Simulación del blindaje de una fuente de fotones gamma mediante el método de Monte Carlo

Trabajo de fin de Máster

Borja Martínez Pérez

CURSO 2011-2012

Máster en Física de los Sistemas de Diagnóstico, Tratamiento y Protección en Ciencias de la Salud

Índice

1. INTRODUCCIÓN 4

2. PRINCIPIOS FÍSICOS 5

3. MÉTODO MONTE CARLO 11

4. MESSAGE PASSING INTERFACE 13

5. ALGORITMO DE BLINDAJE 15

5.1. DIAGRAMA DE FLUJO 15
5.2. CÓDIGO DEL PROGRAMA 18
5.3. DESARROLLO Y COMPROBACIÓN DEL PROGRAMA 19

6. RESULTADOS Y SIMULACIONES 20

6.1. SIMULACIÓN DE UN DETECTOR DE NAI 21
6.2. SIMULACIÓN DE UNA ESFERA DE PLOMO 24
6.3. SIMULACIÓN DE UNA ESFERA DE ÓXIDO DE URANIO 29
6.4. SIMULACIÓN DE UNA ESFERA DE PLOMO Y ÓXIDO DE URANIO 32

7. CONCLUSIONES Y LÍNEAS FUTURAS 36

APÉNDICE A. CÓDIGOS DE PROGRAMAS 37

CÓDIGO DEL PROGRAMA DE SIMULACIÓN DE UN DETECTOR DE RADIACIÓN DE NAI 37
CÓDIGO DEL PROGRAMA DE SIMULACIÓN DE BLINDAJE QUE OFRECE UNA ESFERA 41

APÉNDICE B. FICHEROS DE SECCIONES EFICACES. 49

FICHERO DE SECCIONES EFICACES DEL IODURO DE SODIO (NAI) 49
FICHERO DE SECCIONES EFICACES DEL PLOMO 50
FICHERO DE SECCIONES EFICACES DEL ÓXIDO DE URANIO 51

BIBLIOGRAFÍA 52

1. Introducción

Los objetivos de este trabajo son diseñar un algoritmo paralelizado para simular el blindaje de una fuente de fotones gamma mediante el método de Monte Carlo y realizar dichas simulaciones de la fuente de fotones blindada por esferas de plomo y de óxido de uranio de diferentes radios.

Para ello se simula una fuente de fotones que emite un número de fotones con una energía determinada no superior a 1 MeV, ambos elegidos por el usuario. Alrededor de esta fuente habrá una esfera de una o dos capas de radios y de materiales seleccionados por el usuario. Al terminar la ejecución del programa, se mostrará en pantalla la tasa de blindaje y se habrá creado un fichero con el espectro de los fotones salientes que podrá ser visualizado usando un programa de imágenes.

Para la simulación de los fotones se ha usado la técnica de Montecarlo, que es una técnica matemática computarizada que permite tener en cuenta el riesgo en análisis cuantitativos y tomas de decisiones, es decir, la simulación Monte Carlo ofrece a la persona responsable de tomar las decisiones una serie de posibles resultados, así como la probabilidad de que se produzcan según las medidas tomadas [1].

Así mismo, este programa estará paralelizado mediante técnicas de MPI (*Message Passing Interface* o Interfaz de Paso de Mensajes), que es un estándar que define la sintaxis y la semántica de las funciones contenidas en una biblioteca de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores [2]. Es decir, sirve para realizar operaciones en paralelo en máquinas con varios procesadores, disminuyendo de esta manera el tiempo de ejecución.

Mediante el uso de este programa, se podrán hacer distintas simulaciones que nos indicarán el blindaje de distintos tipos de composiciones y radio de las esferas.

2. Principios físicos

El algoritmo del blindaje se basa en los principios físicos básicos de la interacción de los fotones con la materia. La atenuación de un rayo de fotones mediante un material absorbente es causada por cinco tipos principales de interacciones [3]:

- Fotodesintegración.
- Efecto fotoeléctrico.
- Dispersión coherente o de Rayleigh.
- Dispersión Compton.
- Producción de pares.

En la Figura 1 [4] se muestra el coeficiente de atenuación lineal del ioduro de sodio (NaI) de los efectos fotoeléctrico, Compton y de producción de pares.

