes Submetidos a Tratamento de Câncer de Próstata. **Rio de Janeiro: UFRJ/COPPE**, 2018.

MIRZAPOUR, M. *et al.* Fast Monte-Carlo Photon Transport Employing GPU-Based Parallel Computation. **IEEE Transactions on Radiation and Plasma Medical Sciences**, 4, n. 4, p. 450-460, 2020.

MISIC, M. J.; DURDEVIC, D. M.; TOMASEVIC, M. V., 2012, **Evolution and trends in GPU computing**. IEEE. 289-294.

NVIDIA. **CUDA C++ Programming Guide**. 2021. Disponível em: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html . Acesso em: 10/02/2022.

PEETERS, A. *et al.* How costly is particle therapy? Cost analysis of external beam radiotherapy with carbon-ions, protons and photons. **Radiotherapy and oncology**, 95, n. 1, p. 45-53, 2010.

PODGORSAK, E. B. **Radiation oncology physics: A Handbook for Teachers and Students**. IAEA Vienna, 2005. 686 p.

PODGORSAK, E. B. **Radiation Physics for Medical Physicists**. 3 ed. Springer, 2016. 955 p.

SALVAT, F. **PENELOPE-2014: A Code System for Monte Carlo Simulation of Electron and Photon Transport.** Facultat de Física (ECM and ICC), Universitat de Barcelona.: OECD Nuclear Energy Agency France, 2015.

SHEIKH-BAGHERI, D.; ROGERS, D. Sensitivity of megavoltage photon beam Monte Carlo simulations to electron beam and other parameters. **Medical physics**, 29, n. 3, p. 379-390, 2002.

SOSUTHA, S.; MOHANA, D. Heterogeneous parallel computing using cuda for chemical process. **Procedia Computer Science**, 47, p. 237-246, 2015.

SUN, Y. *et al*. Summarizing CPU and GPU design trends with product data. **arXiv preprint arXiv:1911.11313**, 2019.

SÁ, J. R. *et al.* Interação da Física das Radiações com o Cotidiano: uma prática multidisciplinar para o Ensino de Física. **Revista Brasileira de Ensino de Física**, 39, 2016.

TECHPOWERUP. **GeForce MX350**. 2021. Disponível em: https://www.techpowerup.com/gpu-specs/geforce-mx350.c3494. Acesso em: 13/04/2021.

TIAN, Z. *et al.* A GPU OpenCL based cross-platform Monte Carlo dose calculation engine (goMC). **Physics in Medicine & Biology**, 60, n. 19, p. 7419, 2015.

VERHAEGEN, F.; SEUNTJENS, J. Monte Carlo modelling of external radiotherapy photon beams. **Physics in medicine & biology**, 48, n. 21, p. R107, 2003.

WANG, Y. *et al.* A GPU-accelerated Monte Carlo dose calculation platform and its application toward validating an MRI-guided radiation therapy beam model. **Medical physics**, 43, n. 7, p. 4040-4052, 2016.

WHO. **Report on cancer: setting priorities, investing wisely and providing care for all**. World Health Organization, 2020. 149 p.

YORIYAZ, H. Método de Monte Carlo: princípios e aplicações em Física Médica. **Revista Brasileira de Física Médica**, 3, n. 1, p. 141-149, 2009.

YOSHIMURA, E. M. Física das Radiações: interação da radiação com a matéria. **Revista Brasileira de Física Médica**, 3, n. 1, p. 57-67, 2009.

