

PROYECTO INTEGRADOR DE LA CARRERA DE
INGENIERÍA NUCLEAR

IMPLEMENTACIÓN DE MUESTREO ON-THE-FLY EN
EL CÓDIGO KDSOURCE

Matías Agustín Giménez

Dr. Ariel Márquez
Director

Ing. Zoe Prieto
Co-directora

Miembros del Jurado
Dr. Javier Dawidowski
Mgter. Norberto Schmidt

23 de Junio de 2024

Departamento de Física de Reactores y Radiaciones

Instituto Balseiro
Universidad Nacional de Cuyo
Comisión Nacional de Energía Atómica
Argentina

A quienes me quieren y me celebran.

Glosario

- KDE: kernel Density Estimation.
- API: application Programming Interface.
- MCPL: monte Carlo Particle List.
- RAM: random Access Memory.
- On-the-fly: técnica de procesamiento dinámico en tiempo real.
- Samplear: muestrear partículas de una fuente.
- Thread: hilo de ejecución en paralelo.
- Batch: conjunto de partículas muestreadas.
- Track: propiedades de una partícula registrada al cruzar una superficie.
- Tally: contador de una determinada cantidad en el espacio de fases.
- Wrapper o pipe: medio por el que se da el intercambio de información entre dos códigos.
- H*(10): dosis equivalente ambiental.

Índice de contenidos

Glosario	v
Índice de contenidos	vii
Índice de figuras	ix
Índice de tablas	xiii
Resumen	xv
Abstract	xvii
1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	3
2. Herramientas utilizadas	5
2.1. Conceptos fundamentales en desarrollo orientado a objetos	5
2.2. KDSouce	6
2.2.1. Kernel Density Estimation	7
2.2.2. Método KDE en KDSouce	8
2.2.3. Flujo de trabajo con cualquier programa de transporte de radiación por el método Monte Carlo	9
2.2.4. Problemática a resolver	10
2.2.5. Arquitectura del código KDSouce	11
2.3. OpenMC	12
2.3.1. Cadena de cálculo utilizada para simulaciones del tipo <i>fixed source</i>	13
2.3.2. Formas de declarar una fuente fija en OpenMC	14
2.4. MCPL y HDF5	15
2.5. OpenMP	16
2.6. XML	16

3. Desarrollo	17
3.1. Introducción	17
3.2. Implementación de la clase <code>KernelDensitySource</code>	18
3.2.1. Muestreo de la clase <code>KernelDensitySource</code>	19
3.3. Paralelización	21
3.3.1. Desafíos encontrados e implementación	22
4. Verificación continua del desarrollo	27
4.1. Prueba en vacío	27
4.2. Prueba en conducto de agua liviana	29
5. Haz de neutrografía del reactor RA-6	35
5.1. Descripción del instrumento de irradiación	35
5.2. Utilización de <code>KDSource</code> <i>on-the-fly</i>	36
5.3. Normalización y cálculo de factores de fuente	38
5.4. Cálculo del flujo escalar de neutrones y fotones	39
5.5. Cálculo de la dosis equivalente ambiental $H^*(10)$	43
6. Conclusiones	49
6.1. Trabajo futuro	50
A. Implementación de la clase <code>KernelDensitySource</code>	51
A.1. Inclusión de <code>KDSource</code> dentro de <code>OpenMC</code>	51
A.2. Clase <code>KernelDensitySource</code>	51
A.3. Implementación de una versión apta para simulaciones <i>multi-threading</i>	53
A.4. Incorporación de unit tests	56
Bibliografía	57
Agradecimientos	61

Índice de figuras

1.1. Ejemplo de una simulación de transporte de radiación por el método Monte Carlo con baja estadística en la región del punto B.	2
1.2. Introducción al funcionamiento de <code>KDSouce</code> como método de reducción de va- rianza.	3
2.1. Introducción al funcionamiento de <code>KDSouce</code> como método de reducción de va- rianza, acoplando dos simulaciones mediante listas de partículas referidas a una misma superficie. Se presenta un ejemplo de cómo se define una partícula en una simulación.	7
2.2. Flujo de trabajo completo de la herramienta <code>KDSouce</code>	10
2.3. Introducción al funcionamiento de un muestreo <i>on-the-fly</i> de <code>KDSouce</code>	11
2.4. Línea de cálculo utilizada durante el proyecto, para simulaciones <i>fixed source</i> en <code>OpenMC</code>	14
3.1. Clases y funcionalidades de la librería de <code>KDSouce</code> utilizadas en esta implementación.	18
3.2. Flujo de trabajo interno de <code>KDSouce</code> y <code>OpenMC</code> , funcionando en simultáneo la simulación con la creación de partículas por <code>KDSouce</code>	20
3.3. Incorporación del tipo de fuente <code>KernelDensitySource</code> , que hereda la clase madre <code>source</code>	21
3.4. Esquema representativo del funcionamiento no controlado del muestreo de partículas al utilizar varios hilos en paralelo. De un lado se encuentra el código de <code>OpenMC</code> y del otro lado el código de <code>KDSouce</code> al cual múltiples hilos acceden a la vez, sobrescribiendo direcciones de memoria comunes.	22
3.5. Esquema representativo del funcionamiento en simultáneo de <code>KDSouce</code> y <code>OpenMC</code> , en una simulación utilizando varios hilos en paralelo. De un lado se encuentra el código de <code>OpenMC</code> y del otro lado el código de <code>KDSouce</code> al cual los hilos acceden asincrónicamente.	23

4.1. Geometría de la prueba en vacío. En rojo se tiene la fuente original y en líneas punteadas azules la superficie de <i>tracks</i> , donde luego se configura una fuente de KDSource	28
4.2. Esquema tridimensional de las simulaciones en vacío. En rojo se tiene la fuente original utilizada y en líneas punteadas azules la superficie de <i>tracks</i> , donde luego se utiliza una fuente de KDSource . Se observa un ejemplo donde se generan partículas conocidas (simulación 1), y se crean nuevas a partir de su perturbación (simulación 2), las cuales son fácilmente analizables al final del conducto.	29
4.3. Geometría del conducto de agua utilizado en las simulaciones. En rojo se presentan las fuentes utilizadas.	30
4.4. Distribución espacial x-z del flujo neutrónico, normalizado por neutrón de fuente. En la Figura (a) se observa la distribución a partir de la fuente original, mientras que en las figuras (b) y (c) se presentan los flujos correspondientes al muestreo <i>on-the-fly</i> y tradicional, respectivamente.	31
4.5. Flujo neutrónico en función de la distancia z a la fuente, normalizado por neutrón de la fuente original. Se presentan solapadas las curvas para ambas rutinas de muestreo. Debajo se muestra la relación entre ambas expresiones. El error estadístico se encuentra inmerso en la curva.	32
4.6. Corriente de neutrones en la dirección positiva del eje z, normalizada por neutrón de fuente y multiplicada por su energía. Se presentan solapadas las curvas para ambas rutinas de muestreo. La misma corresponde a una superficie de <i>tracks</i> en la posición $z = 60$ cm. El error estadístico se encuentra inmerso en la curva.	32
5.1. Geometría del conducto de neutrografía del reactor RA-6, utilizada en las simulaciones.	36
5.2. Diagrama de las simulaciones realizadas a partir de las superficies S1 y S2. En rojo se muestran las fuentes utilizadas con la cantidad de partículas simuladas N_{sim} . En verde se tienen las contribuciones a los resultados de cada simulación.	37
5.3. Flujo de neutrones utilizando KDSource a partir de la superficie S1. En la Figura (a) se utilizaron 10^7 partículas, mientras en la Figura (b) se simularon 10^9	40
5.4. Flujo de fotones utilizando KDSource a partir de la superficie S1. En la Figura (a) se utilizaron 10^7 partículas, mientras en la Figura (b) se simularon del orden de 10^9	41

5.5. Flujo de neutrones utilizando KDSource a partir de la superficie S2. En la Figura (a) se utilizaron 10^7 partículas, mientras en la Figura (b) se simularon 10^9	42
5.6. Flujo de fotones utilizando KDSource a partir de la superficie S2. En la Figura (a) se utilizaron 10^7 partículas, mientras en la Figura (b) se simularon en el orden de 10^9 fotones.	43
5.7. Dosis equivalente ambiental $H^*(10)$ utilizando KDSource a partir de la superficie S2.	44
5.8. Posiciones en el contorno del conducto de neutrógrafía, para constatar los resultados obtenidos con el muestreo <i>on-the-fly</i> , con los resultados de la rutina tradicional y las mediciones experimentales anteriores.	45

Índice de tablas

3.1. Ejemplo de asignación de semillas de <code>OpenMC</code> para distintas configuraciones de hilos y batchs, para una misma cantidad de partículas totales.	24
4.1. Máximo uso de memoria <code>RAM</code> durante la simulación en el mismo caso de estudio. No se registran <i>tallies</i> o superficie de <i>tracks</i> que puedan utilizar memoria.	33
4.2. Tiempo de simulación en función del número de partículas y el número de hilos en paralelo, utilizando un cpu de 8 núcleos y solo transportando neutrones. No se tiene en cuenta el tiempo asociado a la generación de la lista de partículas perturbadas al usar la rutina tradicional.	33
4.3. Reproducibilidad de los resultados para 10^6 neutrones entre simulaciones, en función del algoritmo de muestreo.	34
5.1. Número de partículas efectivas en cada superficie, por cada lista de <i>tracks</i> utilizada. Para las partículas registradas se presenta la suma de los pesos.	38
5.2. Factor de fuente en cada superficie, para cada lista de <i>tracks</i> utilizada. Los mismos son calculados mediante las ecuaciones 5.2 y 5.3.	39
5.3. Tasa de dosis de neutrones en función de las posiciones ilustradas en la figura 5.8, para las distintas rutinas de <code>KDSource</code> . A su vez se constatan los resultados experimentales de dichas posiciones. Se presentan los errores absolutos de cada magnitud.	46
5.4. Tasa de dosis equivalente ambiental proveniente de fotones en función de las posiciones ilustradas en la figura 5.8, para las distintas rutinas de <code>KDSource</code> . Además, se muestran los resultados experimentales de dichas posiciones. Se presentan los errores absolutos de cada magnitud.	46

Resumen

Las simulaciones de transporte de radiaciones mediante el método Monte Carlo son cruciales para el modelado de geometrías complejas con un alto requerimiento de detalle. Sin embargo, su precisión tiene un costo computacional elevado, especialmente en regiones de material absorbente o alejadas de la fuente de radiación. Para superar este obstáculo, se han desarrollado técnicas como `KDSource`, que emplea la estimación de la distribución de un grupo de partículas existente, para crear nuevas con la misma correlación. Esto mejora la eficiencia de las simulaciones y permite un análisis más preciso en áreas de interés específico.

En el presente trabajo, se desarrolló una técnica de muestreo *on-the-fly* en el código de transporte de radiación Monte Carlo `OpenMC`, integrando la herramienta `KDSource`. Para la misma, se creó una fuente específica que permite la generación de partículas en el dominio de la simulación, en lugar de cargarlas en memoria, utilizando el algoritmo de muestreo de `KDSource` a partir del método KDE. Se logró reducir significativamente el uso de memoria RAM, permitiendo la simulación de un mayor número de partículas. Se implementó una versión apta para simulaciones *multithread*, asegurando resultados reproducibles entre simulaciones bajo las mismas configuraciones. Las pruebas demostraron que el uso de `KDSource` *on-the-fly* mejora la eficiencia en términos de memoria y no afecta la precisión de los resultados ni los tiempos de simulación.

Se aplicó esta técnica al caso de estudio del haz de neutrografía del reactor RA-6, logrando un aumento significativo en la estadística con relación a simulaciones anteriores, y una mejora en la precisión de los resultados, especialmente para fotones.

Palabras clave: KDE, KDSOURCE, OPENMC, MONTE CARLO, REDUCCIÓN DE VARIANZA

Abstract

Radiation transport simulations using the Monte Carlo method are crucial for modeling complex geometries with high detail requirements. However, their accuracy comes with a high computational cost, especially in regions with absorbing materials or areas far from the radiation source. To overcome this obstacle, techniques such as `KDSource` have been developed. This tool employs the estimation of the distribution of an existing group of particles to create new ones with the same correlation. This improves simulation efficiency and allows for more precise analysis in specific areas of interest.

In the present work, an *on-the-fly* sampling technique was developed within the `OpenMC` Monte Carlo radiation transport code, integrating the `KDSource` tool. A specific source was created to enable particle generation within the simulation domain instead of loading them into memory, utilizing the `KDSource` sampling algorithm based on the KDE method. A significant reduction in RAM usage was achieved, allowing the simulation of a larger number of particles. A suitable multi-threaded simulation version was implemented, ensuring reproducible results across simulations under identical configurations. The performed tests demonstrated that the *on-the-fly* use of `KDSource` enhances efficiency in terms of memory without affecting the accuracy of results or simulation times.

This technique was applied to the case study of the neutron imaging instrument of the RA-6 reactor, resulting in a significant increase in statistics compared to previous simulations, and an improvement in the precision of results, especially for photons.

Keywords: KDE, KDSOURCE, OPENMC, MONTE CARLO, VARIANCE REDUCTION

Capítulo 1

Introducción

1.1. Motivación

En el ámbito de las simulaciones de física de radiaciones, el método Monte Carlo es una herramienta indispensable para modelar geometrías complejas o extensas que requieren un alto nivel de detalle. Es especialmente adecuado en problemas que presenten un campo de radiación con gran anisotropía, como el diseño de blindajes o haces de irradiación; gracias a que no introduce aproximaciones sobre la distribución angular del flujo.

Sin embargo, a pesar de que ofrecen gran precisión, llevan un elevado costo computacional, ya que la fiabilidad de los resultados está estrechamente ligada a la estadística de las simulaciones. Esto plantea un desafío en áreas alejadas de la fuente de radiación o de material absorbente, donde el número de partículas simuladas puede ser insuficiente. En la Figura 1.1 se ilustra un ejemplo en el que se tiene una fuente de radiación dentro de un recinto con una abertura que funciona como conducto para la radiación. En esta configuración, se desea estimar la dosis en los puntos A y B. Mientras que al punto A llega una cantidad significativa de partículas, para que la radiación alcance el punto B, debe atravesar zonas muy absorbentes como el blindaje y el contenedor. Esto provoca que, para obtener resultados confiables en el punto B, se necesite simular un gran número de partículas desde la fuente, lo que resulta costoso.

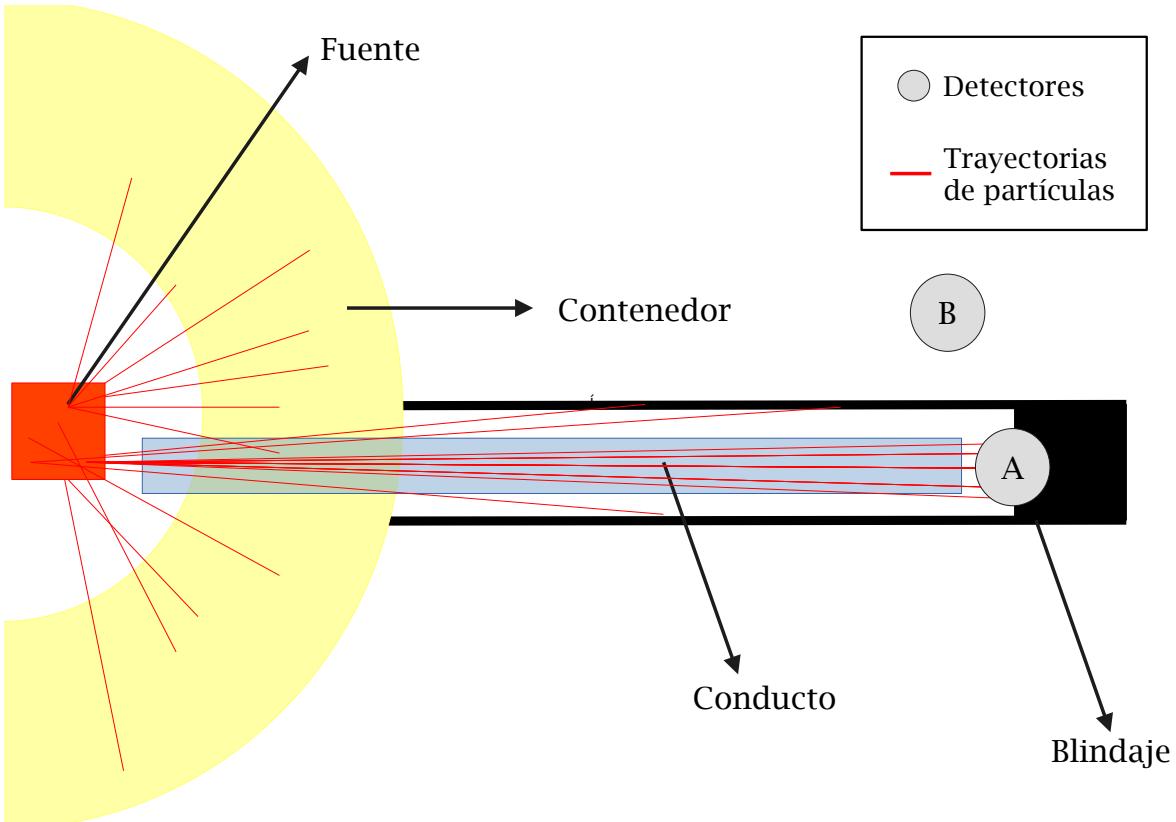


Figura 1.1: Ejemplo de una simulación de transporte de radiación por el método Monte Carlo con baja estadística en la región del punto B.

Para abordar esta problemática, a lo largo de los años se han desarrollado diversas técnicas de reducción de varianza, como el muestreo por importancia [1], la absorción implícita [2], los histogramas multidimensionales [3–5], entre otras. Estas permiten enfocar significativamente la simulación en aquellas partículas que tienen más probabilidad de llegar a una zona de interés o de contribuir a un *tally*. En ocasiones se logra transportar partículas en estas regiones, pero no las que son de interés para el caso de estudio.

El uso de histogramas multidimensionales consiste en estimar la distribución de las partículas en el espacio de variables que las definen (energía, posición, dirección, tiempo y peso estadístico), conocido como espacio de fases. El mismo es dividido en regiones, contando la suma de los pesos estadísticos de las partículas en cada una, y asignando una probabilidad constante en ella. Esta distribución discreta se utiliza luego como fuente de nuevas partículas en simulaciones subsiguientes.

En este contexto surge KDSource [6], una herramienta de código libre desarrollada en 2021 por la Comisión Nacional de Energía Atómica [7], la cual ha sido continuamente mejorada en los últimos años [8, 9]. KDSource ofrece un método avanzado de reducción de varianza para el transporte de partículas, con un enfoque distinto al uso de histogramas multidimensionales. El mismo genera una distribución continua mediante el método KDE, cuya explicación se detalla en las secciones siguientes.

KDSouce brinda la posibilidad de generar nuevas partículas, a partir de una lista de partículas y la estimación de su distribución en el espacio de fases. Por esto se propone desacoplar geométricamente el problema inicial, como se presenta en la Figura 1.2. En una primera simulación se abarca desde la fuente hasta una superficie donde se graben las partículas y la estadística sea suficiente para estimar una fuente con KDSouce. Esta fuente se materializa como una nueva lista con una mayor amplitud y es utilizada en una segunda instancia, desde esta superficie hasta las zonas de interés, donde se consigue un aumento significativo de la estadística.

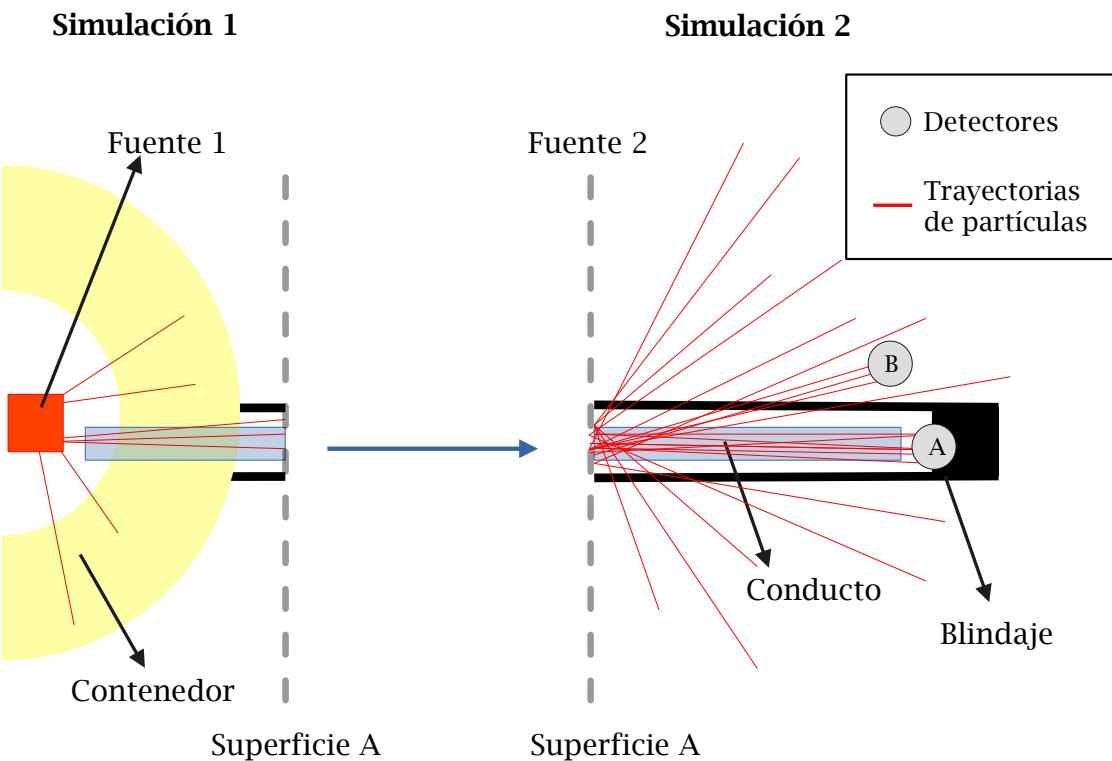


Figura 1.2: Introducción al funcionamiento de KDSouce como método de reducción de varianza.