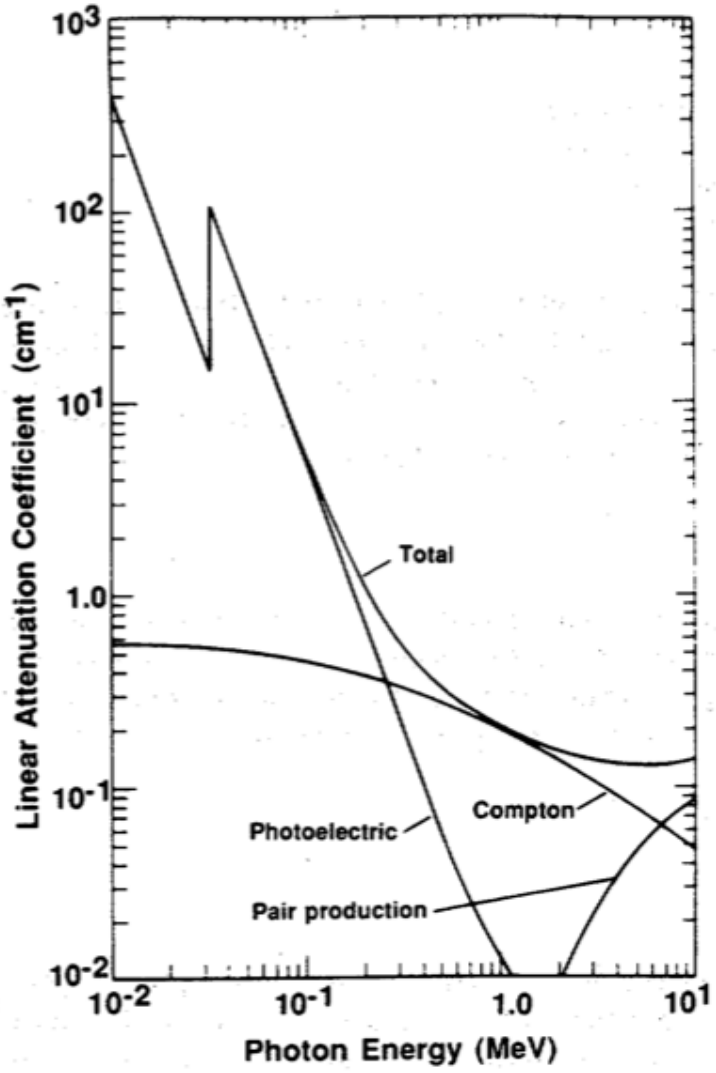

Figura 1. Coeficiente de atenuación lineal del NaI mostrando las contribuciones de la absorción fotoeléctrica, la dispersión Compton y la producción de pares.

Cuando una radiación de intensidad I_0 penetra en un material absorbente de grosor L, la intensidad transmitida al salir del material viene dada por la expresión:

$$I = I_0 e^{-\mu_l L}$$

donde μ_l es el coeficiente de atenuación lineal expresado en cm^{-1}. Este coeficiente depende de la energía de la radiación y del número atómico y densidad del material absorbente. El inverso del coeficiente de atenuación lineal $1/\mu_l$, se suele conocer como el camino libre promedio y, como se verá más adelante, es la distancia promedio que un fotón viaja en el medio absorbente hasta que interactúa con la materia por medio de algunos de los efectos mencionados y que se explican mejor a continuación.

La fotodesintegración es la interacción de un fotón de muy alta energía (mayor de 10 MeV) con un núcleo atómico que desemboca en una reacción nuclear y en la emisión de uno o más nucleones. Dado que en el programa se van a usar energías de hasta 1 MeV, no hace falta tener en cuenta este tipo de interacción. Este efecto no aparece en la Figura 1 porque sólo se muestra hasta el valor máximo de 10 MeV, que es justo cuando empiezan las energías a las que podría darse este tipo de interacción.

En la producción de pares lo que se produce es que el fotón interacciona fuertemente con el campo electromagnético del núcleo atómico y deja toda su energía en el proceso de creación de un par consistente en un electrón e^- y un positrón e^+. Esto puede observarse en la Figura 2. Como la energía en reposo del electrón es equivalente a 0.511 MeV, se necesita un mínimo de energía de 1.022 MeV para crear dicho par. Puesto que en el programa el máximo de energía será de 1 MeV, como se ha comentado, no se llega a esta energía umbral y tampoco se tendrá en cuenta esta interacción. En la Figura 1 se observa perfectamente cómo este efecto empieza a influir en el coeficiente de atenuación lineal a partir de 1.022 MeV.