APÊNDICE A: Código fonte do programa principal com as chamadas das funções de simulação em GPU

```
int main() {

    //Alocação da memória da CPU para as structs
    memoryAllocCPU();

    //Ativa simulação na GPU (GPU: 1, CPU: 0)
    int simGPU = 1;

    //Le os arquivos de entrada e inicializa a simulação.
    pmrdr2_();

    //seta o tamanho da piha com a constante pilhaPart = 3072
    sizeTrack = pilhaPart;

    if (*CSOUR0_.JOBEND != 0)
        goto L103;

    //Aloca a memoria na GPU para as structs
    memoryAllocGPU();

    //Inicializa as semnentes para GPU e para a CPU
    initializeISSEDS_();
    *RSEED_.ISEED1 = IS1[0];
    *RSEED_.ISEED2 = IS2[0];

    //seta a quantidade de threads por block.
    dim3 block(blockSize);
    //seta a quantidade de blocks na grid.
    dim3 grid(ceil(sizeTrack / block.x)+1);

    //trasnferindo dados para GPU
    transfCPU_to_GPU();

    //Realiza a simulação enquanto não atingir o tempo ou a quantiade de particulas estipuladas.
    while ((*CNTRL_.TSEC < *CNTRL_.TSECA) && (*CNTRL_.SHN < *CNTRL_.DSHN)){

        //Limpa as variaveis de quantidades de particulas secundarias.
        cleans2GPU_();

        //Cria, ordena, transfere para GPU e simula um lote de particulas primárias.
        simPriTrack_G();

        //Transfere as variaveis das partículas secundárias geradas da GPU para a CPU
        transfnTRACKSGPU_to_CPU();

        //Realiza a simulação das partículas secundárias enquanto existir.
        while ((nTRACKS_.nSECTRACK_E > 0) || (nTRACKS_.nSECTRACK_G > 0) || (nTRACKS_.nSECTRACK_P > 0))
        {
            //Simula as partículas secundárias elétrons se existir
            if (nTRACKS_.nSECTRACK_E > 0)
            {
                simSecTrack_E();
            }
```

```
54              //Simula as partículas secundárias pósitrons se existir
55              if (nTRACKS_.nSECTRACK_P > 0)
56              {
57                  simSecTrack_P();
58              }
59              //Simula as partículas secundárias fótons se existir
60              if (nTRACKS_.nSECTRACK_G > 0)
61              {
62                  simSecTrack_G();
63              }
64          }
65
66          //sincroniza a execução da GPU com a CPU.
67          gpuErrchk(cudaDeviceSynchronize());
68
69          //contabiliza a contribuição das particulas para a dose geral
70          showers_cont<<<grid,block>>>(sizeTrack);
71
72          gpuErrchk(cudaDeviceSynchronize());
73
74          //Calcula o tempo da simulação
75          timer2_(*CNTRL_.TSEC);
76
77          //Verifica se atingiu o tempo de DUMP ou os números parciais de partículas
78          //primárias para relizar a impressão parcial dos resultados.
79          if ((*CNTRL_.TSEC - *CNTRL_.TSECAD > *CNTRL_.DUMPP) || (*CNTRL_.SHN == 1e4) ||
80              (*CNTRL_.SHN == 1e5) || (*CNTRL_.SHN == 1e6) || (*CNTRL_.SHN == 1e7) ||
81              (*CNTRL_.SHN == 1e8) || (*CNTRL_.SHN == 1e9))
82          {
83
84              // retorna os dados da gpu para cpu
85              transfGPU_to_CPU();
86              gpuErrchk(cudaDeviceSynchronize());
87
88              //Atualiza a variavel TSIM
89              *CNTRL_.TSIM = *CNTRL_.TSIM + cputim2_() - *CNTRL_.CPUT0;
90
91              //Imprime os resultados parciais da simulação
92              pmwrt2_(1);
93
94              //Imprime o plotdose para gerar o mapa de distribuição de dose
95              plotdose2_();
96
97              //Atualiza o tempo da simulação
98              *CNTRL_.TSECAD = *CNTRL_.TSEC;
99              *CNTRL_.CPUT0 = cputim2_();
100
101             //Imprime o total de partículas simualadas
102             printf("  Number of simulated showers = %.6E\n", *CNTRL_.SHN);
103         }
104
105     }
106 }
```

APÊNDICE B: Código fonte para chamadas das funções que realizam a simulação das partículas primárias.

```
void simPriTrack_G()
{
    //Cria as partículas primárias para simulação
    iniPRITRACK(pilhaPart);

    //Ordena o vertor de partículas por energia
    quickSort( 0, nTRACKS_.nFINISH-1);

    //Trasnfere as partículas da memória da CPU para a GPU
    gpuErrchk(cudaMemcpyToSymbol(dg_TRACK_mod_, &PRITRACK, sizeof(hd_TRACK_MOD)));

    //Sincroniza a simulação
    gpuErrchk(cudaDeviceSynchronize());

    //Zera os contadores de dose do lote de partículas
    g_showers_ZeroCont<<grid, block>>>(sizeTrack);
    gpuErrchk(cudaDeviceSynchronize());

    //Realiza a simulação enquanto houver partícula ativa nFINISH
    while (nTRACKS_.nFINISH > 0)
    {
        //Inicia a simulação
        g_showers_Start<<<grid, block>>>(sizeTrack);
        gpuErrchk(cudaDeviceSynchronize());

        //Verifica a posiçao da particula
        g_showers_Posicao<<<grid, block>>>(sizeTrack);
        gpuErrchk(cudaDeviceSynchronize());

        //Calcula o tamanho do caminho que a particula ira percorrer
        g_showers_JUMP<<<grid, block>>>(sizeTrack);
        gpuErrchk(cudaDeviceSynchronize());

        //Desloca a particula na geometria realizando as verificações
        //se a particula cruzou uma fronteira e se permanece no sistema
        //de geometria delimitado
        g_showers_STEP<<<grid, block>>>(sizeTrack);
        gpuErrchk(cudaDeviceSynchronize());

        //Executa os eventos de interação com a chamada da função KNOCK
        g_showers_KNOCK<<<grid, block, 2*blockSize*sizeof(int)>>>(sizeTrack);
        gpuErrchk(cudaDeviceSynchronize());

        //Realiza verificações das particulas que atingiram o limite mínimo de energia
        //e se necessário encerra a historia da particula atualizando o valor nFINISH
        g_showers_EMIN<<<grid, block,2*blockSize*sizeof(int)>>>(sizeTrack);
        gpuErrchk(cudaDeviceSynchronize());

        //Adiciona as particulas secundárias na pilha e atualiza os seus contadores.
        g_showers_SECPAR<<<grid, block>>>(sizeTrack);
        gpuErrchk(cudaDeviceSynchronize());
    }
    //transfere os contadores de particulas secundárias para a CPU
    transfnTRACKSGPU_to_CPU();
}
```