1.2. Objetivos

El propósito de este proyecto es contribuir al desarrollo de la herramienta KDSouce mediante la optimización del uso de memoria y la expansión de sus funcionalidades. La limitación actual de la herramienta radica en la gestión de las listas de partículas necesarias al desacoplar la geometría, para realizar una segunda simulación. Estas listas, debido al elevado consumo de memoria RAM requerido para su lectura, escritura y manejo general durante la simulación, tienen una capacidad máxima de partículas que pueden contener. Este límite es específico de la memoria disponible en cada ordenador. Además, los diferentes códigos de transporte de radiación suelen emplear formatos distintos para manipular estas listas, lo que implica la necesidad de convertir

los archivos al cambiar de formato. En la rutina de trabajo más general de `KDSource`, la cual es aplicable a múltiples códigos de transporte de radiación, se manejan dos listas de partículas. Una original, derivada del guardado de una superficie de *tracks* en una simulación, y otra que es una ampliación de ella, para ser utilizada como método de reducción de varianza. En este proyecto, nos enfocaremos exclusivamente en el uso de memoria relacionado con esta última lista, ya que la primera no suele presentar limitaciones para el uso de la herramienta.

Es evidente que existe una tensión entre aumentar la cantidad de partículas a simular y la memoria disponible para ello, lo que contradice el objetivo principal de `KDSource`. La solución preexistente a esta problemática es implementar un muestreo *on-the-fly* donde las partículas se lean de la lista original y se utilicen como fuente para la nueva simulación, siendo gestionadas en el proceso por `KDSource`. Esta metodología se detalla en las secciones siguientes, destacando que permite eliminar la gestión de memoria asociada a la segunda lista de partículas, a cambio de requerir un acoplamiento entre el código Monte Carlo utilizado y `KDSource`. Este enfoque de trabajo ya se ha aplicado a los programas `McStats` [10, 11] y `TRIPOLI` [12], pero en este proyecto se busca extender su funcionalidad a `OpenMC` [13], un *software* de transporte de neutrones y fotones. La elección de esta herramienta se debe a que es de código libre, permitiendo la implementación de un algoritmo de muestreo.

Capítulo 2

Herramientas utilizadas

Este capítulo ofrecerá una visión completa de las herramientas utilizadas, destacando su relevancia y contribución a la consecución de los objetivos establecidos. Se detalla en profundidad la herramienta **KDSouce**, el *software* de simulación Monte Carlo, los formatos de archivos y las tecnologías de programación utilizadas, todas ellas esenciales para alcanzar los objetivos establecidos.

En primer lugar, se introduce en el vocabulario usado en las secciones subsiguientes, en lo referente al desarrollo de código. Luego se presenta la herramienta **KDSouce**, con un análisis detallado de su algoritmo de generación de partículas y la arquitectura de su código. Se describe también su flujo de trabajo habitual y se aborda la problemática a resolver. Posteriormente, se expone la herramienta **OpenMC**, con un enfoque en su uso específico en el proyecto y la estructura del código utilizada. Finalmente, se mencionan otras herramientas secundarias de gran importancia debido a su constante presencia en el desarrollo del proyecto.

2.1. Conceptos fundamentales en desarrollo orientado a objetos

La herramienta **OpenMC** presenta un código en C++ programado orientado a objetos, donde se utilizan clases y objetos para modelar la simulación. En esta sección se presentará una introducción de algunos conceptos claves en el desarrollo, que serán utilizados a lo largo del capítulo:

- Clase: es la representación de un objeto o concepto real, en un lenguaje de programación orientado a objetos. Esta tiene atributos y métodos que definen su comportamiento.
- Método: es una función que pertenece a una clase y define su comportamiento. Puede acceder a los atributos de la clase y modificarlos.

- Herencia: es una relación entre clases donde una clase hija hereda los atributos y métodos de una clase madre. La clase hija puede modificar los métodos de la clase madre o agregar nuevos.
- Clase virtual: es una clase que tiene al menos un método virtual puro. Un método virtual puro es aquel que no tiene implementación en la clase base y debe ser implementado en las clases derivadas.
- Constructor: cuando se crea una instancia de una clase, se llama al constructor de la misma. Este inicializa los atributos de la clase y puede recibir argumentos para personalizar la instancia.
- Instancia: es un objeto particular, creado a partir de una clase general. Cada instancia tiene sus propios valores de atributos y puede ejecutar los métodos de la clase.

2.2. KDSource

Como se mencionó anteriormente, **KDSource** es una herramienta de código libre aplicada al transporte de radiación por el método Monte Carlo, principalmente para el cálculo de haces de irradiación. La misma propone una técnica de reducción de varianza basada en el método de *Kernel Density Estimation*, dividiendo la geometría original a partir de una superficie determinada, las cuales serán utilizadas para realizar simulaciones contiguas. Se comienza con una primera simulación donde se registran las partículas que cruzan esta superficie, como se ilustra en la Figura 2.1. Se genera una lista de partículas en un formato específico que contiene todas las variables que componen el espacio de fases, así como el tipo de partícula. Luego **KDSource** va a muestrear un número mayor de partículas, con una distribución conforme a la original. Estas mismas son guardadas en una nueva lista para servir como fuente en una próxima simulación.

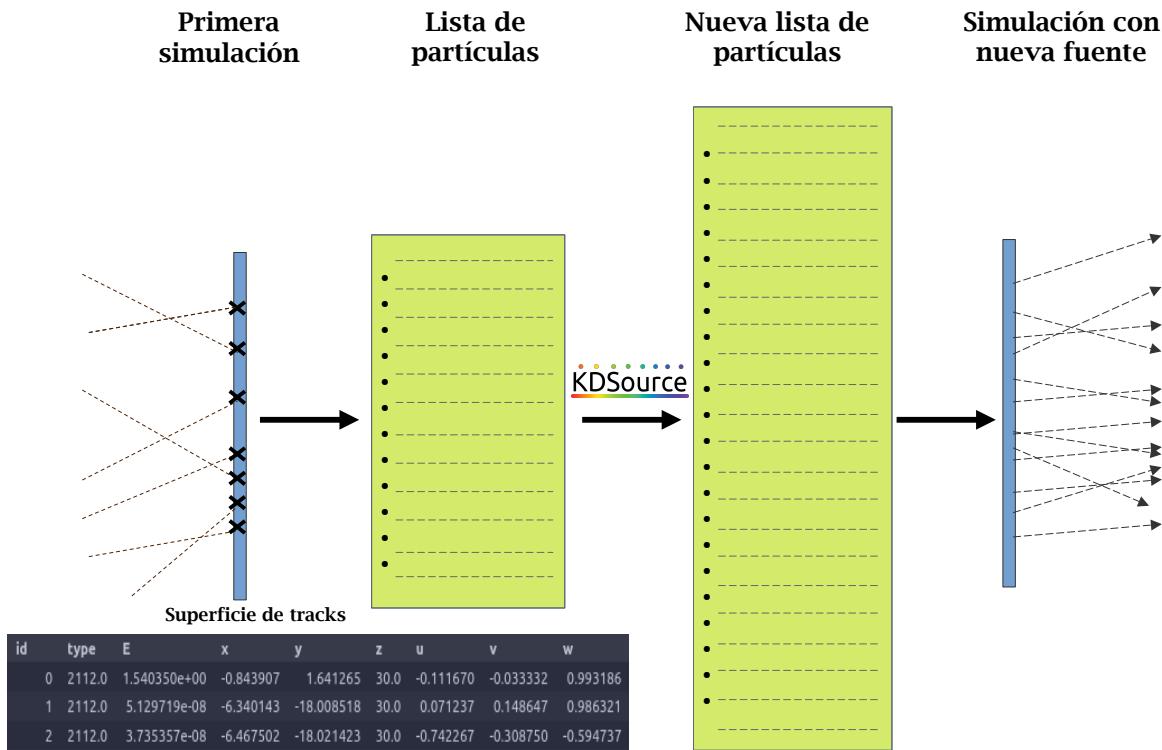


Figura 2.1: Introducción al funcionamiento de **KDSouce** como método de reducción de varianza, acoplando dos simulaciones mediante listas de partículas referidas a una misma superficie. Se presenta un ejemplo de cómo se define una partícula en una simulación.

2.2.1. Kernel Density Estimation

El método Kernel Density Estimation (KDE) es un método no paramétrico que estima una distribución de probabilidad desconocida asociada a un conjunto de muestras [14]. Esta distribución de probabilidad está compuesta por la contribución de cada muestra con su propia distribución de probabilidad. Para un conjunto de muestras $\mathbf{P} = (p_1, p_2, \dots, p_N)$, donde cada muestra es D dimensional, es decir, $p_i = \{(p_i)_1, (p_i)_2, \dots, (p_i)_D\}$, la distribución de probabilidad adjunta se estima como [6]:

$$\hat{f}(\mathbf{X}) = \hat{f}(x_1, x_2, \dots, x_D) = \sum_{i=1}^N W_i \left\{ \prod_{j=1}^D \frac{1}{h} K \left(\frac{x_j - (\tilde{p}_i)_j}{h} \right) \right\} \quad (2.1)$$

donde K representa una función *kernel* y h es el ancho de banda. W es un peso asociado al aporte de la muestra i a la distribución final. Cada función K está centrada en $(\tilde{p}_i)_j$, que es el valor de la componente j de cada muestra, normalizado por la desviación estándar de la variable j . Existen otros métodos KDE donde h es variable para cada muestra; estos métodos se llaman KDE adaptativo [15].

La solución que brinda este método está fuertemente ligada a la selección de los anchos de banda y las funciones *kernel*, así como al grupo de muestras inicial. En contraste con otros métodos basados en histogramas, presenta la ventaja de generar

una función suave a partir de un grupo discreto de datos, teniendo en cuenta los valores exactos de cada muestra, sin la necesidad de agruparlos en *bins*.

2.2.2. Método KDE en KDSource

KDSource utiliza el método KDE para estimar la distribución en el espacio de fases adjunta a una lista de partículas [8]. El mismo está compuesto por el conjunto de dimensiones $\mathbf{X} = \{E, x, y, z, u, v, w, t, wgt\}$, donde E corresponde a la energía de una partícula; x, y y z son sus coordenadas de posición en ejes cartesianos; mientras que u, v y w son los cosenos directores de su dirección con los respectivos ejes. A su vez, t es la coordenada temporal y wgt es el peso de cada partícula. Su objetivo es poder generar partículas sintéticas con una distribución conforme a la original, para esto KDSource realiza pequeñas perturbaciones aleatorias en las variables de cada partícula, sorteadas a partir de la función *kernel* centrada en el origen y el ancho de banda asignado.

En la Ecuación 2.2 se presenta un caso unidimensional del muestreo a partir de la perturbación en δ del valor x_i presente en la lista de partículas original, con un ancho de banda h y una distribución de probabilidad $p(\delta)$.

$$\hat{x} = x_i + \delta \quad p(\delta) = h^{-1} K\left(\frac{\delta}{h}\right) \quad (2.2)$$

Este algoritmo reconstruye la distribución original 2.1 porque se utilizan todas las partículas de forma equitativa, manejando los pesos de cada una adecuadamente como se explicará más adelante.

La selección de anchos de banda se optimiza con tres posibles métodos que buscan minimizar alguna distancia definida entre la distribución real $p(x)$ y la distribución estimada por KDSource $\hat{p}(x)$. El más simple de ellos es el uso de la regla de Silverman para obtener una aproximación del ancho de banda que minimice esta distancia para cada función *kernel*. Otro método disponible es el *Maximum-Likelihood Cross-Validation* (MLCV), que consiste en dividir la muestra en K grupos, donde se entrena el modelo con $K - 1$ grupos y se evalúa con el restante. Para distintos anchos de banda, se repite este proceso K veces y se calcula el valor promedio de una figura de mérito, seleccionando aquel que la maximice. Por último se puede utilizar un método KDE adaptativo, donde el ancho de banda es variable para cada partícula, seleccionado a partir de los K vecinos cercanos según una métrica definida. Los procesos de optimización mencionados pueden utilizarse de forma combinada y se encuentran detallados en [6, 7].

2.2.3. Flujo de trabajo con cualquier programa de transporte de radiación por el método Monte Carlo

La herramienta `KDSouce` está conformada por una API en `Python` y un código fuente en `C` [16]. La primera se encarga de la definición de las variables del espacio de fases a perturbar, se elige qué función *kernel* será utilizada, y se configuran los anchos de banda correspondientes. En este momento están implementados tres tipos de *kernels* (*gaussiano*, *epanechnikov*, y *escalón*). Por otro lado, el código en `C` se encarga de realizar el muestreo de las partículas y de la gestión de los archivos en formato `MCPL`.

Como se mencionó, `KDSouce` va a estimar la distribución en el espacio de fases de una lista de partículas, para obtener partículas sintéticas como fuente de una nueva simulación. El flujo de trabajo clásico de esta herramienta está condicionado por la utilización de listas de partículas como acople entre sucesivas simulaciones. En la Figura 2.2, extraída de [6], está representado el punto de partida con una simulación en un código Monte Carlo particular, grabando las partículas que cruzan una superficie de *tracks* y guardándolas en una lista de partículas en un formato particular, generalmente nativo de cada uno. En el entorno de trabajo al que está orientado `KDSouce`, esto presenta una problemática común para quienes trabajan con distintas herramientas. Por este motivo se busca a través del formato `MCPL` (ver Sección 2.4), estandarizar el almacenamiento de partículas y generar un acople entre distintos códigos. El siguiente paso es guardar la lista de partículas en formato `MCPL` para que con la API en `Python` de `KDSouce` se genere la fuente y se optimicen los anchos de banda.

Toda la información de la perturbación se exporta a un archivo en formato `XML`, que es el núcleo de `KDSouce`. El mismo almacena la ubicación de la lista original de partículas, las variables del espacio de fases que serán perturbadas, los anchos de banda seleccionados y la función *kernel*. Es el encargado de comunicar las preferencias del usuario con el código fuente en `C` que va a realizar el muestreo de las nuevas partículas.

Continuado con la Figura 2.2, se observa que la API en `C` tiene dos opciones para comunicarse con el siguiente *software* de cálculo. Se referirá como rutina tradicional al camino B, que consta de guardar las partículas sintéticas en una lista en formato `MCPL`, que luego tiene que ser convertido a un formato compatible con el código Monte Carlo utilizado en la siguiente simulación. A su vez, el camino A representa una rutina de muestreo *on-the-fly*, donde se realiza simultáneamente la generación de partículas de `KDSouce` con la simulación de transporte. En este último caso se utiliza a `KDSouce` como una fuente que genera partículas desde el archivo `MCPL` original concurrentemente a la simulación y sin un guardado intermedio. Es importante destacar que se necesita de un acoplamiento entre ambos códigos (como se ha implementado previamente con `McStas` y `TRIPOLI`), mientras la rutina tradicional funciona con cualquier *software* de transporte de radiación que permita leer una lista de partículas.

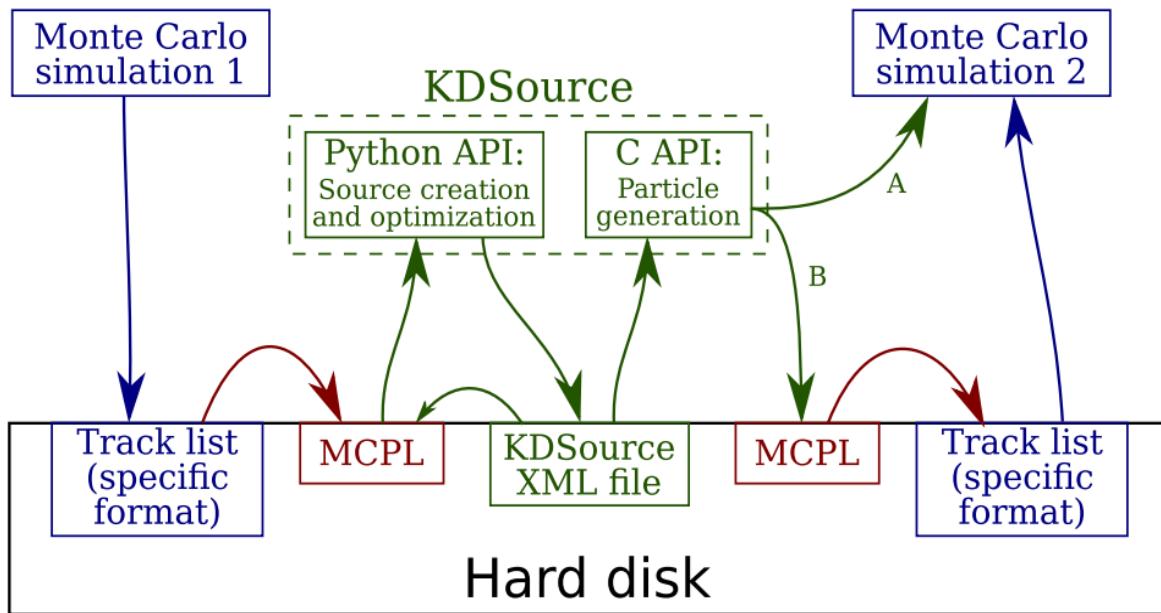


Figura 2.2: Flujo de trabajo completo de la herramienta KDSource.

2.2.4. Problemática a resolver

La utilidad del código KDSource no depende solamente de las bondades intrínsecas del mismo, sino en su capacidad para poder acoplarse a diversos códigos de Monte Carlo para servir como una herramienta de reducción de varianza. En este sentido, la principal limitación que tiene el uso de esta herramienta es la necesidad de contar con memoria disponible para el manejo de la lista de partículas perturbadas. Si bien KDSource tiene la capacidad de leer y guardar partículas en formato MCPL de forma eficiente, existen conflictos al aplicar la rutina de trabajo tradicional. Por un lado, la necesidad de guardar las partículas en un archivo intermedio, establece un límite en su tamaño a la hora de almacenarlo en la memoria masiva del ordenador. A su vez, si el código de transporte lee la lista de *tracks* completa durante la simulación, se establece otro límite en la cantidad de partículas a muestrear por lista. El mismo está relacionado con el tamaño de la memoria RAM y generalmente es más restrictivo que el anterior. En ambos casos mencionados, la limitación no es provocada esencialmente por KDSource, pero si dificulta notablemente su uso. Para computadoras con hasta 64GB de memoria RAM, el límite está en aproximadamente 10^8 partículas, magnitud no desproporcionada en el transporte de radiación. Para múltiples aplicaciones, esta cantidad no es suficiente para obtener una estadística confiable. Cabe destacar que una solución poco práctica es generar diferentes listas de partículas estadísticamente independientes y acumular los resultados de las simulaciones realizadas.

Otra solución, más amigable para el usuario, es incorporar una rutina de muestreo *on-the-fly* en el código de transporte que se quiere utilizar. Donde las partículas demandadas por el código Monte Carlo, se generan en vivo desde KDSource, leyendo de

a una por vez la lista de partículas original y aplicando el algoritmo de perturbación mencionado anteriormente. Como se ejemplifica en la Figura 2.3, no es necesario el guardado intermedio de una lista de partículas, y cualquier limitación relacionada a ella desaparece.

Existe otra restricción correspondiente al uso de memoria RAM al cambiar el formato de una lista de partículas, pero nuevamente no es propia de KDSource. Si bien el muestreo *on-the-fly* ataca parcialmente esta dificultad, ya que descarta el uso de la lista de partículas de mayor tamaño, sigue existiendo una limitación en el tamaño de la lista original, la cual puede solucionarse si se implementa una función que aproveche la capacidad de lectura y escritura eficiente de los formatos utilizados (ver Sección 2.4). La misma no fue tratada en el desarrollo del proyecto, y nos enfocaremos en la implementación de la rutina de muestreo *on-the-fly*.

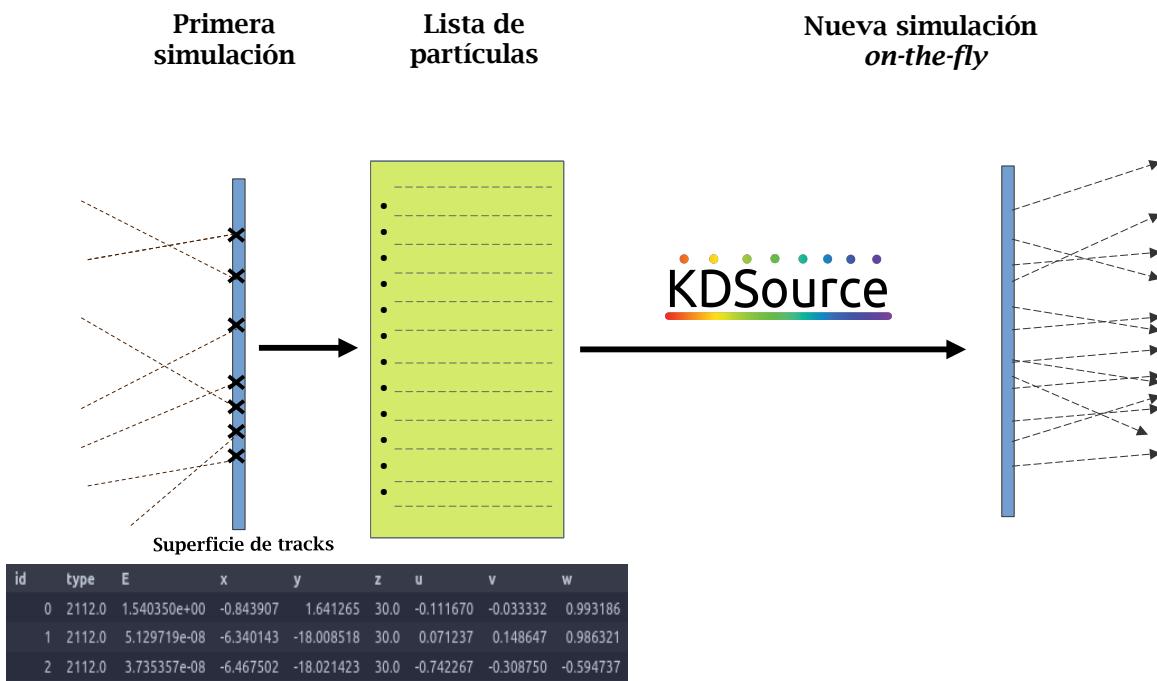


Figura 2.3: Introducción al funcionamiento de un muestreo *on-the-fly* de KDSource.