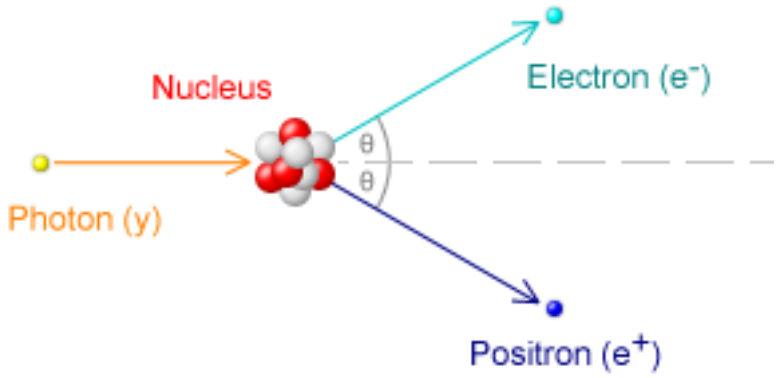

Figura 2. Esquema del efecto de producción de pares electrónicos.

El efecto fotoeléctrico es un fenómeno en el que un fotón interacciona con un átomo y extrae un electrón de alguna de las órbitas del átomo. Esta interacción se muestra en la Figura 3. En este proceso, la energía total del fotón *hν* se transfiere al electrón atómico o fotoelectrón. La energía cinética de dicho fotoelectrón es igual a

$h\nu - E_B$, donde E_B es la energía de enlace del electrón. Este tipo de interacciones pueden tener lugar con electrones en las capas K, L, M o N.

Después de que el electrón haya sido expulsado del átomo se ha creado una vacante en la capa dejando al átomo en un estado excitado. Esta vacante puede ser ocupada por un electrón de una órbita exterior con la emisión de rayos X característicos.

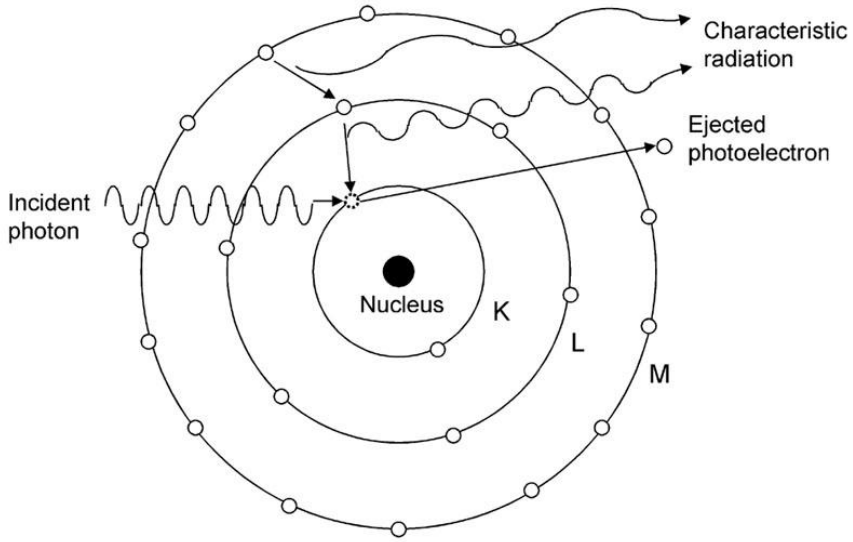

Figura 3. Esquema de efecto fotoeléctrico [5].

La probabilidad de la absorción fotoeléctrica depende de la energía del fotón como se observa en la Figura 4, donde se muestra el coeficiente de atenuación másico fotoeléctrico en función de la energía del fotón. Los datos son para el agua, que representa un material de número atómico bajo similar a los tejidos, y para el plomo, representando un material de número atómico alto. El gráfico tiene discontinuidades en el caso del plomo. Estas discontinuidades son los bordes de absorción y se corresponden con las energías de enlace de los electrones de las capas L y K.

Como se ha visto y como muestra la Figura 1, ha de tenerse en cuenta el efecto fotoeléctrico para el algoritmo, dado que se produce a energías bajas y, de hecho, es a energías bajas donde ocurre más a menudo.

La dispersión coherente o elástica o de Rayleigh se muestra en la Figura 5. El proceso puede ser visualizado considerando la naturaleza de onda de la radiación electromagnética. Esta interacción consiste en una onda electromagnética que pasa cerca de un electrón y lo hace oscilar. El electrón oscilante radia de nuevo la energía a la misma frecuencia que la onda incidente. Estas ondas dispersadas tienen la misma longitud de onda que las incidentes, por lo que no hay cambio de energía de las ondas: Se trata de una dispersión elástica. Este efecto es probable en materiales de alto número atómico y con fotones de baja energía, por tanto ha de ser tenido en cuenta en el desarrollo del algoritmo.

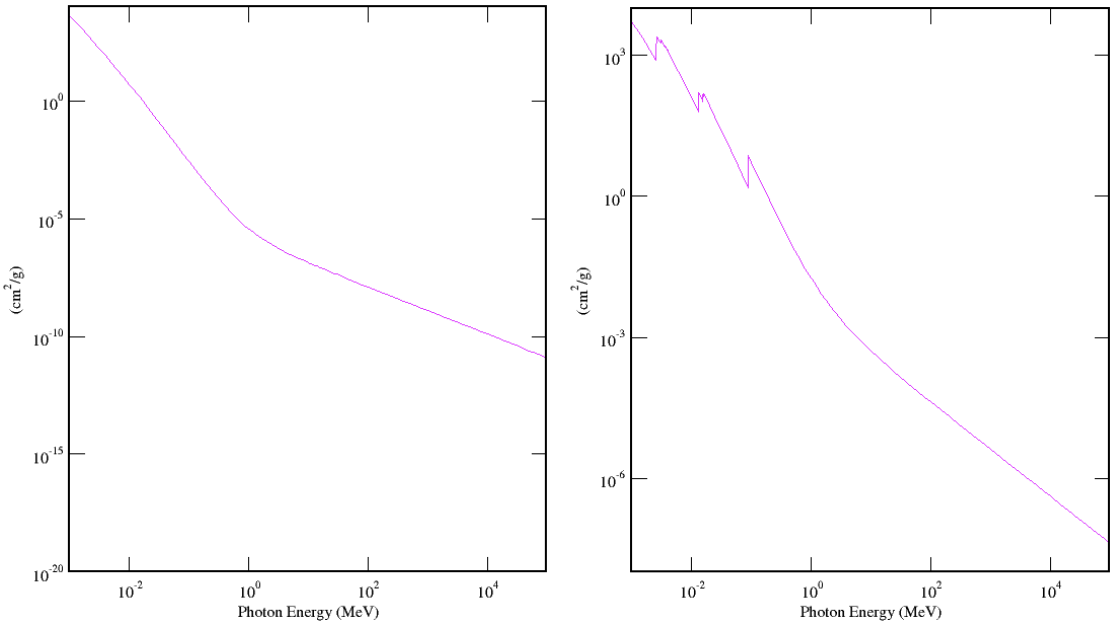

Figura 4. Coeficiente de atenuación másico fotoeléctrico en función de la energía del fotón. Curva izquierda para el agua (Z_{efec} = 7.42) y derecha para el plomo (Z = 82). Curvas obtenidas de [6].

Por último, en el efecto Compton, el fotón interacciona con un electrón atómico como si fuera un electrón "libre". El término "libre" quiere decir que la energía de enlace del electrón es mucho menor que la energía del fotón incidente. Esta interacción, como se observa en la Figura 6, es una dispersión inelástica: El electrón recibe parte de la energía del fotón y es emitido con ángulo θ mientras que el fotón, con menor energía, se dispersa en un ángulo ϕ.