2.2.5. Arquitectura del código KDSource

Es necesario introducir la arquitectura de la herramienta KDSource [16] que fue utilizada para el algoritmo de muestreo *on-the-fly*. Como se verá más adelante, esta herramienta en su código fuente en C, está dividida en tres partes principales. Por un lado, se encuentra la capacidad de leer partículas de un archivo en formato MCPL a través de una combinación de funciones nativas de dicho formato. Simultáneamente, KDSource maneja las variables del espacio de fases presentes para la perturbación de las partículas y, por último, se provee un conjunto de métodos para realizar adecuadamente

la generación de partículas conservando la relación entre las variables en su espacio de fases.

Las principales entidades de `KDSource` con relación a la composición mencionada son las siguientes.

- **Metric:** Definición de una clase madre para las posibles variables o conjunto de variables en el espacio de fases. Consta con la definición de parámetros y una función de perturbación propia de cada variable.
- **Geometry:** Clase particular que se encarga de manejar todas las métricas definidas y de perturbar las partículas según ellas. Es la encargada de administrar los anchos de banda y la normalización de las variables.
- **Plist:** Clase específica para el manejo de la lista de partículas, actuando como un *wrapper* del archivo `MCPL` y utilizando funcionalidades previstas por la API en C de dicho formato. Permite acceder secuencialmente a la lista de partículas, producir saltos en la lectura, realizar transformaciones de variables, entre otras capacidades.
- **KDSource:** Clase para modelar la fuente de partículas, a partir de una lista de partículas y la capacidad de generar nuevas con el método KDE. Se genera a partir de un archivo `XML`, que contiene la información para instanciar las clases mencionadas.
- **KDS_sample2:** Función que se encarga de generar las nuevas partículas a partir de un objeto `KDSource`. Tiene la capacidad de asignar peso uno a todas las partículas, sin perturbar la distribución en el espacio de fases.

2.3. OpenMC

`OpenMC` es un código desarrollado en el Instituto Tecnológico de Massachusetts (MIT) enfocado en el transporte de neutrones y fotones mediante el método Monte Carlo [13]. En los últimos años ha tomado mayor relevancia, ya que, debido a su característica *open-source* ha tenido un desarrollo continuo por medio de la comunidad de usuarios. Se puede acceder a él por medio de su repositorio en GitHub [17], donde cuenta con una API en Python con la que se define completamente los parámetros de simulación Monte Carlo y su código fuente en C++ que se encarga de llevarla a cabo y del manejo de los datos de manera eficiente.

Esta herramienta se puede utilizar para realizar simulaciones de criticidad, fuente fija y de medio multiplicativo subcrítico, y cuenta con la capacidad de realizar simulaciones *multithreading*, gracias a su modelo de programación híbrida utilizando MPI y OpenMP. La misma se encuentra detallada en [18].

2.3.1. Cadena de cálculo utilizada para simulaciones del tipo *fixed source*

Las simulaciones *fixed source* son aquellas en las que la distribución o el comportamiento de las partículas se define a partir de una fuente externa, fijada a una región de la geometría, como es el caso de aplicación de `KDSource`. En la Figura 2.4 se presenta el flujo de trabajo utilizado para este tipo de simulaciones.

OpenMC provee una API en `Python` con la que se declaran todos los parámetros necesarios para una simulación y se exportan a archivos en formato `XML`, los cuales funcionan como *wrapper* entre las APIs de OpenMC y dan las condiciones para que el código en `C++` construya la simulación. En este sentido, tiene un acople similar al de `KDSource` entre su API y su código fuente.

En la figura 2.4 se observa que para definir todos los parámetros de la simulación, se deben generar cuatro archivos `XML`. Uno de ellos es `geometry.xml`, el cual contiene la información de toda la geometría de la simulación, con las condiciones de borde de las fronteras que existan y los materiales que la componen. Estos últimos se encuentran definidos en `materials.xml`, con las composiciones de elementos y sus densidades numéricas. En el mismo se registra la dirección de memoria de la biblioteca de secciones eficaces a utilizar. Luego se encuentra el archivo `settings.xml` que como su nombre lo indica, posee todos los parámetros de ejecución necesarios. En el mismo se definen el número de partículas, la fuente utilizada, el tipo de simulación, el número de *batches*, entre otras diversas opciones. Uno puede dividir la simulación en lotes llamados *batches*, donde cada uno va a simular la misma cantidad de partículas. Este parámetro es de interés para el proyecto, ya que permite acumular estadística y estimar un error a base de la comparación de los resultados de cada *batch*.

Con estos archivos declarados en principio ya se puede generar una simulación, pero normalmente es de interés poder contabilizar cantidades físicas y obtener algún resultado. Para esto se definen contadores de una determinada magnitud en el espacio de fases, a partir de un filtro en el mismo, permitiendo registrar el flujo de escalar, la potencia depositada en un material, espectros de energía, entre otras magnitudes. Los mismos son llamados *tallies* y se exportan al archivo `tallies.xml`.

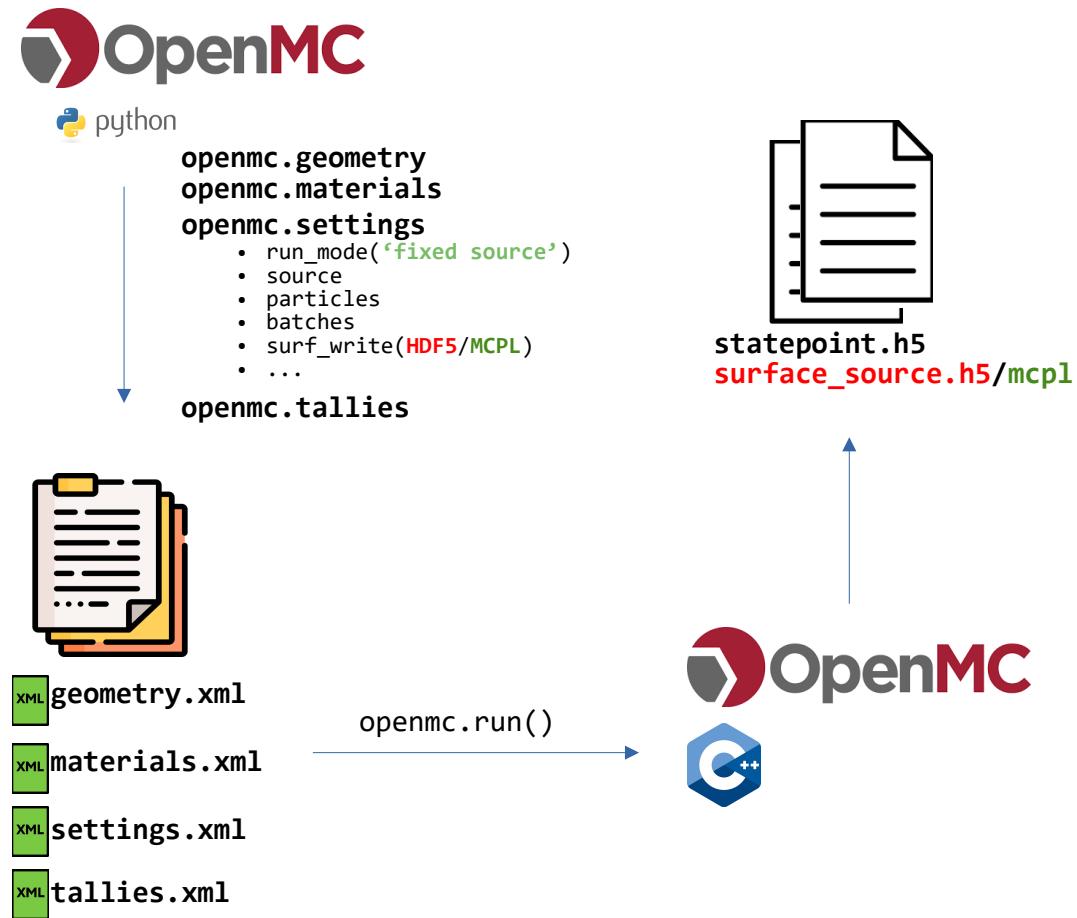


Figura 2.4: Línea de cálculo utilizada durante el proyecto, para simulaciones *fixed source* en OpenMC.

OpenMC da la posibilidad de generar una superficie de *tracks* con las partículas que cruzan alguna frontera, guardando una lista de partículas en su formato nativo HDF5. La misma puede ser guardada como archivo MCPL si se compila OpenMC con la extensión de dicho formato [19]. Esta funcionalidad se define con el parámetro `surf_write` dentro del apartado `settings`.

Continuando el flujo presentado en la Figura 2.4, la API en C++, concreta la simulación, generando dos ficheros de interés para este proyecto. Uno es el archivo `surface_source.h5/mcpl`, que contiene la lista de partículas mencionada y utilizada para una próxima simulación con KDSource. El otro es el archivo `statepoint.h5`, que recopila toda la información relevante de la simulación. En el mismo se registran las configuraciones aplicadas, la versión de OpenMC utilizada, los resultados de los *tallies* con los filtros que lo definen, entre otros datos de interés.

2.3.2. Formas de declarar una fuente fija en OpenMC

OpenMC tiene diversas formas de declarar una fuente fija externa para una simulación, en esta sección se presentarán las que fueron utilizadas durante el proyecto.

- **Source**: definición de una clase virtual para las posibles fuentes. Consta de métodos globales para el manejo de una fuente y posee un método virtual para el muestreo de partículas llamado `sample()`.
- **IndependentSource**: clase particular para definir una fuente a partir de las herramientas de estadística provistas por `OpenMC`. Permite la implementación sencilla de fuentes caracterizadas por distribuciones clásicas en el espacio de fases de las partículas.
- **CompiledSource**: Para casos donde la fuente es muy compleja o las herramientas para definir una fuente en `OpenMC` no alcanzan, se puede definir una fuente compilada de forma externa actuando como un *wrapper* entre el código en C++ y la fuente a implementar, con la flexibilidad de realizar un muestreo tan específico como uno quiera.
- **FileSource**: esta clase se utiliza para modelar una fuente a partir de una lista de partículas, con la capacidad de muestrear aleatoriamente elementos de esta lista.

2.4. MCPL y HDF5

MCPL (*Monte Carlo Particle List*) es un formato de código abierto diseñado para registrar partículas en simulaciones de transporte de radiación [20]. En un campo donde cada *software* de transporte Monte Carlo tiene su propio formato nativo de almacenamiento de partículas, MCPL surge como una solución para estandarizar este proceso. Su objetivo principal es fomentar la interacción entre diferentes códigos de transporte y herramientas relacionadas.

KDSource, por ejemplo, aprovecha este formato para gestionar las partículas. Este programa puede leer de manera secuencial archivos en formato MCPL, extraer y modificar una partícula, guardarla en otro archivo y proceder con la siguiente. Sin embargo, OpenMC utiliza el formato HDF5, el cual no es compatible directamente con KDSource. Para solucionar este inconveniente, se emplea el módulo `surfaceSource` de KDSource para convertir los archivos al formato MCPL.

HDF5, abreviatura de “*Hierarchical Data Format version 5*”, es un formato de archivo binario diseñado para almacenar grandes volúmenes de datos de manera eficiente y versátil [21]. Con una estructura jerárquica basada en grupos y *sets* de datos, HDF5 es capaz de manejar información compleja. OpenMC lo utiliza extensivamente para almacenar datos externos a la simulación, como resultados de la misma o listas de partículas.

Ambos formatos, MCPL y HDF5, ofrecen eficiencia en la lectura de datos, ya que no requieren cargar toda la información en memoria para acceder a ella. Además,

proporcionan una API en Python para facilitar el manejo de los archivos desde este lenguaje de programación.

2.5. OpenMP

OpenMP (*Open Multi-Processing*) es una interfaz de programación utilizada por OpenMC para paralelizar eficientemente su código en C++. OpenMP proporciona una API para la programación de sistemas con memoria compartida, que se basa en tres componentes principales [22]:

- **Directivas:** son instrucciones utilizadas para definirle al compilador cómo debe manejar y paralelizar un bloque de código. Se utiliza el comando “#pragma” para definir cada directiva en los lenguajes C/C++.
- **Rutinas de biblioteca:** OpenMP provee un conjunto de funciones nativas en C para acceder a información de la paralelización y manejar tareas como la gestión de los hilos, la configuración del entorno de paralelización, entre otras.
- **Variables de entorno:** variables globales para controlar el comportamiento de la paralelización.

2.6. XML

XML (*Extensible Markup Language*) es un lenguaje utilizado a lo largo del proyecto para almacenar información de manera estructurada. Está basado en una jerarquía de nodos y atributos, permite organizar datos y estructuras de manera lógica y flexible. En este proyecto, tanto OpenMC como KDSource integran sus API en Python con sus respectivos códigos fuente en C/C++ mediante archivos en formato XML.

Para manipular archivos XML en C/C++, se emplearon las bibliotecas pugixml y libxml2, mientras que en Python se utilizó la API lxml junto con el módulo xml nativo de este lenguaje.

Capítulo 3

Desarrollo

En el presente capítulo se expone el desarrollo de la técnica de muestreo *on-the-fly* dentro de `OpenMC` utilizando `KDSouce`. Este trabajo se fundamenta en la incorporación de un nuevo tipo de fuente en el código de transporte, aprovechando eficientemente las implementaciones de otros tipos de fuentes nativas del mismo.

El capítulo se estructura en una descripción inicial del acoplamiento entre ambas herramientas, seguido de una descripción de las modificaciones realizadas para permitir la ejecución de simulaciones en paralelo, así como las dificultades encontradas durante este proceso.

3.1. Introducción

Para introducir el desarrollo, se recuerda la arquitectura presentada en la Sección 2.2.5, ilustrada en la figura 3.1, donde se muestra que la herramienta `KDSouce` tiene la capacidad de generar un objeto de la clase tipo `KDSouce`, el cual contiene toda la información necesaria para la generación de nuevas partículas. Este objeto contiene las `métricas` para cada variable del espacio de fases de las partículas; posee la lista de partículas y diferentes funcionalidades para manejarlas, y almacena toda la configuración de las perturbaciones leídas desde el archivo XML. Además, cada `métrica`, asociada a una dimensión o conjunto de dimensiones, tiene la capacidad de recibir una partícula y perturbarla mediante un método propio para cada dimensión. La ventaja de este enfoque radica en la capacidad de contener toda la información necesaria para la generación de partículas perturbadas dentro de una sola instancia del tipo `KDSouce`.

A su vez, se observa que `OpenMC` ofrece diversas formas de definir una fuente de partículas para simulaciones del tipo *fixed source*, todas las cuales requieren la implementación de un método `sample()` que devuelva una partícula. A partir de la forma en que se incorpora una fuente compilada como una biblioteca externa, y cómo está implementada una fuente a partir de un archivo de partículas, surge la iniciativa de

construir un tipo de fuente específico para la nueva técnica de muestreo.

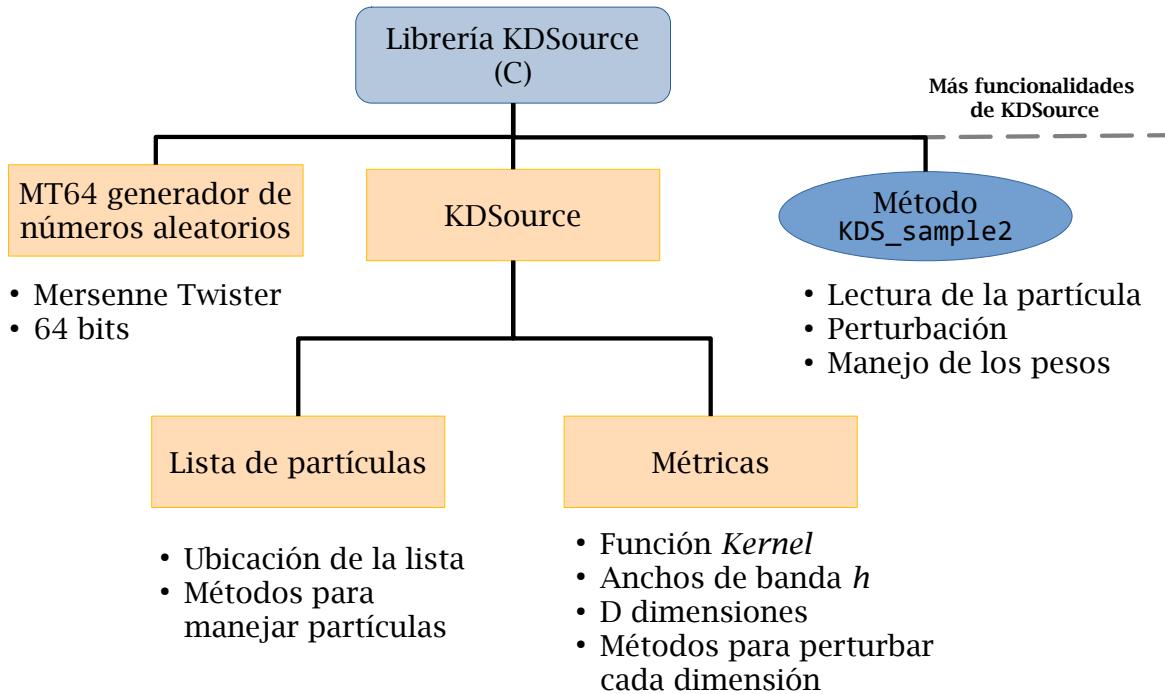


Figura 3.1: Clases y funcionalidades de la librería de `KDSource` utilizadas en esta implementación.

3.2. Implementación de la clase `KernelDensitySource`

Para poder incorporar un tipo de fuente que utilice el algoritmo de muestreo presentado en la Sección 2.2.2 sincrónicamente a la simulación, es necesario que, desde su código fuente de `OpenMC`, se puedan utilizar las funcionalidades de `KDSource`. Para ello, se realizó una compilación conjunta de ambos códigos, documentada en el Apéndice A.1, permitiendo que `OpenMC` tenga acceso completo a las librerías de `KDSource`. Además, el formato `MCPL` ya se encuentra integrado dentro de `OpenMC`, lo que facilitó el manejo de las partículas generadas por `KDSource`.

Siguiendo la línea de implementación de la clase `FileSource`, desde su definición en la API de Python hasta su construcción en el código de C++, se generó una nueva clase de fuente llamada `KernelDensitySource`. La misma es inicializada desde la API en Python de `OpenMC` y a diferencia de la clase `FileSource`, no recibe una lista de partículas, sino un archivo en formato XML previamente generado por el usuario mediante la API en Python de `KDSource`. La clase en C++ que representa a la fuente, está compuesta por un objeto del tipo `KDSource`, instanciado a partir de la información en dicho archivo.

De esta manera, `OpenMC` tiene acceso a la lista original de partículas, a las configuraciones de la perturbación y a la generación de partículas a través de dicho objeto

durante toda la simulación. Ya no es necesario cargar ningún contenedor de partículas perturbadas; en su lugar, estas pueden generarse *on-the-fly* cada vez que la simulación requiere una nueva partícula.

3.2.1. Muestreo de la clase KernelDensitySource

El muestreo de esta fuente implica utilizar métodos nativos de `KDSource` para obtener una partícula perturbada que respete la distribución del espacio de fases de la lista de partículas original, según el algoritmo de muestreo comentado en la Sección 2.2.2. Las partículas no se retienen en memoria durante toda la simulación, sino que solo se utilizan temporalmente.

El flujo de trabajo interno de la implementación se ilustra en la Figura 3.2. La simulación inicializa la historia de una partícula utilizando la función `sample` de la fuente, que al ser un método virtual de la clase madre, debe estar definido para cada instancia que la herede. En el caso particular de la clase `KernelDensitySource`, funciona como `wrapper` para emplear el método `KDS_sample2` de la librería de `KDSource`. Este último devuelve una partícula del archivo MCPL original, aplicando las configuraciones de perturbación establecidas.

En este contexto, se puede configurar `KDSource` para que restablezca los pesos de las partículas generadas a 1 o que conserve los pesos que llegan de la fuente. La primera opción mejora la estadística, debido a que se pueden utilizar otros métodos de reducción de varianza, (como la *absorción implícita*) que transportan las partículas a zonas más lejanas, en función del peso de ellas. A su vez, si se les establece un peso $w = 1$, se alcanza un número de partículas efectivas mayor que si se conservan los originales. Esta funcionalidad requiere otro algoritmo para no perturbar la distribución en el espacio de fases. Esto se resuelve mediante dos métodos distintos de reducción de varianza, *ruleta rusa* y *splitting*. La configuración de esta funcionalidad puede realizarse desde la inicialización de la fuente, y depende del valor asignado a la variable `W_critic` de la clase, como se detalla en el Apéndice A.3. Cuando dicho valor es negativo, los pesos no se restablecen, pero cuando es positivo, se aplica *splitting* a las partículas con un peso mayor y *ruleta rusa* a las de peso menor.

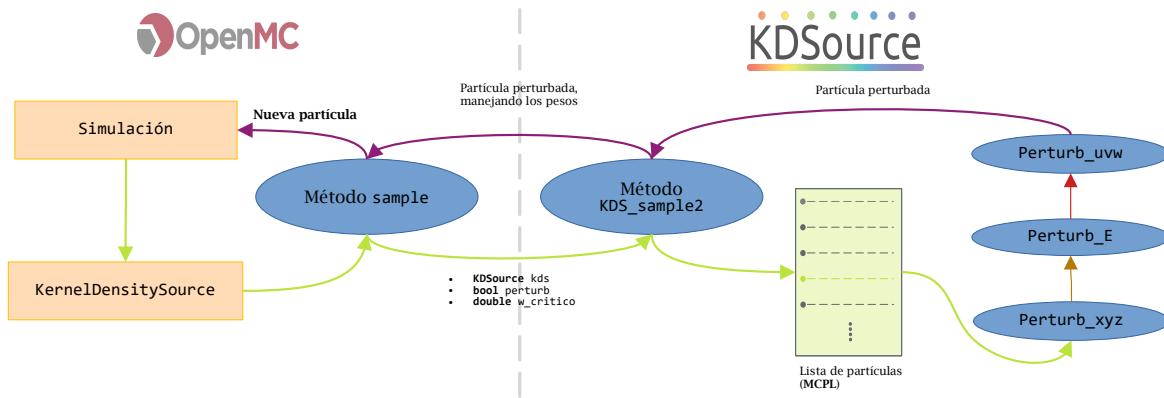


Figura 3.2: Flujo de trabajo interno de KDSource y OpenMC, funcionando en simultáneo la simulación con la creación de partículas por KDSource.

Es importante destacar que, cuando se utiliza KDSource en la rutina tradicional, se inicializa a partir de una instancia `FileSource` con un archivo de *tracks*. Esto impide la simulación en múltiples *batches* de forma nativa desde OpenMC, ya que, repetir numerosas veces la misma fuente no asegura que la convergencia de los resultados sea la correcta. En contraste, el muestreo *on-the-fly* de KDSource garantiza que cada nueva partícula mantenga la distribución original permitiendo simular múltiples *batches* y estimar el error de los resultados en la simulación, como mencionamos en la Sección 2.3.1. A su vez, se acumulan los resultados en cada *batch*, permitiendo iniciar una “nueva simulación”, lo que reduce el uso de memoria RAM. Los detalles de esta implementación se encuentran documentados en el Apéndice A.2.

Continuando con el análisis, KDSource utiliza funciones nativas del formato MCPL para leer la lista de partículas original de forma secuencial, asegurando que todas ellas sean utilizadas, sin perturbar la distribución en el espacio de fases de la misma. El número de partículas muestreadas puede no coincidir con un múltiplo exacto del número de partículas en el archivo MCPL original, pero no afecta la distribución estimada si partimos de la premisa de que el registro de *tracks* no tiene ninguna distribución preferencial.

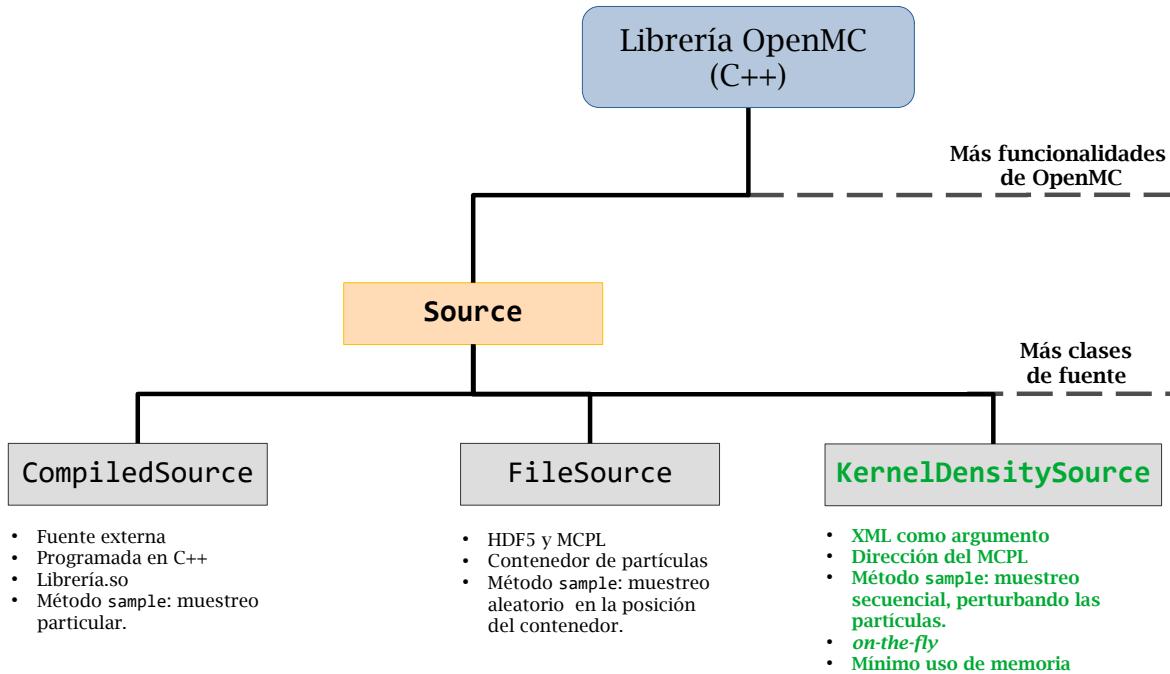


Figura 3.3: Incorporación del tipo de fuente `KernelDensitySource`, que hereda la clase madre `source`.

En la Figura 3.3 se resume la implementación de la clase `KernelDensitySource` descrita anteriormente, con un muestreo *on-the-fly* y un uso de memoria RAM reducido, como se demostrará en el capítulo siguiente.

3.3. Paralelización

La solución presentada en la sección anterior fue diseñada considerando la ejecución en un solo hilo, lo que resultó en grandes dificultades para realizar simulaciones *multithreading*. Aunque OpenMC es capaz de ejecutarse en paralelo utilizando OpenMP, `KDSource` actualmente no es compatible con este tipo de paralelización, lo que genera numerosos desafíos para evitar conflictos de memoria cuando múltiples hilos acceden a direcciones de memoria de manera concurrente. En esta sección se aborda el desarrollo de una versión capaz de ejecutarse en paralelo, con el objetivo adicional de garantizar la reproducibilidad de los resultados entre dos simulaciones bajo las mismas configuraciones.

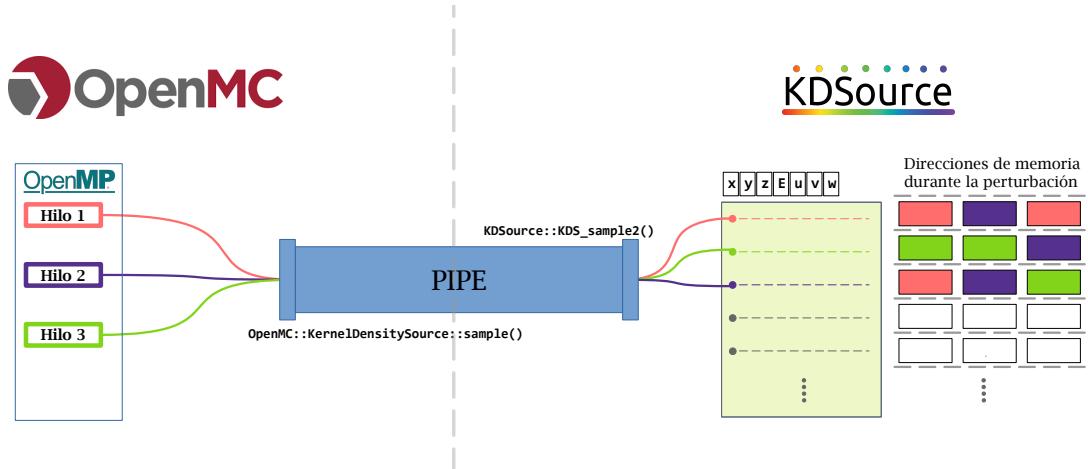


Figura 3.4: Esquema representativo del funcionamiento no controlado del muestreo de partículas al utilizar varios hilos en paralelo. De un lado se encuentra el código de `OpenMC` y del otro lado el código de `KDSource` al cual múltiples hilos acceden a la vez, sobrescribiendo direcciones de memoria comunes.

3.3.1. Desafíos encontrados e implementación

La implementación actual de `KDSource` no es adecuada para ejecutarse en paralelo, principalmente porque lee secuencialmente un archivo MCPL, manteniendo siempre la referencia de alguna partícula. Además, toma decisiones basadas en esta referencia, operación que debe realizarse de manera única por hilo para evitar comportamientos no controlados. En la Figura 3.4 se ilustra la incapacidad de la versión presentada en la sección anterior, de soportar una simulación *multithreading*. En la misma se destaca que el intercambio de datos entre ambos códigos durante la simulación se realiza mediante los métodos `sample()` de la clase `KernelDensitySource` y `KDS_sample2()`. Al utilizar varios hilos simultáneamente, diferentes hilos solicitan partículas perturbadas de manera concurrente y utilizan funcionalidades del código de `KDSource`, lo que produce un manejo no controlado de múltiples direcciones de memoria del código fuente de `KDSource`.

La solución aplicada a este problema es marcar todo este bloque de código como un bloque crítico para `OpenMP` utilizando la directiva `#pragma omp critical`. Esta le indica al compilador, que dicho bloque debe ser ejecutado por un hilo a la vez. En la figura 3.5, se representa que el compilador entiende esta directiva, sin generar problemas en el manejo de variables en el código `KDSource`. Aunque esta solución puede reducir la eficiencia, los resultados son aceptables, ya que se logra disminuir significativamente el tiempo de cálculo, como se mostrará en el siguiente capítulo.

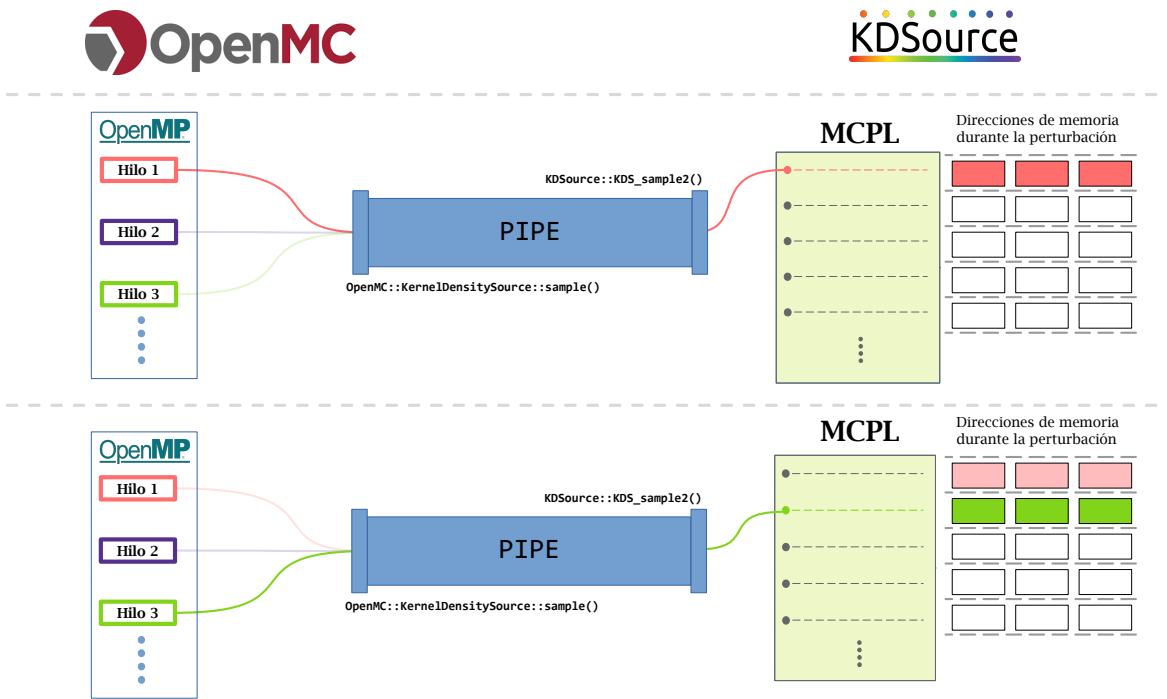


Figura 3.5: Esquema representativo del funcionamiento en simultáneo de KDSOURCE y OpenMC, en una simulación utilizando varios hilos en paralelo. De un lado se encuentra el código de OpenMC y del otro lado el código de KDSOURCE al cual los hilos acceden asincrónicamente.

Para introducir en el desarrollo de la reproducibilidad de los resultados, se presenta la Tabla 3.1 con un ejemplo de cómo OpenMC asigna semillas, para su generador de números aleatorios, al inicializar la historia de cada partícula. Al inicio de la simulación, las partículas se distribuyen de manera equitativa entre los hilos disponibles. En casos donde el número de partículas no es un múltiplo del número de hilos en paralelo, las x partículas restantes se asignan a los primeros x hilos. Además, se garantiza que las semillas recibidas por cada hilo se repitan en el mismo orden entre simulaciones, lo cual es un aspecto muy importante, ya que establece una fuerte relación entre las secuencias de números aleatorios de OpenMC y sus *threads*. Adicionalmente, se encontró que las semillas son asignadas a cada *thread* de modo que, si son ordenadas secuencialmente por el número del *thread* (N^{th}) mantengan el mismo orden, independientemente de la cantidad de ellos. Finalmente, en caso de ejecutar la simulación en *batches*, se repite este proceso de asignación de partículas y números aleatorios para cada *batch*.

	1 thread 2 batches		2 threads 1 batch		2 threads 2 batches	
Semilla	Nºth	Nºb	Nºth	Nºb	Nºth	Nºb
5.49757E+018	1	1	1	1	1	1
4.00365E+018	1	1	1	1	1	1
1.713827E+019	1	1	1	1	2	1
1.757456E+019	1	2	2	1	1	2
7.50168E+018	1	2	2	1	1	2
1.316815E+019	1	2	2	1	2	2

Tabla 3.1: Ejemplo de asignación de semillas de OpenMC para distintas configuraciones de hilos y batchs, para una misma cantidad de partículas totales.

Es necesario considerar que existen dos generadores de números aleatorios en simultáneo, y que para garantizar resultados repetibles, se debe lograr reproducibilidad tanto en el muestreo de las partículas por `KDSouce` como en el transporte por `OpenMC`.

En el mismo sentido, se observa que cuando el archivo MCPL es leído secuencialmente, no siempre se le asigna la misma partícula al mismo hilo, lo que conduce a la utilización de diferentes números aleatorios en la simulación. Sin embargo, se destaca que la perturbación de las partículas es reproducible, gracias a la forma de lectura y a que el generador de números aleatorios de `KDSouce` es único, proporcionando la misma secuencia.

A partir de la relación mencionada entre las semillas de `OpenMC` y los *threads* utilizados, surge la motivación de un manejo adicional de las partículas recibidas por cada hilo, con el objetivo de garantizar que siempre reciban la misma secuencia de números aleatorios. La solución propuesta consiste en implementar un constructor para la clase `KernelDensitySource` que depende fuertemente del número de *threads*. Se establece un desplazamiento con respecto a la posición en la lista de partículas para cada hilo, de manera que siempre maneja las mismas partículas, inicializando sus historias con las mismas semillas entre diferentes simulaciones. Este constructor se encuentra plasmado en el Apéndice A.3 y es parte de la estrategia utilizada para resolver la reproducibilidad de los resultados.

Este enfoque plantea un orden distinto en la lectura del archivo MCPL, dependiente del número de hilos en paralelo y que no tiene por qué ser el mismo entre dos simulaciones, ya que estos no se encuentran coordinados. En este sentido, para no perder reproducibilidad en el muestreo de la fuente, se implementa la posibilidad de inicializar el generador de números aleatorios de `KDSouce` al muestrear una partícula, con la misma semilla que se inicializará su historia en la simulación.

De esta forma se permite establecer una relación entre los números aleatorios que ve una partícula tanto al ser perturbada como en su transporte, garantizando la repro-

ducibilidad de los resultados, como se verá en el próximo capítulo. El detalle de esta versión está contenido en el apéndice mencionado anteriormente.

Cabe destacar que, `KDSource` tiene la capacidad de aplicar un método de estimación KDE adaptativo, para el cual necesita generar una lista de anchos de banda que se relaciona con el archivo `MCPL` original. A esta se le aplica el mismo algoritmo de desplazamiento por *thread*, para asignar los anchos de banda correspondientes.

Capítulo 4

Verificación continua del desarrollo

En el presente capítulo se presentan las pruebas realizadas durante el desarrollo de la técnica de muestreo *on-the-fly*, con el fin de realizar una verificación continua de las implementaciones. Se llevaron a cabo dos tipos de pruebas, por un lado se realizó un análisis del muestreo en vacío de una fuente del tipo `KernelDensitySource`, y por otro, se evaluó su desempeño durante la simulación, comparándolo con la rutina de muestreo tradicional a partir de una fuente del tipo `FileSource`.

4.1. Prueba en vacío

Para poder analizar el muestreo de la fuente, por simplicidad se optó por comenzar con simulaciones en vacío. Al no haber materia con la que las partículas pudieran interactuar, se facilita el análisis de la implementación. El objetivo principal fue verificar que el muestreo se lleva a cabo de forma correcta, para lo cual es necesario que las partículas recolectadas en la simulación sean correlacionables con la fuente.

Esta validación consiste en un tubo de vacío que en un extremo contiene una fuente plana, monoenergética, monodireccional y colineal con el eje del tubo, y espacialmente uniforme. En la Figura 4.1 se presenta la geometría del *test*, donde se define una superficie de *tracks* en una posición arbitraria, realizando una segunda simulación desde dicha superficie con una fuente del tipo `KernelDensitySource`. Se analizó el muestreo tanto conservando las partículas originales, como aplicándoles perturbaciones.

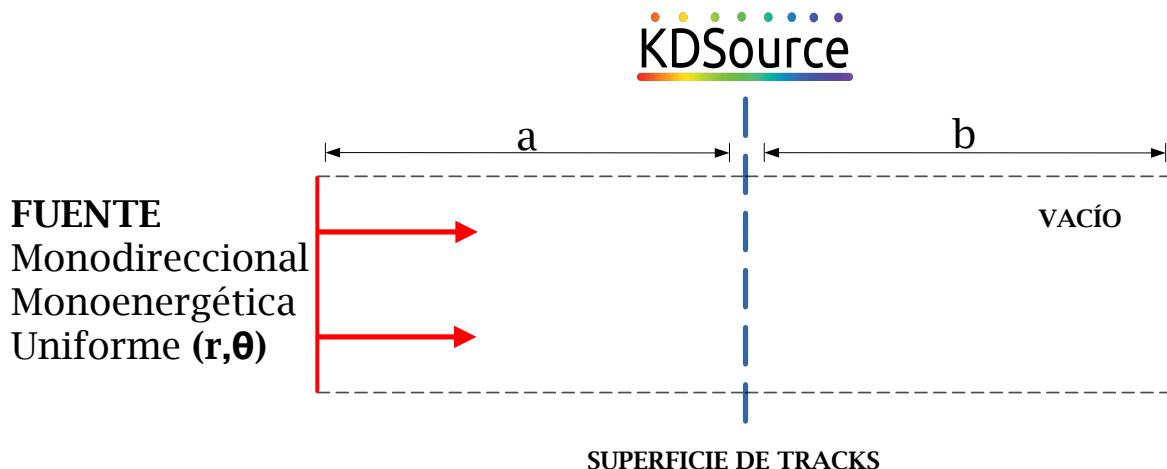


Figura 4.1: Geometría de la prueba en vacío. En rojo se tiene la fuente original y en líneas punteadas azules la superficie de *tracks*, donde luego se configura una fuente de KDSOURCE.

Este *test* permite desacoplar el desempeño en la generación de partículas del resto de la simulación. En la misma imagen se puede apreciar que la primera simulación se utiliza para generar una lista de partículas bien conocida. Con la misma se analizó el funcionamiento de KDSOURCE, dentro de una simulación de OpenMC, obteniéndose lo siguiente:

- Inicialmente, se verificó que la fuente utilizara todas las partículas del archivo original, como lo hace el algoritmo de muestreo tradicional de KDSOURCE al generar una lista. Para esto, se creó una cantidad conocida de neutrones desde la fuente original, aplicando una instancia de KernelDensitySource sin perturbaciones ni ajustes de pesos, con las partículas que lleguen a la superficie intermedia. Se simuló la misma cantidad de partículas desde esta fuente, corroborando que no se registrara ninguna repetida.
- Con esta misma configuración, se construyó la Tabla 3.1 del capítulo anterior, que permitió asociar cada partícula con una semilla, el hilo que la iba a simular y en qué *batch*. Esto se pudo lograr gracias a la facilidad de seguir partícula por partícula en la simulación en vacío.
- Posteriormente, se construyó la fuente mencionada, configurada para que las variables espaciales fueran las únicas perturbadas, manteniendo así el análisis enfocado en una sola métrica que las contenga. Se realizó una simulación con la fuente perturbada, siguiendo el esquema presentado en la Figura 4.2, corroborando que la fuente efectivamente es capaz de generar partículas nuevas, concurrentemente a la simulación.