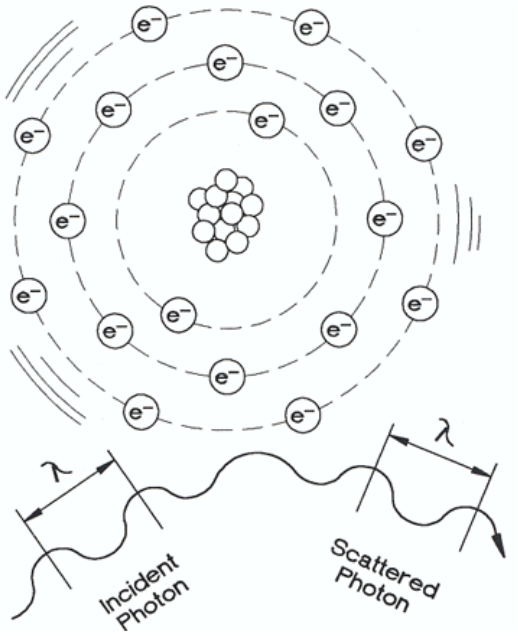

Figura 5. Esquema de la dispersión coherente.

El efecto Compton se puede analizar en términos de colisión entre dos partículas, un fotón y un electrón. Aplicando las leyes de conservación de energía y momento se obtiene la siguiente fórmula para la energía del fotón dispersado [5]:

$$E' = \frac{m_0 c^2}{1 - cos\phi + {}^{m_0 c^2}/_{E}}$$

donde E es la energía del fotón incidente,
E' es la energía del fotón dispersado,
$m_0 c^2$ representa la energía en reposo del electrón (511 keV),
ϕ es el ángulo entre los rayos incidente y dispersado.

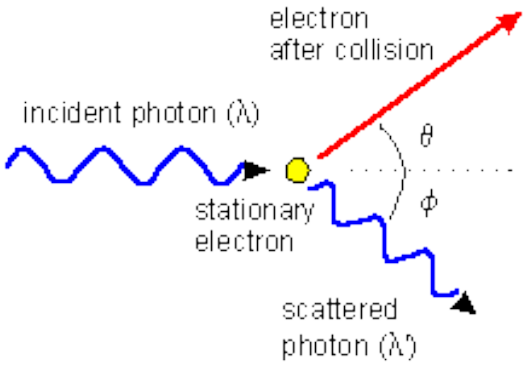

Figura 6. Esquema del efecto Compton.

Esta energía resultante es mínima para una colisión frontal donde el fotón es dispersado 180º y el electrón se mueve en la dirección del fotón incidente. Para ángulos de dispersión muy pequeños, en torno a 0º, la energía del rayo dispersado es ligeramente menor que la del incidente. En la Figura 7, obtenida de [4], se muestra la energía de fotones dispersados por efecto Compton como función del ángulo de dispersión y de la energía del rayo incidente. Las discontinuidades corresponden a la energía máxima que puede ser transferida en una sola dispersión [4].

Cuando se produce una dispersión Compton, el electrón dispersado es normalmente parado en el material de absorción. La dispersión Compton en un detector produce un espectro de pulsos de salida desde 0 hasta la energía máxima dada al electrón (cuando el ángulo de dispersión es de 90º). Como se mencionó anteriormente, el efecto Compton es una interacción entre un fotón y un electrón "libre". Prácticamente, esto quiere decir que la energía del fotón incidente debe ser grande comparada con la energía de enlace del electrón. Esto contrasta con el efecto fotoeléctrico que se va haciendo más probable cuando la energía del fotón incidente es igual o ligeramente superior a la energía de enlace del electrón. Por tanto, si se va incrementando la energía más allá del valor de enlace del electrón de la capa K, el efecto fotoeléctrico disminuye rápidamente con la energía y el efecto Compton se va haciendo cada vez más importante. Sin embargo, como se observa

en la Figura 1, el efecto Compton también disminuye con la energía del fotón. Merece la pena destacar que el efecto Compton es independiente del número atómico del material, ya que sólo se limita a electrones libres del material. Por tanto este efecto depende sólo del número de electrones por gramo. Aunque este número disminuye lenta pero sistemáticamente con el número atómico, se puede considerar que la mayoría de materiales, excepto el hidrógeno, tiene aproximadamente el mismo número de electrones por gramo y por tanto el efecto Compton es casi el mismo para todos los materiales [3].

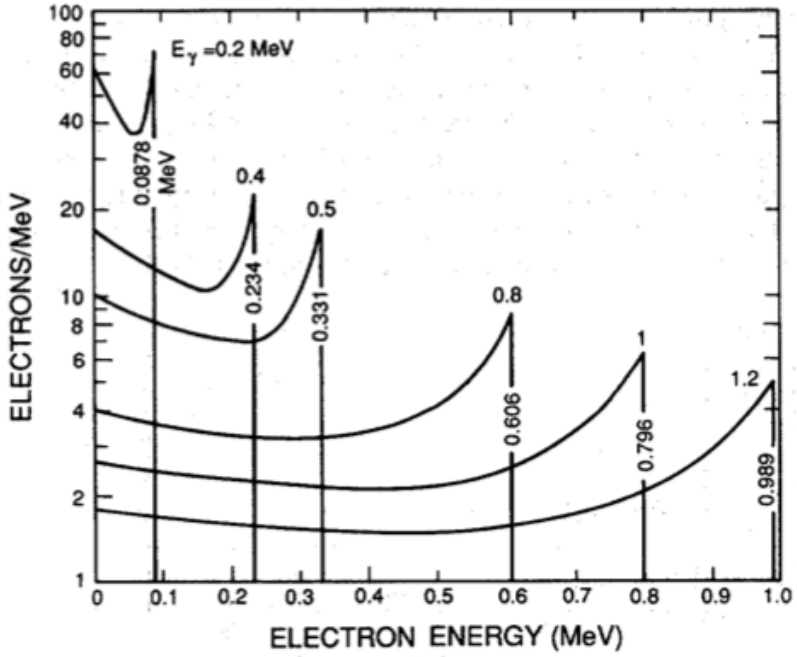

Figura 7. Energía de fotones dispersados por efecto Compton como función del ángulo de dispersión y de la energía del rayo incidente.

Este efecto tiene lugar en el rango de energías que se van a usar en el algoritmo y, por tanto, ha de ser tenido en cuenta en la simulación.

Por último, se muestra en la Figura 8 (extraída de [4]) el espectro de rayos gamma obtenido a partir de una fuente monoenergética de ^{137}Cs, que emite fotones de 662 keV. Se observa el fotopico formado en la energía de 662 keV, debido a efectos fotoeléctricos únicos que dejan toda la energía en el material y también combinados con efectos Compton anteriores que terminan siendo absorbidos por efecto fotoeléctrico. La parte del espectro anterior al fotopico está formada por dispersiones Compton donde los fotones pierden sólo parte de su energía en el material detector. El escalón cerca de 470 keV corresponde a la máxima energía que puede ser transferida a un electrón en un solo efecto Compton y se llama borde Compton. El pico pequeño a 188 keV es el pico de retrodispersión (el término inglés es *backscatter peak*) y se produce cuando un rayo sufre una dispersión de gran ángulo (en torno a 180º) en el material que rodea al detector y éste es absorbido dentro del detector.

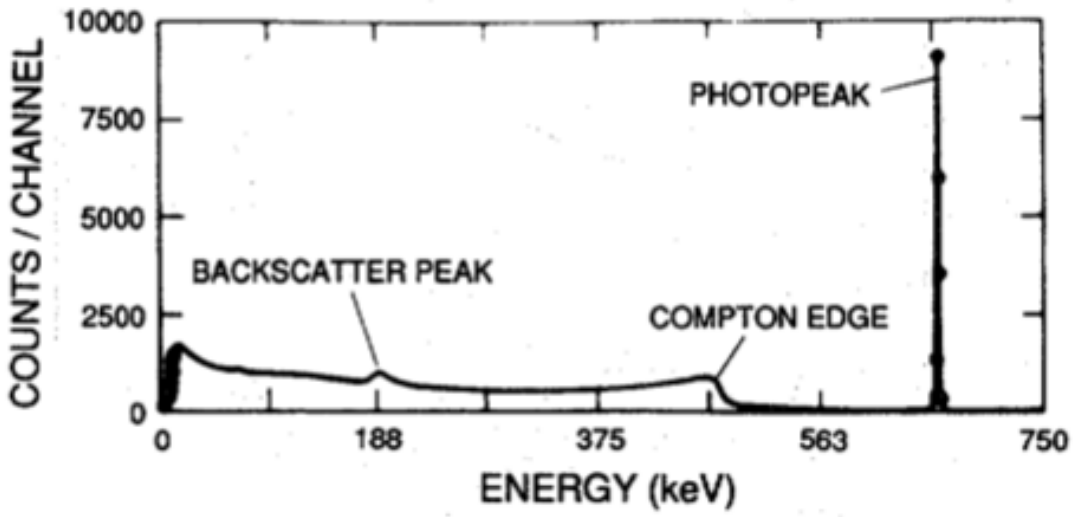

Figura 8. Espectro de una fuente de ^{137}Cs que muestra el fotopico, el borde Compton y el pico de retrodispersión para rayos gamma de 662 keV, registrado por un detector de NaI.