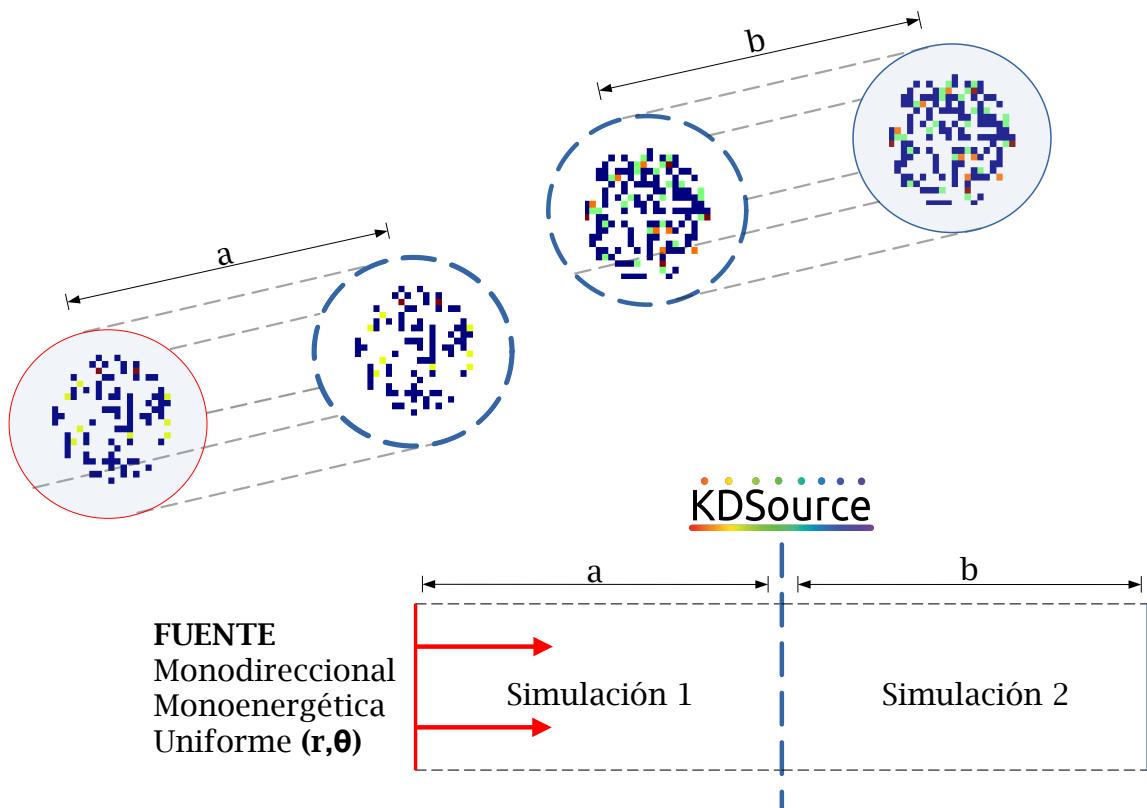


Figura 4.2: Esquema tridimensional de las simulaciones en vacío. En rojo se tiene la fuente original utilizada y en líneas punteadas azules la superficie de *tracks*, donde luego se utiliza una fuente de KDSource. Se observa un ejemplo donde se generan partículas conocidas (simulación 1), y se crean nuevas a partir de su perturbación (simulación 2), las cuales son fácilmente analizables al final del conducto.

4.2. Prueba en conducto de agua liviana

Se realizaron otras simulaciones con el objetivo de validar el funcionamiento de la herramienta en una configuración sencilla, pero más realista que el vacío. Se diseñó un tubo de sección cuadrada lleno de agua liviana, eligiendo este material debido a la dificultad que presenta para generar estadísticas en zonas alejadas de la fuente. Se utilizó una fuente de neutrones plana, monoenergética de 3 MeV, isotrópica y espacialmente uniforme. En la Figura 4.3 se muestra la geometría mencionada, en la que se colocó una superficie de *tracks* a 30 cm, donde se registraron 10^6 de 10^8 neutrones simulados. Esto permitió evaluar nuestra implementación, aplicando KDSource en esta superficie, con una perturbación en todas las variables del espacio de fases.

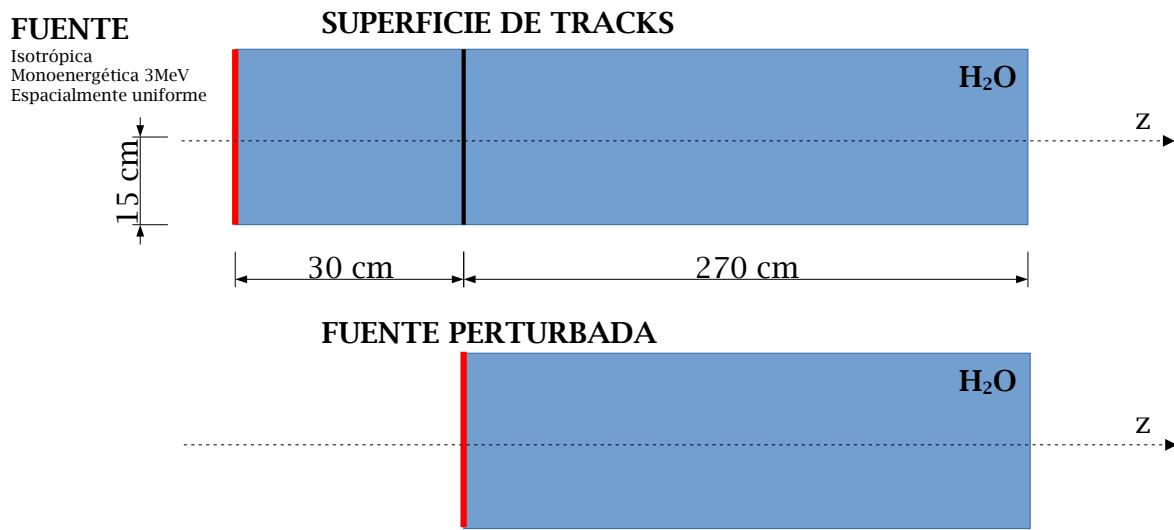


Figura 4.3: Geometría del conducto de agua utilizado en las simulaciones. En rojo se presentan las fuentes utilizadas.

Se comparó el muestreo de partículas *on-the-fly* con el muestreo tradicional, para una misma cantidad de partículas generadas a partir del mismo archivo MCPL, con los mismos anchos de banda y función *kernel*.

En la Figura 4.4 se presentan los mapas de flujo correspondientes a la simulación original y al muestreo de 10^8 neutrones por ambas rutinas de KDSOURCE. Se observa que, a pesar de aumentar en dos órdenes de magnitud la cantidad de partículas, es difícil transportar neutrones a zonas de baja estadística. Sin embargo, junto con la Figura 4.5, se puede ver que ambas rutinas de muestreo de KDSOURCE proporcionan resultados similares hasta una distancia de aproximadamente 70 cm, a partir de la cual, la diferencia aumenta. Esto es atribuible a la baja estadística y a que no se utilizan las mismas partículas en cada simulación. OpenMC sortea las partículas de manera aleatoria cuando utiliza una fuente a partir de una lista de partículas, que es el caso de la rutina tradicional; mientras que, en la rutina *on-the-fly*, se leen secuencialmente. A su vez, en esta última las partículas son perturbadas con números aleatorios inicializados desde OpenMC en cada muestreo, mientras que tradicionalmente el generador de KDSOURCE se inicializa una sola vez.

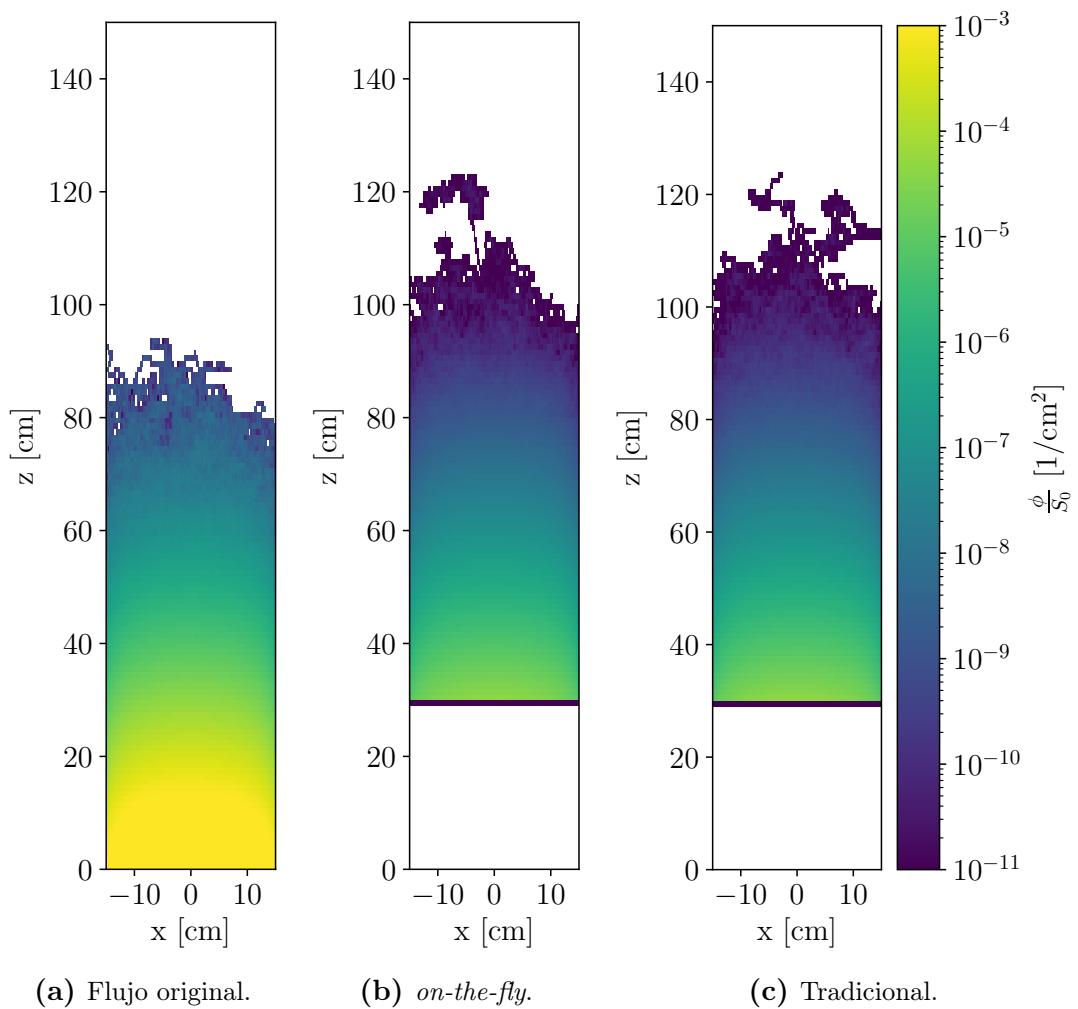


Figura 4.4: Distribución espacial x-z del flujo neutrónico, normalizado por neutrón de fuente. En la Figura (a) se observa la distribución a partir de la fuente original, mientras que en las figuras (b) y (c) se presentan los flujos correspondientes al muestreo *on-the-fly* y tradicional, respectivamente.

Se realizó un análisis del comportamiento en la variable energética a partir de la distribución de la corriente parcial de neutrones en una superficie de *tracks*. La misma se ilustra en la Figura 4.6, donde ambas rutinas de muestreo presentan resultados similares. A bajas energías se ve una discrepancia entre los resultados, pero nuevamente es atribuible a que es una zona en el espacio de fases de escasa estadística y el muestreo por ambas rutinas se realiza con diferentes números aleatorios.

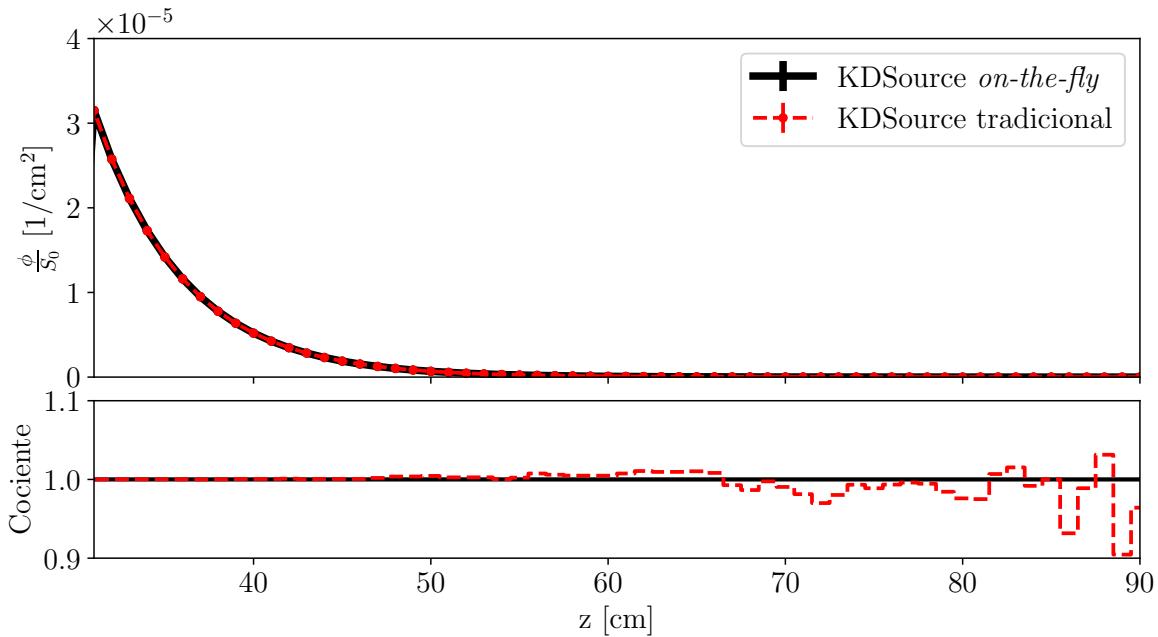


Figura 4.5: Flujo neutrónico en función de la distancia z a la fuente, normalizado por neutrón de la fuente original. Se presentan solapadas las curvas para ambas rutinas de muestreo. Debajo se muestra la relación entre ambas expresiones. El error estadístico se encuentra inmerso en la curva.

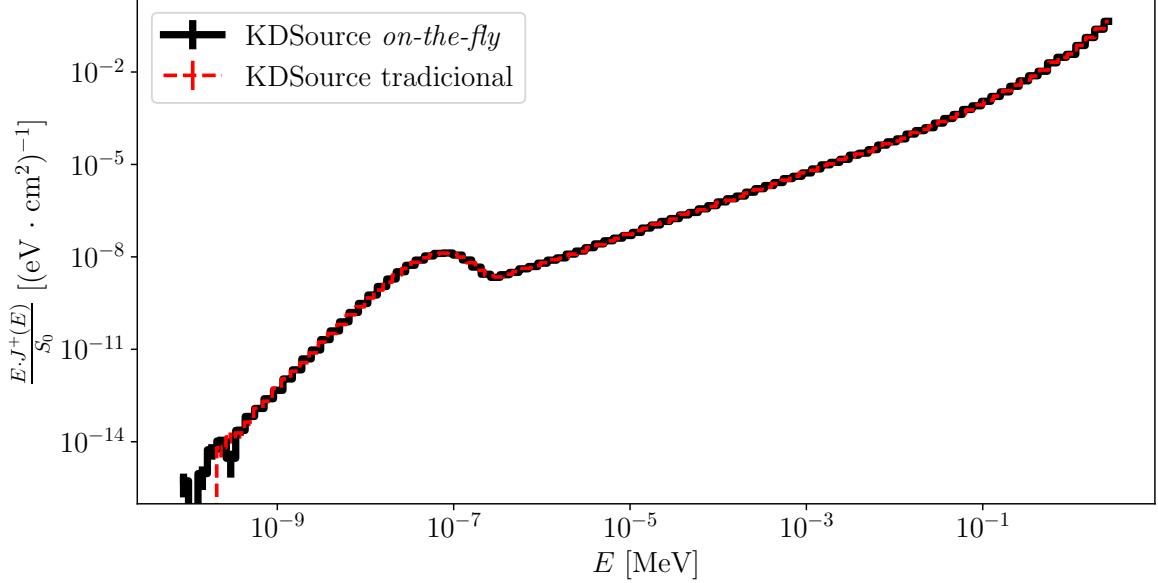


Figura 4.6: Corriente de neutrones en la dirección positiva del eje z , normalizada por neutrón de fuente y multiplicada por su energía. Se presentan solapadas las curvas para ambas rutinas de muestreo. La misma corresponde a una superficie de *tracks* en la posición $z = 60$ cm. El error estadístico se encuentra inmerso en la curva.

Luego se evaluó el desempeño de la fuente `KernelDensitySource` con relación al uso de memoria durante la simulación, cuyos resultados se encuentran plasmados en la Tabla 4.1. Se observan resultados prometedores, ya que al utilizar la técnica de muestreo *on-the-fly*, el uso de memoria RAM máximo durante una simulación es menor

al 1% independientemente del número de partículas. El mismo fue monitoreado con la herramienta `top` nativa de los sistemas Linux. Por otro lado, se tiene que para la rutina tradicional existe un límite del orden de 10^7 partículas, un número considerablemente bajo si hablamos, por ejemplo, del cálculo de blindajes. Con esta implementación ya no existe una limitación en cuanto a la cantidad máxima de partículas a simular, debido al tamaño máximo de una lista de partículas que se pueda cargar en memoria. Ahora la limitación estará determinada por los tiempos, la capacidad de cómputo y los recursos que use OpenMC en su funcionamiento normal; y no por el manejo ineficiente de memoria del mismo al leer una lista de *tracks*.

% Memoria RAM utilizada de 8 Gb		
Nº partículas	<i>on-the-fly</i>	TRADICIONAL
10^6	<1 %	1,3 %
10^7	<1 %	13 %
10^8	<1 %	130 %

Tabla 4.1: Máximo uso de memoria RAM durante la simulación en el mismo caso de estudio. No se registran *tallies* o superficie de *tracks* que puedan utilizar memoria.

Por último, se realizó una prueba de la paralelización implementada con el objetivo de analizar los tiempos de cálculo y compararlos con los tiempos tradicionales para una cantidad de partículas manejable en listas de partículas. Además, se evaluó la reproducibilidad de los resultados al aplicar perturbaciones en todas las variables del espacio de fases.

Tiempo de simulación [min]						
Rutina	<i>on-the-fly</i>			TRADICIONAL		
Nº partículas\Nº Threads	2	4	8	2	4	8
10^6	0,47	0,33	0,25	0,43	0,31	0,23
10^7	4,58	3,22	2,43	4,05	2,85	2,06
10^8	60,25	34,68	27,63	54,78	31,90	25,37

Tabla 4.2: Tiempo de simulación en función del número de partículas y el número de hilos en paralelo, utilizando un cpu de 8 núcleos y solo transportando neutrones. No se tiene en cuenta el tiempo asociado a la generación de la lista de partículas perturbadas al usar la rutina tradicional.

La Tabla 4.2 presenta resultados muy interesantes en cuanto a la paralelización de la rutina. Se puede observar que el tiempo de simulación disminuye en función del número de hilos en paralelo, de forma similar en ambas rutinas y que los tiempos de simulación son comparables. Si bien tienden a ser un poco mayores para el muestreo *on-the-fly*, se podría asociar esto al tiempo necesario para leer un archivo MCPL y realizar operaciones

sobre partículas en cada inicialización (es de esperar que sea significativamente mayor al tiempo de leer partículas ya en memoria como en la rutina tradicional). Sin embargo, esto no resulta ser así, probablemente debido a que el tiempo de muestreo es considerablemente menor al tiempo de transporte de las partículas. En este ejemplo, esto se puede ver incrementado debido a que los neutrones tienen mucha interacción con el medio. Adicionalmente, en la Tabla 4.2 no se tiene en cuenta el tiempo de generación de la lista de partículas perturbadas en la rutina tradicional, que es considerablemente mayor que la diferencia entre los tiempos de simulación presentados. Para el caso de una lista de 10^8 partículas, este tiempo fue de aproximadamente 5 minutos.

Asimismo, se pudo analizar la reproducibilidad de los resultados al aplicar la rutina *on-the-fly*. El estudio se basó en obtener exactamente la misma distribución del flujo escalar entre simulaciones, en función de las variables espaciales x (ancho del tubo) y z (distancia a la fuente). Se simularon 10^6 neutrones utilizando el método KDE adaptativo, con el mismo archivo MCPL, la misma función *kernel* y el mismo archivo de anchos de banda. Se analizaron cambios en la distribución, variando el número de *threads*, el número de *batches* y sin producir modificaciones en ellos. Los resultados se expresan en la Tabla 4.3. Se observa que conservando los pesos de la lista de partículas original, se consiguen resultados reproducibles independientemente del número de *batches* y el número de hilos en paralelo. Para el caso en que se utilice el algoritmo de ajuste de pesos de KDSource, los resultados solo son repetibles si no se introducen modificaciones, es decir, para un número determinado de *batches* y *threads*. Esto se debe a que el constructor de la clase utilizada genera desplazamientos en la posición de lectura de la lista de partículas para cada hilo, produciendo una simulación desacoplada por cada uno. A su vez, al aplicar el algoritmo de ajuste de los pesos mencionado en la Sección 3.2.1, se avanza en la lista de partículas en función de la probabilidad de tomar o no cada una, desplazamiento que no es tenido en cuenta en el constructor. Esto produce que los resultados dependan de la cantidad de *threads* usados y sus posiciones iniciales. Por este mismo motivo, al dividir la simulación en *batches*, se obtienen resultados distintos, ya que cambian las posiciones iniciales de lectura de la lista de partículas.

Reproducibilidad de los resultados entre simulaciones		
Modificaciones	Conserva los pesos	Ajusta los pesos
Sin modificaciones	SÍ	SÍ
Diferentes <i>batches</i>	SÍ	NO
Diferentes <i>threads</i>	SÍ	NO

Tabla 4.3: Reproducibilidad de los resultados para 10^6 neutrones entre simulaciones, en función del algoritmo de muestreo.

Capítulo 5

Haz de neutrografía del reactor RA-6

En este capítulo se discuten los resultados de implementar `KDSource` *on-the-fly* dentro de `OpenMC` a partir de una simulación de transporte de fotones y neutrones el haz de neutrografía del reactor RA-6 [23], ubicado en el Centro Atómico Bariloche. Este es un reactor experimental con distintos dispositivos de irradiación y una potencia de 1 MW. Sin embargo, para irradiaciones en el conducto de interés (N°1) se alcanza la mitad de la potencia.

Se eligió este ejemplo de aplicación, ya que en proyectos anteriores relacionados con desarrollos en `KDSource`, también se utilizó esta configuración, permitiéndonos comparar la implementación del muestreo *on-the-fly* [8, 9].