Aparte de los efectos hay que tener en cuenta la distancia recorrida por el fotón entre efecto y efecto, entre colisiones. Esto se define como el camino libre promedio (*mean free path*), que es la distancia promedio que viaja un fotón monoenergético en un material hasta que se producen colisiones con los átomos del mismo. Viene dado por la siguiente ecuación:

$$\lambda = \frac{1}{\sigma(E)\rho}$$

donde λ es el camino libre promedio (unidades de longitud),
σ(E) es la sección eficaz del material (unidades de superficie por átomo),
ρ es la densidad de centros de dispersión (número de centros de dispersión por volumen).

Téngase en cuenta que la sección eficaz del medio depende de la energía del fotón y del tipo de efecto que se produce.

Todos estos principios físicos son la base física del algoritmo de blindaje.

3. Método Monte Carlo

El método de Monte Carlo puede ser descrito como un conjunto de métodos estadísticos que usan números aleatorios como la base para realizar simulaciones de cualquier situación específica. Su nombre fue elegido en el Proyecto Manhattan durante la segunda guerra mundial por la fuerte conexión del método con los juegos basados en el azar y por la localización de un casino muy famoso en Monte Carlo [7].

Por tanto el método de Monte Carlo es un conjunto de algoritmos, no un único algoritmo, basados en muestreo repetitivo aleatorio y que se usan cuando no es posible o no es práctico llevar a cabo el cálculo de un resultado mediante un algoritmo determinista y generalmente se usan para cálculos en ordenadores [8].

La estructura general de los algoritmos es la siguiente:

- Definen un rango, intervalo o dominio de posibles números o entradas.
- Generan entradas o números aleatoriamente dentro del rango/intervalo/dominio.
- Llevan a cabo una computación determinista usando las entradas.
- Agregan los resultados de las computaciones individuales en un resultado final.

Como se ha visto este método se basa en la generación de números aleatorios. El problema es que un ordenador no genera números aleatorios como tales, sino números pseudo-aleatorios. Estos números son en realidad una serie de números que se repiten y que dependen de un número inicial, llamado semilla. Por tanto, si la semilla es igual en dos procesos distintos de generación de números aleatorios entonces las series de números serán exactamente iguales.

Es fundamental generar distintas semillas. Para ello una buena forma es utilizar el tiempo actual en el momento de la generación, puesto que siempre cambia. De hecho, en un ordenador el tiempo es el número de segundos desde el 1 de enero de 1970, o el número de milisegundos.

Hay que tener en cuenta también que los números que se generan son números comprendidos entre 0 y 1, por lo que si se desea un intervalo distinto, el usuario deberá aumentar o disminuir el intervalo dado por el ordenador mediante multiplicaciones y/o sumas.

Se ha visto que esta técnica hace uso de números aleatorios, o mejor dicho pseudo-aleatorios, por tanto está basado en probabilidades. Para obtener resultados precisos será necesario el uso de grandes cantidades de estos números y por tanto de muchas operaciones, aparte de la aleatoriedad de dichos números. Para ello hay dos posibilidades [9]:

- Una sola simulación Monte Carlo con una gran cantidad de números aleatorios y un buen generador de aleatoriedad.
- Muchas simulaciones Monte Carlo simultáneas del mismo problema, añadiendo los resultados al final.

En el caso del algoritmo la segunda opción será la elegida haciendo uso de la paralelización del algoritmo.

Como ya se ha comentado se hace uso de dicha técnica en los siguientes casos:

- No es práctico usar algoritmos deterministas:
    - Hay un número importante de grados de libertad.

    - Existe una gran complejidad del sistema.
- No es posible usar algoritmos deterministas:
    - Hay una gran incertidumbre de los datos de entrada, como cálculos de riesgo.
    - El modelo correcto del fenómeno a estudiar es desconocido. Por ejemplo, el algoritmo determinista.

Esta simulación es aplicable en diferentes ramas tanto en la física, la química, las matemáticas y las finanzas, entre otras, porque nos permite hacer muestreo estadístico empleando distribuciones de variables aleatorias, aplicándose a problemas ya sea estocásticos o determinísticos.