5.1. Descripción del instrumento de irradiación

Se utilizó un modelo tridimensional de la herramienta en `OpenMC` como ejemplo de aplicación, con una fuente a partir de una lista de partículas obtenida en simulación Monte Carlo del núcleo. En la Figura 5.1 se ilustra un corte axial de la geometría simulada, a partir de un plano de simetría. La misma fue construida a partir de los planos del documento [23] durante el desarrollo de un proyecto previo [8]. En la parte inferior se encuentra la pileta del reactor contenida por una pared de hormigón en la cual el conducto está inmerso. Este va desde la pileta del reactor hasta la entrada del recinto de irradiación, pasando a través de colimadores. En una primera instancia se encuentra la cámara interna que, además de colimar, posee un filtro de zafiro para reducir la contribución de radiación γ y neutrones rápidos. Luego se encuentra la cámara externa, que es un segundo colimador con forma cónica, la cual genera la forma final del haz. Este está colimado hacia un *beam catcher* de hormigón, pasando a través de un recinto de irradiación blindado con políboro y una capa de plomo. Esta configuración funciona como blindaje del haz, tanto para las partículas que lleguen al recinto, como para aquellos fotones secundarios que se generen en el blindaje de neutrones. A su vez,

existe un *shutter* que consta de un dispositivo móvil para blindar el haz cuando no es utilizado.

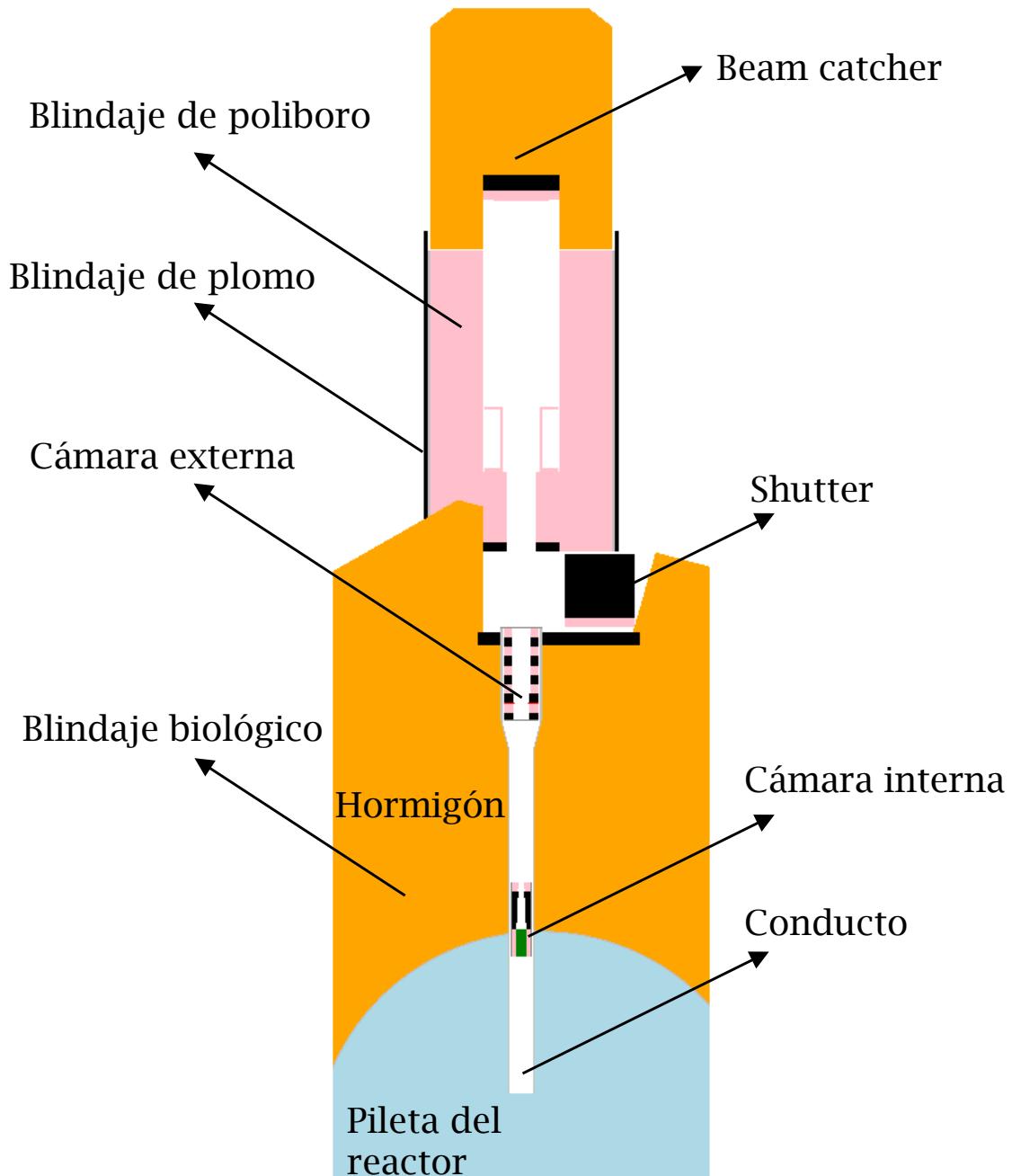


Figura 5.1: Geometría del conducto de neutrografía del reactor RA-6, utilizada en las simulaciones.

5.2. Utilización de KDSource *on-the-fly*

En este estudio, se utiliza KDSource con el objetivo de aumentar la estadística en el recinto próximo al conducto de neutrografía, para obtener un mapa de dosis e intensidad de flujo escalar de neutrones y fotones. Se generaron dos simulaciones com-

plementarias, a partir de una lista de *tracks* compuesta por ambos tipos de partículas. La misma se obtuvo con una simulación de $N_{\text{sim}}^0 = 10^9$ neutrones, del tipo de cálculo k_{efectivo} del núcleo con la configuración de barras correspondiente al instrumento de neutrografía. Se agrupó cada tipo de partícula en distintas fuentes, ya que naturalmente presentan distribuciones en el espacio de fases distintas. En la Figura 5.2 se puede ver la geometría utilizada para la primera simulación a partir de la superficie S1, un poco antes de que las partículas crucen la cámara interna. Se puede observar que la geometría original es recortada, colocando vacío en la zona inferior al plano, debido a que cualquier partícula que pueda cruzar la superficie en dirección al conducto ya está contenida en la distribución original de la fuente. En la posición S2 se colocó una segunda superficie de *tracks*, debido a que se vio la necesidad de aplicar nuevamente KDSource para conseguir un mapa de flujo convergido en toda la zona circundante, como se ejemplifica más adelante.

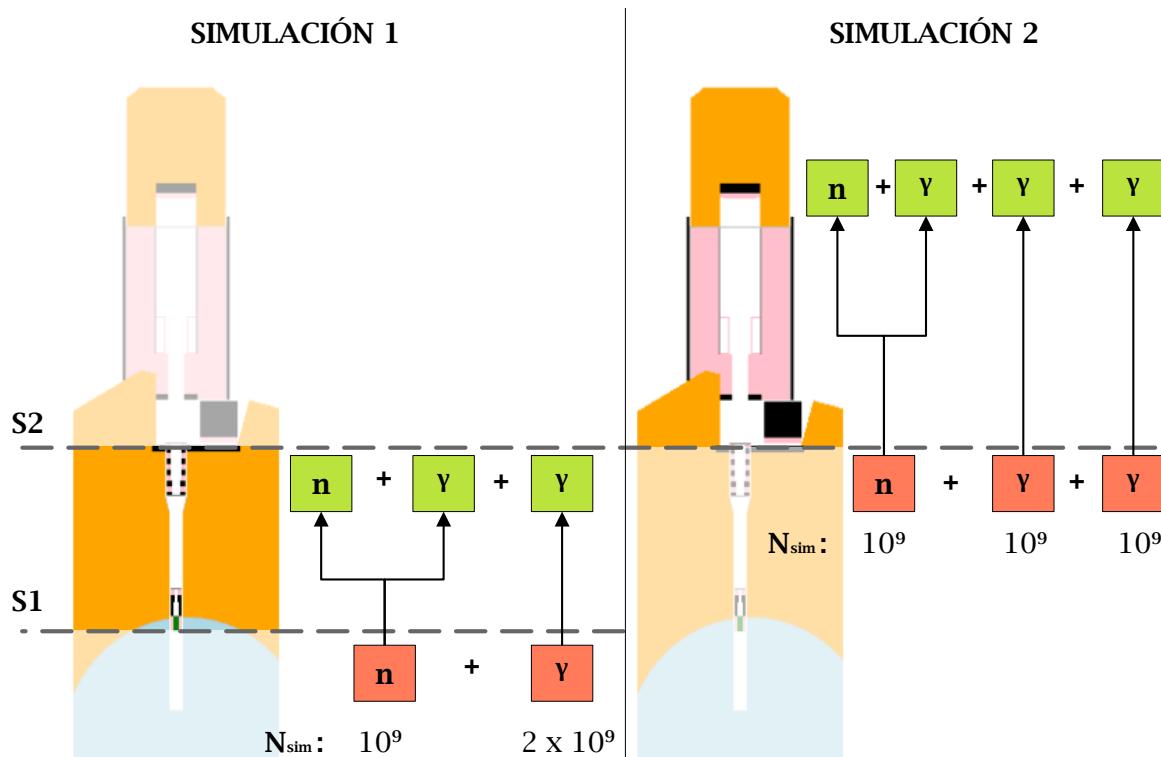


Figura 5.2: Diagrama de las simulaciones realizadas a partir de las superficies S1 y S2. En rojo se muestran las fuentes utilizadas con la cantidad de partículas simuladas N_{sim} . En verde se tienen las contribuciones a los resultados de cada simulación.

Por cada simulación de neutrones se suma una contribución de fotones que deben tenerse en cuenta para el resultado final. En la Figura 5.2 se ilustran las simulaciones realizadas para cada superficie de *tracks*, dejando al final tres contribuciones de fotones y una de neutrones. Como ahora la cantidad de partículas a simular no es una limitación en cuanto a uso de memoria, desde la superficie 1 se simuló una cantidad de neutrones y fotones que genere una lista de un orden de 10^6 partículas en la superficie 2, para

luego simular 10^9 por cada grupo. En la Tabla 5.1 se muestra la suma de los pesos de las partículas en cada lista y cuántas de ellas se simularon a partir de una fuente de KDSource con ella.

Número de partículas efectivas por superficie		
Superficie 1		
Lista de <i>tracks</i>	Registradas	Simuladas
Neutrones de fuente	42982	10^9
Fotones de fuente	121629	20^9
Superficie 2		
Lista de <i>tracks</i>	Registradas	Simuladas
Neutrones de fuente	1187797	10^9
Fotones de fuente	1316387	10^9
Fotones secundarios	36163	10^9

Tabla 5.1: Número de partículas efectivas en cada superficie, por cada lista de *tracks* utilizada. Para las partículas registradas se presenta la suma de los pesos.

5.3. Normalización y cálculo de factores de fuente

Debido a la aplicación de dos superficies de *tracks* intermedias, se deben normalizar los resultados para obtener una estimación correcta de la intensidad de flujo en el recinto. Para esto se calculó el factor de fuente que le corresponde a cada superficie, igualando las corrientes de partículas salientes en cada una. La misma se define como el factor de fuente por la proporción de partículas de interés que llegan a la superficie (en nuestro caso los neutrones o fotones en dirección saliente al reactor). En la Ecuación 5.1 se muestra el cálculo correspondiente a la superficie 1, donde se simularon $N_{sim}^0 = 10^9$ neutrones y w_i es el peso de la partícula i de las N_1 que llegaron a la superficie. A su vez, el factor de fuente es $S_0 = 3,87 \times 10^{16}$ n/s y corresponde a la grilla de núcleo N°16 del RA-6 a 0,5 MW de potencia y la configuración de barras correspondiente al conducto N°1. En la Tabla 5.1 se puede ver la suma total de los pesos de las partículas en la dirección de interés para cada lista de *tracks*.

$$J_{n,p}^+ \cdot A = S_0 \frac{\sum_{i=1}^{N_1} w_i}{N_{sim}^0} = S_1 \quad (5.1)$$

El factor de fuente de la nueva simulación a partir de la superficie S1, es la propia corriente en unidades de n/s, ya que, representa la tasa de neutrones que aportan a los *tallies* a partir de dicha superficie. En este mismo sentido, el factor de fuente correspondiente a la superficie S2, proveniente de igualar corrientes en ella, se define

en la Ecuación 5.2.

$$S_2 = S_0 \frac{\sum_{i=1}^{N_1} w_i}{N_{sim}^0} \frac{\sum_{j=1}^{N_2} w_j}{N_{sim}^1} \quad (5.2)$$

La herramienta `KDSouce` puede provocar que se suavicen algunas distribuciones que no lo son, o en el caso de un haz, donde existe una dirección preferencial de las partículas, puede no respetar correctamente esta distribución. Para ello, se utilizó el factor de fuente expresado en la Ecuación 5.3, que está afectado por este efecto.

$$S_1^* = S_0 \frac{\sum_{i=1}^{N_1} w_{i,\mu < x}}{N_{sim}^0} \frac{N_{KDS}}{N_{KDS}^{\mu < x}} \quad (5.3)$$

Ahora las partículas de interés son solo las que están colimadas con una dirección menor a cierto ángulo relacionado con $\mu = x$, determinado a partir de los colimadores, y se agrega un factor que es la inversa de la proporción de partículas que `KDSouce` muestrea con este mismo criterio de ángulo. En la Tabla 5.2 se muestra que esta normalización no afecta significativamente los resultados, produciendo una variación en el factor de fuente menor al 3,5 %, límite que corresponde al factor de fuente de *tracks* de fotones secundarios en la superficie 2, el cual es el que menor estadística posee. Este resultado demuestra que, a pesar de que la fuente de *tracks* esté colimada, `KDSouce` de todas formas respeta la distribución en el espacio de fases de las partículas.

Factor de fuente por superficie		
Superficie 1		
Lista de tracks	S1	S1*
Neutrones de fuente	$1,66 \cdot 10^{12}$	$1,67 \cdot 10^{12}$
Fotones de fuente	$4,70 \cdot 10^{12}$	$4,82 \cdot 10^{12}$
Superficie 2		
Lista de tracks	S1	S1*
Neutrones de fuente	$2,03 \cdot 10^9$	$1,99 \cdot 10^9$
Fotones de fuente	$3,09 \cdot 10^9$	$3,17 \cdot 10^9$
Fotones secundarios	$6,01 \cdot 10^7$	$5,80 \cdot 10^7$

Tabla 5.2: Factor de fuente en cada superficie, para cada lista de *tracks* utilizada. Los mismos son calculados mediante las ecuaciones 5.2 y 5.3.

5.4. Cálculo del flujo escalar de neutrones y fotones

Para el cálculo del flujo escalar se utilizó un *tally* de `OpenMC` que registra el flujo en la geometría y en el recinto cercano, luego se separaron los resultados por contribuciones

de fotones y neutrones. A continuación, se presentan los mapas de flujos de la geometría para las simulaciones a partir de la superficie S1. En la Figura 5.3a se ilustra el flujo de neutrones para una simulación con 10^7 partículas, que fue la máxima cantidad que se podía simular en la computadora utilizada con la rutina tradicional, mientras que en la Figura 5.3b se encuentra la misma simulación pero utilizando KDSource *on-the-fly* con 10^9 partículas. Se puede ver que aumenta considerablemente la estadística en algunas zonas para las que anteriormente no había ninguna traza de partículas. Lo mismo se observa con el flujo de fotones en la Figura 5.4.

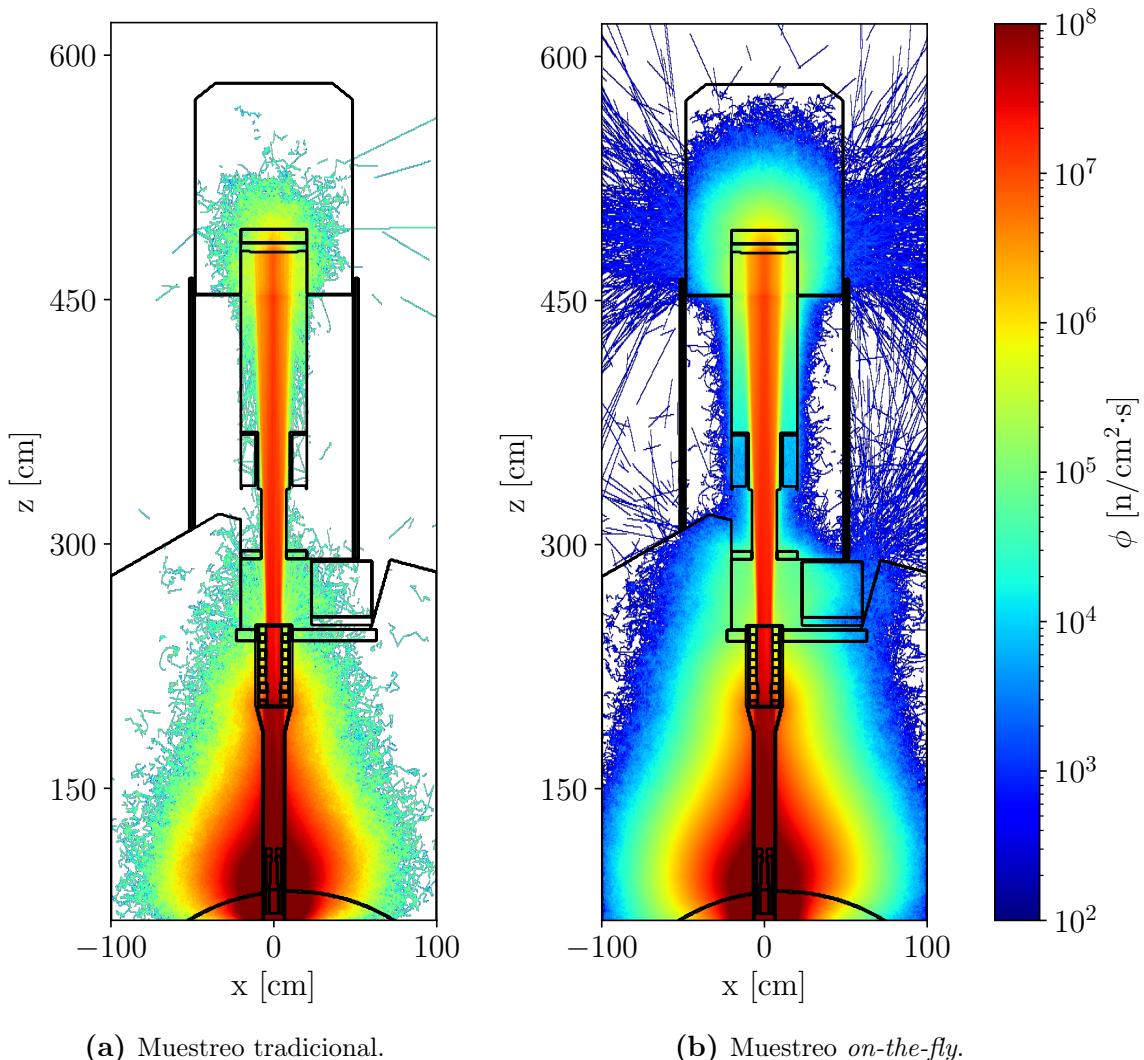


Figura 5.3: Flujo de neutrones utilizando KDSource a partir de la superficie S1. En la Figura (a) se utilizaron 10^7 partículas, mientras en la Figura (b) se simularon 10^9 .

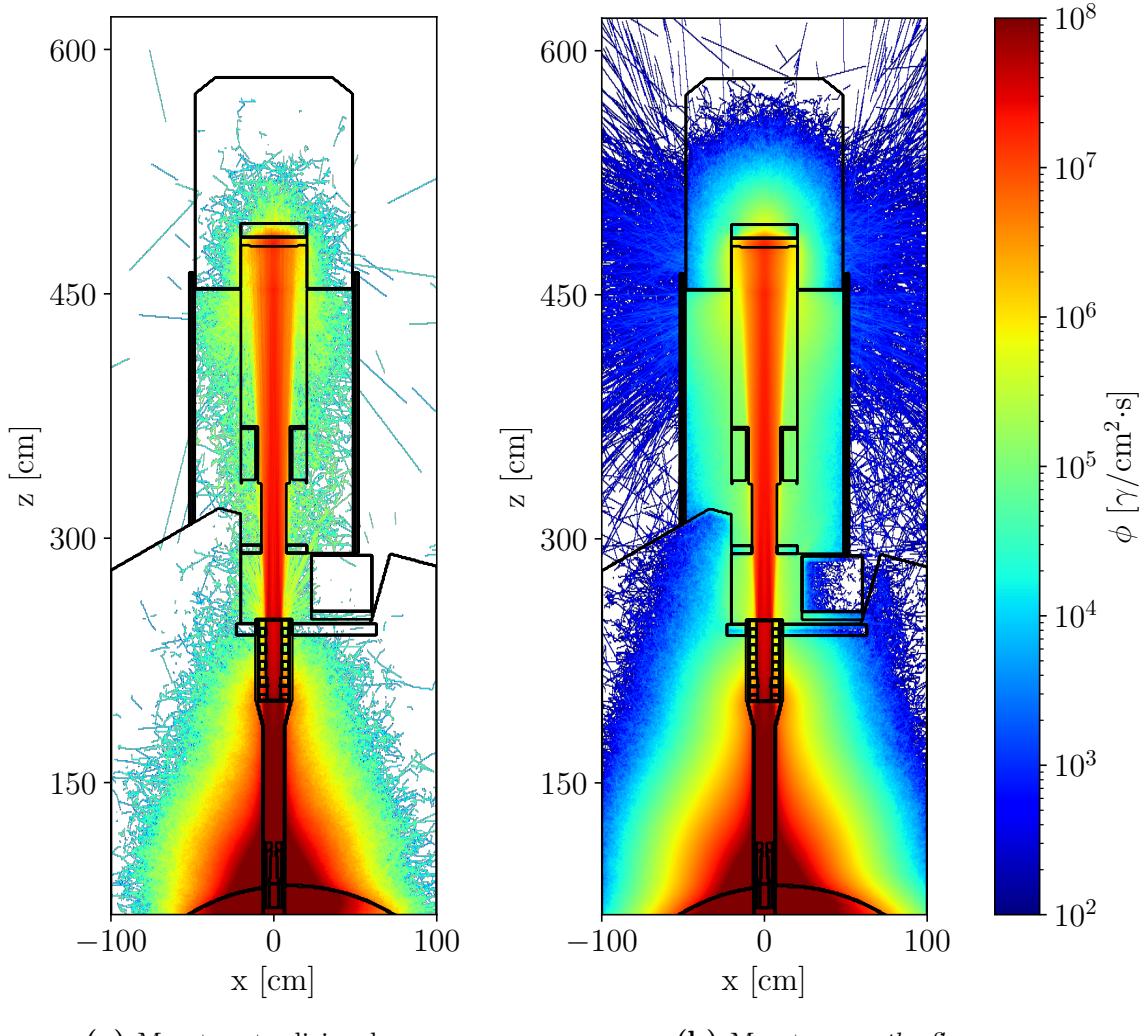


Figura 5.4: Flujo de fotones utilizando KDSOURCE a partir de la superficie S1. En la Figura (a) se utilizaron 10^7 partículas, mientras en la Figura (b) se simularon del orden de 10^9 .

Sin embargo, como se mencionó anteriormente, aparece la necesidad de colocar otra superficie más adelante para obtener *tallies* convergidos en todo el mapa. Por esto, se utilizó nuevamente una fuente de KDSOURCE en la posición S2. En las Figuras 5.5 y 5.6 se presentan los resultados de utilizar KDSOURCE en esa posición, donde nuevamente se consigue una mejor estadística con la rutina *on-the-fly*. Puede notarse que con 10^7 partículas aún en algunas zonas complicadas, como los puntos pegados al blindaje de poliboro (ver Figura 5.1), no se consigue estadística suficiente. Utilizando KDSOURCE *on-the-fly* se corrió 100 veces dicha cantidad, sin la necesidad de realizar mayores acciones para la generación de la fuente, observándose que la estadística converge en todo el recinto del conducto. Se observa además que si bien el mapa de flujo en la Figura 5.5a parece estar convergido en algunas zonas, se presentan rayos discontinuos, dando indicios de una zona de baja estadística, aún donde existe una mayor concentración de partículas.

En un proyecto anterior [9], con la misma configuración de superficies de *tracks* y

geometrías, se pudo simular 10^8 partículas pero requirió de varias simulaciones consecutivas, acumulando resultados y con un significativo trabajo posterior para el procesamiento de los datos. Ahora, se resuelve nativamente desde OpenMC.

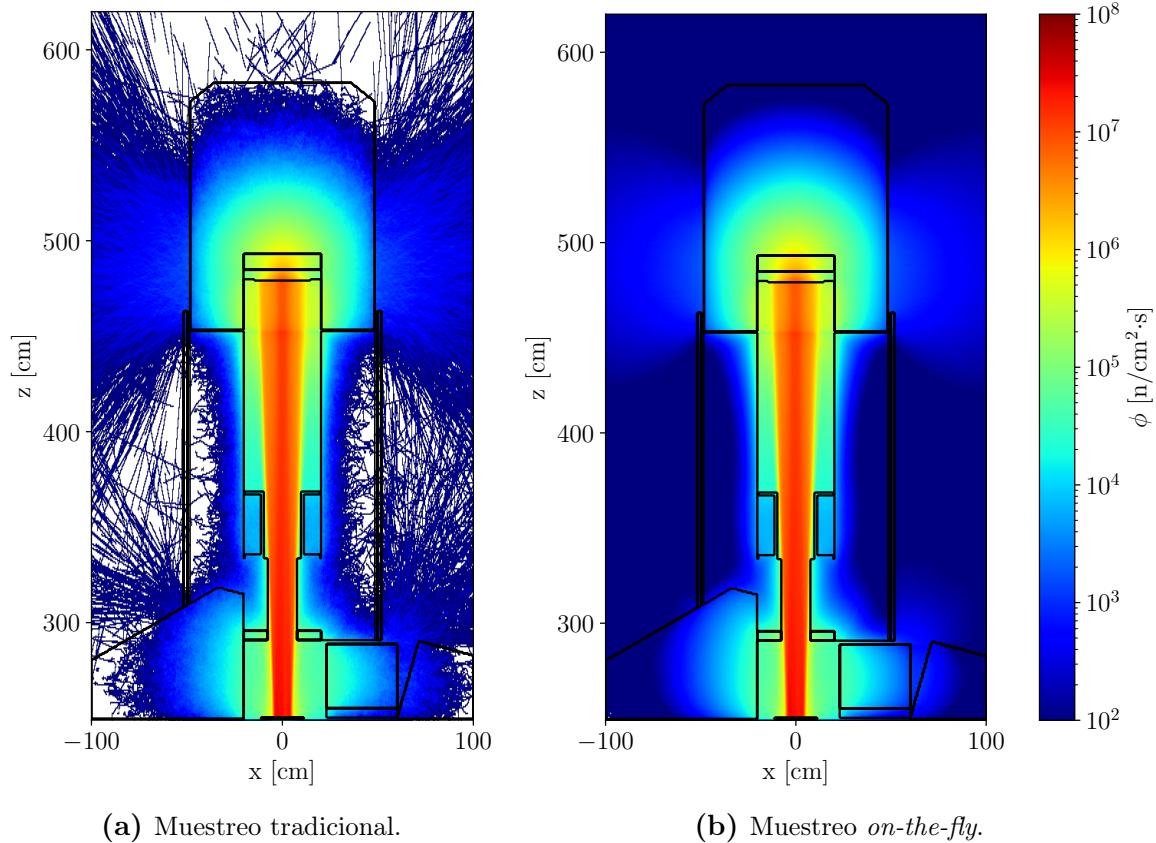


Figura 5.5: Flujo de neutrones utilizando KDSource a partir de la superficie S2. En la Figura (a) se utilizaron 10^7 partículas, mientras en la Figura (b) se simularon 10^9 .

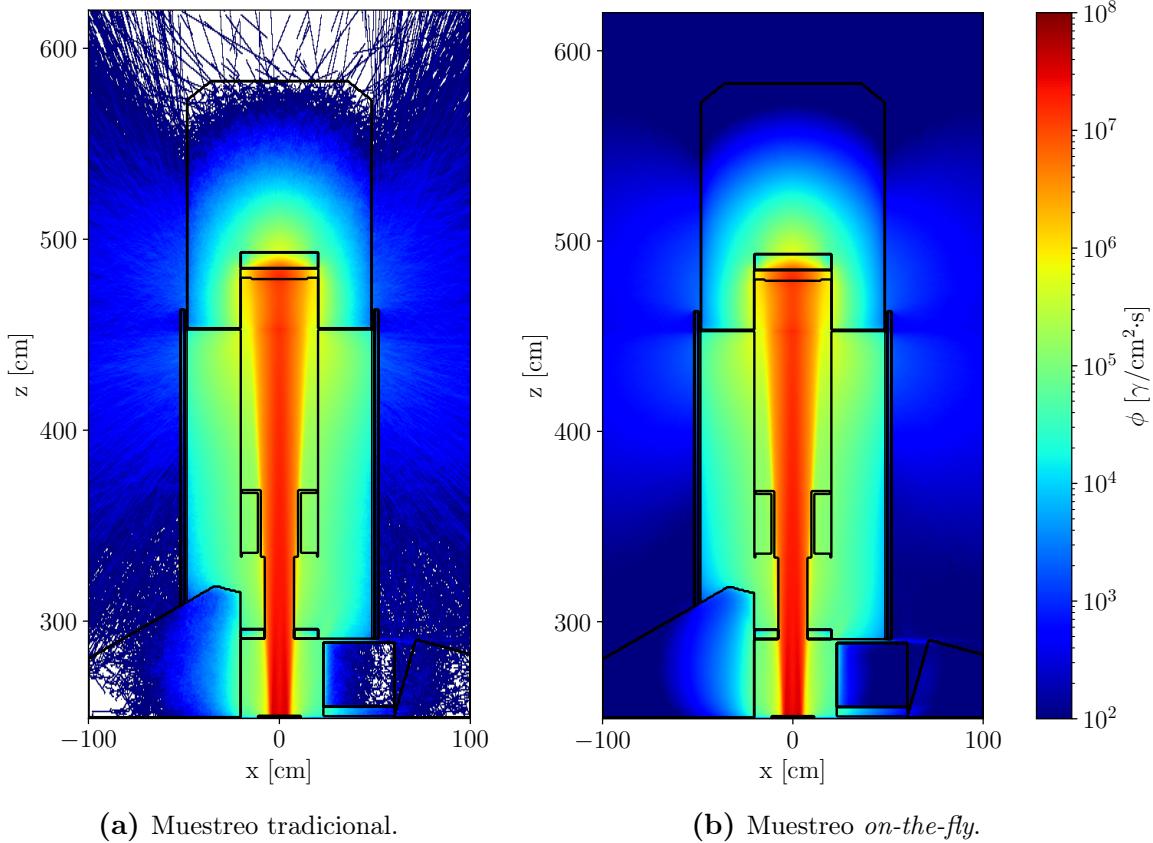


Figura 5.6: Flujo de fotones utilizando KDSource a partir de la superficie S2. En la Figura (a) se utilizaron 10^7 partículas, mientras en la Figura (b) se simularon en el orden de 10^9 fotones.

5.5. Cálculo de la dosis equivalente ambiental $H^*(10)$

Para el cálculo de la tasa de dosis ambiental se utilizó un tally de OpenMC de flujo, junto con un filtro de energía, con el cual se recolectó la dosis integrada para cada energía según los datos de $h^*(10, E)$ proporcionados por la ARN [24]. OpenMC posee una función nativa que permite recolectar esta magnitud integral, dado un archivo con factores de conversión de fluencia a dosis. Con el fin de comparar fácilmente el mapa de dosis con resultados experimentales, se utilizó un mallado con volúmenes comparables a los de los detectores utilizados por el personal de radioprotección del RA-6. En cada celda se registra entonces la magnitud expresada en la Ecuación 5.4:

$$H^*(10) = \int_{V_{cel}} \int_0^\infty \phi(E, V) h^*(10, E) dE dV \quad (5.4)$$

Se pueden ver los mapas de dosis en la figura 5.7, obtenidos a partir de las simulaciones desde la superficie S2 de la figura 5.2.

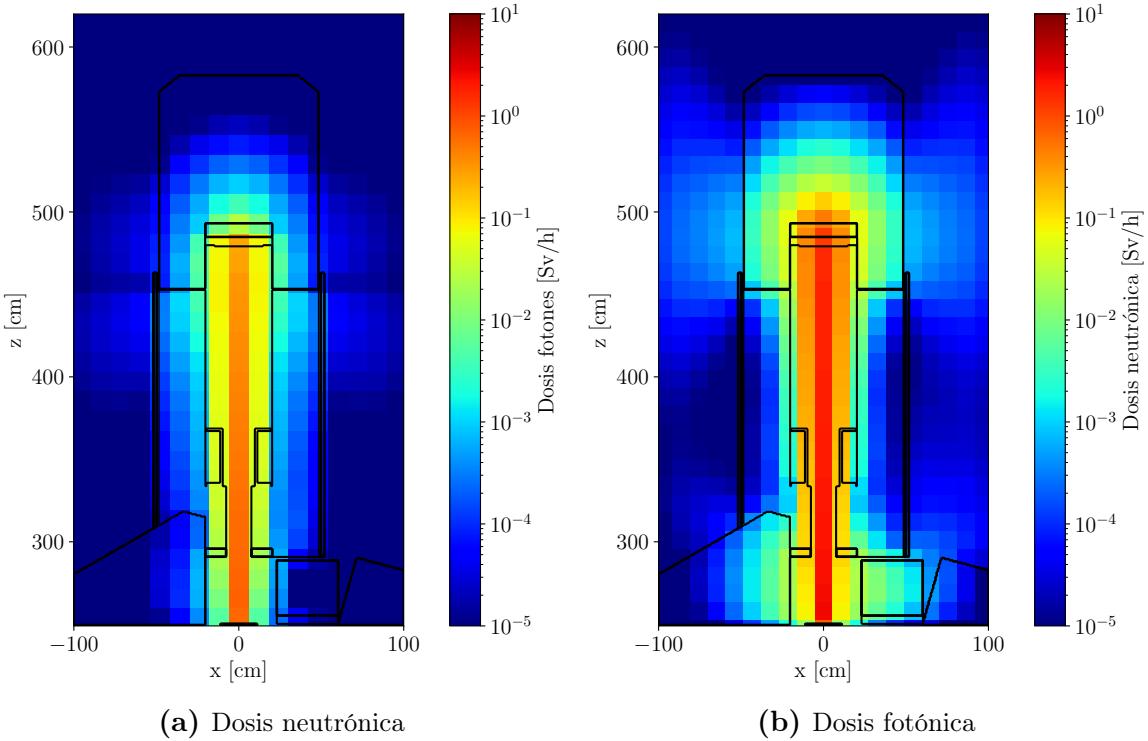


Figura 5.7: Dosis equivalente ambiental $H^*(10)$ utilizando **KDSource** a partir de la superficie S2.

Se observa una mayor dosis correspondiente a los neutrones que a los fotones hacia las afueras del conducto, algo que no era visible en los mapas de flujo. Esto se debe a la dependencia de la dosis equivalente ambiental con la energía y el tipo de radiación incidente (los neutrones tienen un factor de ponderación de radiación mayor que los fotones [25]).

A partir de los datos experimentales obtenidos en las posiciones presentes en la Figura 5.8 y constatados en el documento [26], se realizó una validación de la implementación. A su vez se, realizó una comparación con resultados previos de utilizar **KDSource** en esta geometría aplicando la rutina de trabajo tradicional [9].

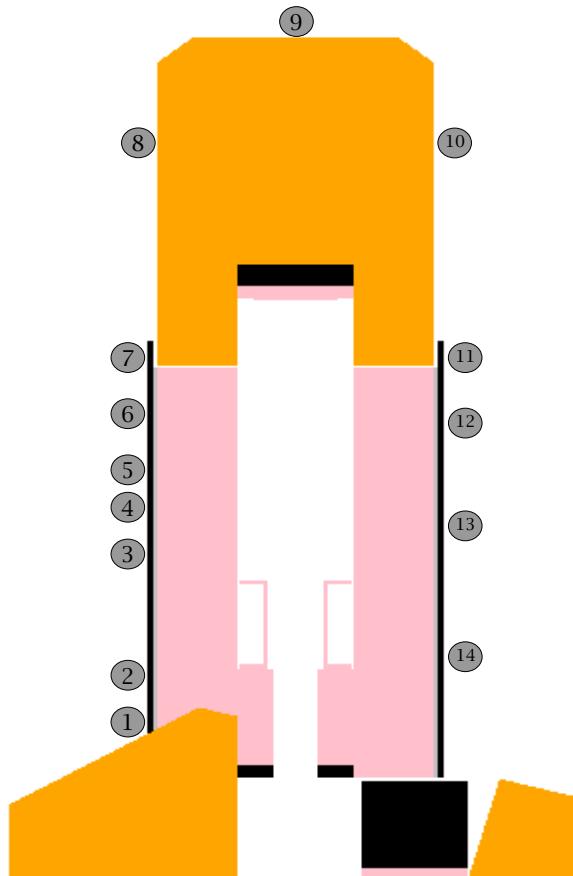


Figura 5.8: Posiciones en el contorno del conducto de neutrógrafía, para constatar los resultados obtenidos con el muestreo *on-the-fly*, con los resultados de la rutina tradicional y las mediciones experimentales anteriores.

En la Tabla 5.3 se presenta la contribución de neutrones a la tasa de dosis equivalente ambiental. Se puede ver que los resultados obtenidos con KDSOURCE *on-the-fly* son muy similares tanto a los experimentales como a los obtenidos en la rutina tradicional, pero el error se reduce en un orden de magnitud. Aunque no se observa una mejora en cuanto a la precisión general de los resultados, no es atribuible a la falta de estadística.

A su vez, en la Tabla 5.4 se observa la contribución de fotones a la tasa de dosis equivalente ambiental, donde se aprecia una mayor precisión de la rutina *on-the-fly* en comparación con la tradicional, con relación a los resultados experimentales. Sin embargo, se observa que en el punto 9, ubicado en una zona complicada para el transporte de fotones, no se registra una mayor dosis que en las simulaciones realizadas por el método tradicional. Además, se descarta una discrepancia por baja estadística, ya que se logra un error de simulación menor al 4 %.

Dosis equivalente ambiental $H^*(10)$ neutrónica [$\mu\text{Sv}/\text{h}$]					
Posición	Experimental	<i>on-the-fly</i>	C/E	Tradicional	C/E
2	30(6)	30,5(5)	1,0(3)	27(3)	0,9(3)
3	30(6)	13,0(3)	0,4(1)	17,0(11)	0,6(2)
4	22(5)	19,2(4)	0,9(3)	23,6(12)	1,1(3)
5	25(5)	31,7(6)	1,3(3)	28(2)	1,1(3)
7	120(30)	161,7(11)	1,3(3)	146(4)	1,2(3)
8	65(13)	74,6(7)	1,1(3)	42(4)	0,6(2)

Tabla 5.3: Tasa de dosis de neutrones en función de las posiciones ilustradas en la figura 5.8, para las distintas rutinas de **KDSouce**. A su vez se constatan los resultados experimentales de dichas posiciones. Se presentan los errores absolutos de cada magnitud.

Dosis equivalente ambiental $H^*(10)$ de fotones [$\mu\text{Sv}/\text{h}$]					
Posición	Experimental	<i>on-the-fly</i>	C/E	Tradicional	C/E
1	39(8)	27,4(2)	0,7(2)	5(3)	0,13(10)
2	36(8)	36,5(3)	1,0(3)	7(4)	0,2(2)
3	43(9)	58,0(3)	1,3(3)	39(4)	0,9(3)
4	70(20)	97,2(4)	1,4(4)	40(4)	0,6(3)
5	110(30)	131,7(5)	1,2(4)	43(3)	0,4(2)
6	140(30)	253,5(7)	1,8(4)	54(2)	0,39(10)
7	220(50)	314,3(7)	1,4(4)	54(1)	0,25(6)
8	21(5)	26,6(2)	1,3(4)	49(3)	2,3(7)
9	13(3)	1,42(5)	0,11(3)	2(4)	0,2(4)
10	47(10)	25,9(2)	0,6(2)	57(2)	1,2(3)
11	280(60)	297,5(7)	1,1(3)	60(2)	0,21(6)
12	420(90)	247,6(4)	0,6(2)	53(2)	0,13(3)
13	90(20)	76,5(6)	0,9(2)	41(3)	0,5(2)
14	19(4)	49,7(4)	2,6(6)	35(2)	1,8(5)

Tabla 5.4: Tasa de dosis equivalente ambiental proveniente de fotones en función de las posiciones ilustradas en la figura 5.8, para las distintas rutinas de **KDSouce**. Además, se muestran los resultados experimentales de dichas posiciones. Se presentan los errores absolutos de cada magnitud.

El error presentado en las tablas de dosis anteriores, en el caso de la utilización de **KDSouce**, solo tiene en cuenta el error estadístico de la simulación, a partir de la Ecuación 5.5. Donde H_i es el valor de la dosis en el *batch* i y \bar{H}_i es el promedio de ella en todos los *batches*. Para el caso de la rutina tradicional, fue posible reconstruir dicha

expresión a partir de múltiples simulaciones con distintas fuentes. Sin embargo, con la nueva implementación, esta funcionalidad se utiliza nativamente desde `OpenMC` [27].

$$\delta H = \sqrt{\frac{1}{N_{batch} - 1} \left(\frac{1}{N_{batch}} \sum_{i=1}^{N_{batch}} H_i^2 - \overline{H}_i^2 \right)} \quad (5.5)$$

No se tienen en cuenta múltiples posibles errores introducidos por el modelo, las secciones eficaces [28], la estimación de la fuente por `KDSource`, entre otros. Sin embargo, ahora que la estadística no es un problema, surge la posibilidad de analizar más en detalle estas contribuciones, como la sensibilidad del método `KDSource` con los anchos de banda, la función *kernel* y la lista de partículas iniciales.

En esta aplicación se observa realmente la ventaja de esta nueva técnica de muestreo, y cómo amplía el rango de utilización de la herramienta `KDSource`, haciéndola más interesante y cómoda para su uso en una cadena de cálculo. Ya no es necesario el guardado de listas de partículas como fuentes de `KDSource`, ni la necesidad de realizar múltiples simulaciones para obtener un resultado más preciso, o poder conocer un error estadístico. Ahora se requiere del mismo esfuerzo para simular cualquier cantidad de partículas.

Capítulo 6

Conclusiones

En el presente trabajo se desarrolló una técnica de muestreo *on-the-fly* optimizando la rutina de cálculo de una simulación Monte Carlo utilizando `KDSource` y `OpenMC`. Este desarrollo ha permitido reducir el uso de memoria RAM a niveles despreciables, lo que posibilita la simulación de un número considerablemente mayor de partículas, limitado únicamente por los tiempos y recursos de cómputo disponibles.

Además, se implementó una versión del muestreo apta para simulaciones *multi-thread*, permitiendo la ejecución en paralelo y asegurando la obtención de resultados repetibles bajo las mismas configuraciones. Para esto se realizó un análisis detallado del funcionamiento del muestreo de `OpenMC` y de cómo asigna las semillas del generador de números aleatorios para cada partícula, realizándose pruebas de reproducibilidad de resultados y confirmando la consistencia del método implementado.

Se llevaron a cabo múltiples pruebas para evaluar la eficiencia de la rutina de muestreo, comparando los resultados obtenidos con los métodos tradicionales. Estas pruebas demostraron que el uso de `KDSource` *on-the-fly* no solo mejora el manejo de memoria, sino que prácticamente no pierde eficiencia en cuanto a tiempos de simulación y precisión de los resultados. La generación de partículas *on-the-fly* representa un tiempo despreciable con respecto al tiempo de simulación.

Se corroboró que la fuente no perturbara la distribución de las partículas en su espacio de fases, observándose que el muestreo *on-the-fly* produce resultados similares al muestreo de partículas convencional.

Finalmente, se aplicó esta herramienta al caso de estudio relacionado con el haz de neutrografía del reactor RA-6. Se logró un aumento significativo en la estadística obtenida en comparación con simulaciones previas, disminuyendo el error estadístico en un orden de magnitud, lo que a su vez se simplificó la rutina de trabajo desde la generación de la fuente hasta la obtención de resultados, con la posibilidad de calcular el error estadístico a partir de la simulación en múltiples *batches*. Los resultados obtenidos para la dosis equivalente ambiental de neutrones y fotones fueron comparables con los

datos experimentales, observándose una mejora en la precisión para los fotones respecto a la rutina tradicional. Además, se identificó que las discrepancias observadas en algunos puntos respecto a los valores experimentales no se deben a una falta de estadística en la simulación, sino a otras fuentes de error no consideradas. Estas se pueden atribuir al modelado de las fuentes por `KDSource`, las secciones eficaces utilizadas y la geometría del modelo.

6.1. Trabajo futuro

Así como en este trabajo se ha logrado separar naturalmente fotones y neutrones en las simulaciones, es posible que una fuente presente partículas de la misma naturaleza, pero con grupos diferenciados por características específicas. Manejarlos con la misma correlación en el espacio de fases podría ser un error, por lo tanto, se propone la implementación de un selector de grupos de partículas dentro de `KDSource`. De esta forma se podría asignar configuraciones de anchos de banda específicos a cada uno. Esto es distinto del método KDE adaptativo (ver Sección 2.2.2), ya que propone una segregación más fuerte. En algunas superficies, se podrían mejorar los resultados si la perturbación no causa el apartamiento de la partícula al grupo que pertenece. Por ejemplo si existen diversos materiales, las mismas deberían ser registradas y muestreadas dentro del mismo material.

A su vez, queda como trabajo a futuro incorporar una herramienta para que el compilador de `KDSource` entienda su funcionamiento en paralelo, como lo hace `OpenMC` con MPI y OpenMP. Esto probablemente aumentaría la eficiencia en la generación de partículas, y facilitaría la implementación de un muestreo *on-the-fly* con otros códigos de transporte de radiaciones.

Simultáneamente, se podría mejorar el método de compilación de `KDSource` para que sea más amigable al usuario, incorporando un archivo configurable donde se establezcan las opciones de compilación. Además de ser más ordenado, facilitaría incorporar `KDSource` como un *plugin* de `OpenMC`, para ser parte del código en un futuro.

Apéndice A

Implementación de la clase KernelDensitySource

A.1. Inclusión de KDSouce dentro de OpenMC

En el Código A.1 se muestra como se debe incluir la librería `libkdsouce` en el proyecto OpenMC. La misma debe ser introducida dentro del apartado donde se genera el ejecutable `libopenmc`, en el fichero `CMakeLists.txt` del directorio `openmc`.

La primera línea de código incluye los archivos de cabecera de la librería en C de `KDSouce` para ser accesibles a tiempo de compilación. Por otro lado, la segunda línea, vincula la librería `libkdsouce.so` ya compilada, al ejecutable mencionado. De esta forma, al compilarlo se vuelven accesibles todas las funcionalidades de `KDSouce` dentro del código fuente de OpenMC.

```
1 include_directories(path_to_libkdsouce_headers)
2 target_link_libraries(libopenmc path_to_libkdsouce.so)
```

Código A.1: Compilación conjunta de `KDSouce` y `OpenMC`.

A.2. Clase KernelDensitySource

En el bloque de Código A.2 se muestra un ejemplo de cómo se puede utilizar la clase `KernelDensitySource` en un script de Python. En este caso, se inicializa la fuente desarrollada a partir de un archivo XML. A su vez, se activa la perturbación de las partículas y se ajusta el peso de las mismas.

```
1 import openmc
2 settings = openmc.Settings()
3 settings.source = openmc.KernelDensitySource(
4     path          = 'path_to_KDSouce.xml',
5     perturb      = True,
```

```

6     adjust_weight = True
7
8     )
9
10    settings.run_mode = "fixed source"
11    settings.batches = 1E2
12    settings.particles = 1E7

```

Código A.2: Ejemplo de utilización de una fuente KernelDensitySource.

En el bloque de Código A.3 se muestra el constructor de la clase `KernelDensitySource` desarrollada en C++. En el mismo, se inicializa una instancia de `KDSouce` a partir del archivo XML que recibe como argumento su clase equivalente en la API en Python. Esta instancia tiene acceso a toda la información para la generación de partículas (el archivo MCPL y las configuraciones del método KDE utilizado).

```

1 KernelDensitySource::KernelDensitySource(pugi::xml_node node)
2 {
3     auto path = get_node_value(node, "KDSouce", false, true);
4     perturb = get_node_value_bool(node, "perturb");
5     if (ends_with(path, ".xml")) {
6         const char* filename = path.data();
7         kdsouce = KDS_open(filename);
8     }
9 }

```

Código A.3: Constructor de la clase `KernelDensitySource` en C++.

En el bloque de Código A.4 se muestra el método `sample()` de la clase `KernelDensitySource`. El cual realiza el muestreo de una partícula a partir de la función `KDS_sample2()` de la librería `KDSouce`. Esta función recibe dos direcciones de memoria, una es la del propio objeto `kdsouce` de la clase, con la información de la perturbación y la otra es donde se guardará la partícula, luego de ser leída del archivo MCPL y perturbada por el código fuente de `KDSouce`. Por último se usa la función `mcpl_particle_to_site()` para crear una representación de partícula que OpenMC entienda durante la simulación.

```

1 SourceSite KernelDensitySource::sample(uint64_t* seed) const
2 {
3     mcpl_particle_t particle;
4     const mcpl_particle_t* ptr_particle = &particle;
5     KDS_sample2(kdsouce, &particle, perturb, w_critic, NULL, 1);
6     return mcpl_particle_to_site(ptr_particle);
7 }

```

Código A.4: Muestreo de la fuente `KernelDensitySource` en C++.

A.3. Implementación de una versión apta para simulaciones *multi-threading*

En el bloque de Código A.5 se presenta la clase `KernelDensitySource` en C++ con la implementación de un método para la generación de partículas en paralelo, utilizando la librería de OpenMP para lograrlo.

Como se mencionó en la Sección 3.3.1, para esta implementación se realizó una versión de la fuente que tenga un desplazamiento en la posición de lectura de la lista de partículas, para cada *thread*, por esto es que se optó por utilizar un contenedor de objetos `KDSource`, uno por cada thread. A su vez se agrega un vector de desplazamientos llamado `threads_offset` para llevar un control de la posición de lectura de cada instancia.

```

1 class KernelDensitySource : public Source {
2 public:
3     explicit KernelDensitySource(pugi::xml_node node); // Constructor
4     ~KernelDensitySource(); // Destructor
5     // Metodos
6     SourceSite sample(uint64_t* seed) const override;
7     void load_KDSource_from_file(const std::string& path);
8     void set_seed_to_pertub(uint64_t* seed, size_t i) const;
9     void reset_source_for_batch() const;
10    private:
11        // Atributos
12        vector<KDSource*> kdsouce;
13        mutable vector<uint64_t> threads_offset;
14        uint64_t mcpl_nparticles;
15        bool perturb;
16        double w_critic = 0;
17        mutable int current_batch = 1;
18 };

```

Código A.5: Composición de la clase `KernelDensitySource` en C++, capaz de realizar simulaciones en paralelo.

En el Código A.6, se muestra el constructor de la clase `KernelDensitySource` en C++. En contraste con la implementación anterior, en vez de inicializar un objeto `KDSource`, llama al método `load_KDSource_from_file()` con el *path* del archivo XML.

```

1 KernelDensitySource::KernelDensitySource(pugi::xml_node node)
2 {
3     auto path = get_node_value(node, "KDSource", false, true);
4     perturb = get_node_value_bool(node, "perturb");
5
6     if (!get_node_value_bool(node, "adjust_weight"))
7         w_critic = -1;

```

```
8     this->load_KDSource_from_file(path);  
9 }
```

Código A.6: Constructor de la clase KernelDensitySource en C++.

El método `load_KDSource_from_file()` se encarga de crear la fuente de partículas a partir de un archivo XML. En el Código A.7 se puede ver que se inicializa un objeto del tipo `KDSource` por cada *thread*. Estos mismos son iguales entre ellos, pero poseen un desplazamiento diferente dado por el vector `threads_offset`. Se calculan los factores de desplazamiento necesarios para cada *thread*, en función de la cantidad de partículas que cada uno va a simular por *batch*. Recordando la forma de asignación de semillas presentada en la Sección 3.3.1, a todos se les asigna la misma cantidad de partículas, y las sobrantes se reparten en los primeros *threads*.

Por último las funcionalidades `PList_seek()` y `Geom_seek()` se encargan de mover el puntero de lectura de la lista de partículas y de los anchos de banda, respectivamente, a la posición correspondiente al desplazamiento calculado.

```

1 void KernelDensitySource::load_KDSource_from_file(const std::string&
2   path)
3 {
4   if (ends_with(path, ".xml")) {
5     const char* filename = path.data();
6
7     // Reserva de memoria necesaria
8     kdsource.reserve(num_threads());
9     threads_offset.reserve(num_threads());
10
11    // Se inicializan variables necesarias
12    kdsource[0] = KDS_open(filename);
13    mcpl_nparticles = kdsource[0]->plist->npts;
14    threads_offset[0] = 0;
15
16    // Se calculan factores de desplazamiento
17    uint64_t part_thr =
18      openmc::settings::n_particles / num_threads();
19    uint64_t rest_part_thr =
20      openmc::settings::n_particles % num_threads();
21    for (int i = 1; i < num_threads(); i++) {
22      kdsource[i] = KDS_open(filename);
23      if (i < rest_part_thr) {
24        threads_offset[i] = i * (part_thr + 1) ;
25      } else {
26        threads_offset[i] = i * part_thr + rest_part_thr ;
27      }
28      PList_seek(kdsource[i]->plist,
29                 threads_offset[i] % mcpl_nparticles);

```

```

29     Geom_seek(kdsouce[i]->geom,
30                 threads_offset[i] % mcpl_nparticles);
31 }
32 } else {
33     fatal_error("Specified starting source file not a source file type
34             "
35             "compatible with KDSource.");
36 }

```

Código A.7: Método `load_KDSource_from_file` de la clase `KernelDensitySource` en C++.

Esta implementación presenta dos diferencias principales en el muestreo de las partículas, en relación a la versión presentada anteriormente, como se puede ver en el Código A.8. La primera es que se realiza bajo la directiva `omp critical`, para evitar los problemas mencionados en la Sección 3.3.1. La segunda es que se inicializa la secuencia de números aleatorios en cada muestreo a partir del método `set_seed_to_pertub()`. Por último se observa que se realiza un nuevo desplazamiento en las posiciones de lectura de cada objeto `KDSource` por cada *batch*, a partir del método plasmado en el Código A.9.

```

1 SourceSite KernelDensitySource::sample(uint64_t* seed) const
2 {
3     mcpl_particle_t particle;
4     const mcpl_particle_t* ptr_particle = &particle;
5 #pragma omp critical
6 {
7     if (openmc::simulation::current_batch > current_batch)
8     {
9         current_batch++;
10        this->reset_source_for_batch();
11    }
12    if (perturb)
13        this->set_seed_to_pertub(seed, thread_num());
14    KDS_sample2(kdsouce[thread_num()], &particle,
15                perturb, w_critic, NULL, 1);
16 }
17 return mcpl_particle_to_site(ptr_particle);
18 }

```

Código A.8: Método `sample()` de la clase `KernelDensitySource` en C++.

```

1 void KernelDensitySource::reset_source_for_batch() const
2 {
3     for (int i = 0; i < num_threads(); i++) {
4         threads_offset[i] += openmc::settings::n_particles;
5         PList_seek(kdsouce[i]->plist,
6                     threads_offset[i] % mcpl_nparticles);

```

```

7     Geom_seek(kdsource[i]->geom,
8                 threads_offset[i] % mcpl_nparticles);
9 }
10 }
```

Código A.9: Método `reset_source_for_batch()` de la clase `KernelDensitySource` en C++.

La variable `w_critic`, representa un peso de referencia que se utiliza en el algoritmo de restablecimiento de pesos explicado en [3.2.1](#). En caso de que el peso w de una partícula leída del archivo MCPL, sea mayor a él, se la sortea nuevamente con una probabilidad $(1 - w_{critic}/w)$ (*splitting*). En caso de que el peso sea menor, se usa w/w_{critic} como probabilidad de tomarla (*ruleta rusa*). La variable `w_critic` se calcula a partir de un promedio de las primeras mil partículas en la lista.

A.4. Incorporación de unit tests

Se incorporaron dos *unit tests* con el objetivo de poder verificar el correcto funcionamiento de la clase `KernelDensitySource`. El primer *test* se encarga de verificar que la clase `KernelDensitySource` pueda ser instanciada y que pueda ser utilizada en una simulación de `OpenMC` en paralelo. A su vez verifica si el muestreo se realiza teniendo en cuenta todas las partículas. Como mencionamos en la Sección [4.1](#), esto se consigue generando una fuente de partículas conocidas, y simulando la misma cantidad de ellas. Luego se observa que no exista alguna repetida.

El segundo test se encarga de verificar que la clase `KernelDensitySource` pueda generar resultados reproducibles entre dos simulaciones contiguas.

En ambos *tests* se utiliza un modelo de `OpenMC` simple. Se propone una esfera de vacío, con una fuente puntual, registrando las superficies que llegan a la esfera. A su vez, se configura la fuente de forma que no use el algoritmo de ajuste de los pesos.

Bibliografía

- [1] Robert, C. P., Casella, G. Monte Carlo Statistical Methods. 1^a ed.^{ón}. New York, NY: Springer, 1999. [2](#)
- [2] OpenMC. Variance Reduction Techniques, 2024. URL https://docs.openmc.org/en/stable/methods/neutron_physics, accedido: 2024-06-01. [2](#)
- [3] Fairhurst, R. Cálculo neutrónico detallado de haces y guías de neutrones del reactor RA-10. Proyecto Integrador de la carrera de Ingeniería Nuclear, Instituto Balseiro, Junio 2017. [2](#)
- [4] Ayala, J. Implementación de una línea de cálculo basada en el código TRIPOLI a problemas de blindaje del reactor RA-10. Proyecto Integrador de la carrera de Ingeniería Nuclear, Instituto Balseiro, Junio 2019.
- [5] Cisterna, G. Diseño y performance de las guías de neutrones fríos para el instrumento SANS del laboratorio Argentino de haces de neutrones del reactor RA-10. Proyecto Integrador de la carrera de Ingeniería Nuclear, Instituto Balseiro, Junio 2020. [2](#)
- [6] Schmidt, N., Abbate, O.I., Prieto, Z.M., Robledo, J.I., Márquez, J.I., Márquez, A.A., Dawidowski, J. KDSource, a tool for the generation of Monte Carlo particle sources using kernel density estimation. *Annals of Nuclear Energy*, **177**, 109309, 2022. URL <https://www.sciencedirect.com/science/article/pii/S0306454922003449>. [2](#), [7](#), [8](#), [9](#)
- [7] Abbate, I. KDSource: Desarrollo de una herramienta computacional para el cálculo de blindajes. *Tesis carrera de maestría en ingeniería, Instituto Balseiro*, Diciembre 2021. [2](#), [8](#)
- [8] Prieto, Z. Incorporación de algoritmos de KDE al modelado de fuentes de radiación en cálculos de Monte Carlo. Proyecto Integrador de la carrera de Ingeniería Nuclear, Instituto Balseiro, Junio 2021. [2](#), [8](#), [35](#)

- [9] Fox, F. Optimización de algoritmos de estimación de densidades para el cálculo de haces de neutrones y fotones. Proyecto Integrador de la carrera de Ingeniería Nuclear, Instituto Balseiro, Junio 2022. [2](#), [35](#), [41](#), [44](#)
- [10] Lefmann, K., Nielsen, K. McStas, a general software package for neutron ray-tracing simulations. *Neutron News*, **10** (3), 20–23, 1999. URL <https://doi.org/10.1080/10448639908233684>. [4](#)
- [11] Willendrup, P., Farhi, E., Lefmann, K. McStas 1.7 - a new version of the flexible Monte Carlo neutron scattering package. *Physica B: Condensed Matter*, **350** (1, Supplement), E735–E737, 2004. URL <https://www.sciencedirect.com/science/article/pii/S0921452604004144>, proceedings of the Third European Conference on Neutron Scattering. [4](#)
- [12] Brun, E., Damian, F., Diop, C., Dumonteil, E., Hugot, F., Jouanne, C., *et al.* TRIPOLI-4®(R), CEA, EDF and AREVA reference Monte Carlo code. *Annals of Nuclear Energy*, **82**, 151–160, 2015. URL <https://www.sciencedirect.com/science/article/pii/S0306454914003843>, joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013, SNA + MC 2013. Pluri- and Trans-disciplinarity, Towards New Modeling and Numerical Simulation Paradigms. [4](#)
- [13] Romano, P. K., Horelik, N. E., Herman, B. R., Nelson, A. G., Forget, B., Smith, K. OpenMC: A state-of-the-art Monte Carlo code for research and development. *Annals of Nuclear Energy*, **82**, 90–97, 2015. URL <https://www.sciencedirect.com/science/article/pii/S030645491400379X>, joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013, SNA + MC 2013. Pluri- and Trans-disciplinarity, Towards New Modeling and Numerical Simulation Paradigms. [4](#), [12](#)
- [14] Silverman, B. W. Density Estimation for Statistics and Data Analysis. London: Chapman and Hall, 1986. [7](#)
- [15] Terrell, G. R., Scott, D. W. Variable Kernel Density Estimation. *The Annals of Statistics*, **20** (3), 1236 – 1265, 1992. URL <https://doi.org/10.1214/aos/1176348768>. [7](#)
- [16] KDSource. Repositotio en GitHub, 2023. URL <https://github.com/KDSource/KDSource>, accedido: 2024-06-01. [9](#), [11](#)
- [17] OpenMC. Repositotio en GitHub, 2023. URL <https://github.com/openmc-dev/openmc>, accedido: 2024-06-01. [12](#)

- [18] OpenMC. Documentación. <https://docs.openmc.org/en/stable/>. 12
- [19] OpenMC. Manual OpenMC: instalación, 2023. URL <https://docs.openmc.org/en/stable/usersguide/install.html?highlight=mcpl>, accedido: 2024-06-01. 14
- [20] Kittelmann, T., Klinkby, E., Knudsen, E., Willendrup, P., Cai, X., Kanaki, K. Monte Carlo Particle Lists: MCPL. *Computer Physics Communications*, **218**, 17–42, 2017. URL <https://www.sciencedirect.com/science/article/pii/S0010465517301261>. 15
- [21] HDF5 Documentation. <https://docs.hdfgroup.org/hdf5/develop/index.html>. 15
- [22] OpenMP Reference Guide. <https://www.openmp.org/wp-content/uploads/OpenMPRefGuide-5.2-Web-2024.pdf>. 16
- [23] CNEA. Descripción de la instalación experimental de radiografía de neutrones del RA-6. Inf. Téc. ITE-EN-GIN-RI-PL-010, CNEA, 2021. 35
- [24] ARN. GUÍA AR 1: Factores dosimétricos para irradiación externa y contaminación interna, y niveles de intervención para alimentos. https://www.argentina.gob.ar/sites/default/files/gr1-r1_0.pdf, 2003. 43
- [25] ARN. GUÍA AR 1: Factores dosimétricos para exposición externa y exposición interna, niveles guía de radionucleidos en alimentos y agua, y recomendaciones para el control de la exposición a gas radón, 2022. Revisión 2, 24/06/22, página 5. 44
- [26] Marín, J., J., A. Relevamiento radiológico preliminar de la instalación experimental de neutrografía del RA-6 a 1MW. Inf. Téc. ITE-EN-GIN-RI-PL-012, CNEA, 2021. 44
- [27] OpenMC. Manual OpenMC: Tallies. <https://docs.openmc.org/en/stable/methods/tallies.html>. 47
- [28] OpenMC. Biblioteca de secciones eficaces microscópicas, 2024. URL <https://openmc.org/official-data-libraries/>, accedido: 2024-06-01. 47

Agradecimientos

Agradezco a mis amigos de tesis, Zoe y Ariel. Por los buenos consejos, la gran paciencia y los bellos momentos en la oficina, donde las risas casi siempre fueron protagonistas.

Le agradezco a las sólidas amistades que me dio el IB. A la Scofleta y su humor, por los momentazos como compañero de escritorio; a Nacho por su ternura, su ocurrencia y los mates con burrito; a Jero por los nesquik con pool antes de dormir; y a Joaco por su gran compañerismo y por compartir su amor por la comida. Le agradezco a Alvaro, mi primer amigo en Bariloche, por su gran hermandad y preocupación; a Manu por su carácter, por compartir humor y festejarme los chistes; a Pedro por siempre escuchar y por su alta disponibilidad; a Martín por los partidos de tenis, por las numerosas traiciones y por ser mi cómplice en el instituto; a Eva y a Cynthia por su cariño y las risas compartidas.

A Lucho, por elegirme los tres años para compartir cuarto, por las charlas hasta la madrugada, por su preocupación y por la excelente relación que tenemos.

A Papá, a Mamá, a Lauti, a mis abuelos, a mi madrina y a mi hermosa familia por su amor y dedicación en mi vida.

A la genia de Lucia, por confiar en mí, por su amor, por su sinceridad y por siempre alentarme a realizar esta travesía. Le agradezco por recibirme cada vez que quería escapar y a su bella familia por hacerme parte de ella.

Le agradezco a mis amigos, por la compañía y lo mucho que me quieren.

Finalmente, me agradezco por animarme, y le agradezco al IB, a la UTN y al San Marcos por formarme.