El método de Monte Carlo se crea para la resolución de integrales cuya respuesta es difícil de encontrar por métodos analíticos, aplicándole números aleatorios y así facilitar el cálculo. Se usa en aplicaciones tan dispares como las siguientes [8]:

- Diseño de reactores nucleares
- Cromo dinámica cuántica
- Radioterapia contra el cáncer
- Densidad y flujo de tráfico
- Evolución estelar
- Econometría
- Pronóstico del índice de la bolsa
- Prospecciones en explotaciones petrolíferas
- Diseño de VLSI
- Física de materiales
- Ecología
- Criptografía
- Valoración de cartera de valores
- Programas de ordenador
- Métodos cuantitativos de organización industrial

En el caso del algoritmo diseñado se usan los números aleatorios en varias partes: para hallar qué efecto se producirá, para hallar el camino recorrido por un fotón entre dos efectos y para hallar las coordenadas (r, θ, Φ) del fotón gamma. Si no se considera la dispersión de Rayleigh, se pueden dar sólo dos efectos, que el fotón se absorba en un efecto fotoeléctrico o que se produzca una dispersión Compton. En este caso el proceso X consiste en dos eventos: X = {absorción, dispersión} con probabilidades p(x) = {α, 1-α}. Como ya se ha visto esta probabilidad depende de la energía del fotón y del material absorbente.

4. Message Passing Interface

MPI ("*Message Passing Interface*", Interfaz de Paso de Mensajes) es un estándar que define la sintaxis y la semántica de las funciones contenidas en una biblioteca de

paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores [2].

El paso de mensajes es una técnica empleada en programación concurrente para aportar sincronización entre procesos y permitir la exclusión mutua. Su principal característica es que no precisa de memoria compartida, por lo que es muy importante en la programación de sistemas distribuidos.

Los elementos principales que intervienen en el paso de mensajes son el proceso que envía, el que recibe y el mensaje. Dependiendo de si el proceso que envía el mensaje espera a que el mensaje sea recibido, se puede hablar de paso de mensajes síncrono o asíncrono. En el paso de mensajes asíncrono, el proceso que envía, no espera a que el mensaje sea recibido, y continúa su ejecución, siendo posible que vuelva a generar un nuevo mensaje y a enviarlo antes de que se haya recibido el anterior. Por este motivo se suelen emplear buzones, en los que se almacenan los mensajes a espera de que un proceso los reciba. Generalmente empleando este sistema, el proceso que envía mensajes sólo se bloquea o para, cuando finaliza su ejecución, o si el buzón está lleno. En el paso de mensajes síncrono, el proceso que envía el mensaje espera a que un proceso lo reciba para continuar su ejecución. Por esto se suele llamar a esta técnica encuentro, o *rendezvous*. Dentro del paso de mensajes síncrono se engloba a la llamada a procedimiento remoto, muy popular en las arquitecturas cliente/servidor.

La Interfaz de Paso de Mensajes es un protocolo de comunicación entre computadoras. Es el estándar para la comunicación entre los nodos que ejecutan un programa en un sistema de memoria distribuida. Las implementaciones en MPI consisten en un conjunto de bibliotecas de rutinas que pueden ser utilizadas en programas escritos en los lenguajes de programación C, C++, Fortran y Ada. La ventaja de MPI sobre otras bibliotecas de paso de mensajes, es que los programas que utilizan la biblioteca son portables (dado que MPI ha sido implementado para casi toda arquitectura de memoria distribuida), y rápidos (porque cada implementación de la biblioteca ha sido optimizada para el hardware en la cual se ejecuta).

Con MPI el número de procesos requeridos se asigna antes de la ejecución del programa, y no se crean procesos adicionales mientras la aplicación se ejecuta. A cada proceso se le asigna una variable que se denomina *rank*, la cual identifica a cada proceso, en el rango de 0 a p-1, donde p es el número total de procesos. El control de la ejecución del programa se realiza mediante la variable *rank*, que también permite determinar qué proceso ejecuta determinada porción de código.

Las características de MPI son las siguientes:

- Estandarización.
- Portabilidad: multiprocesadores, multicomputadores, redes, clústeres heterogéneos, etc.
- Buenas prestaciones.
- Amplia funcionalidad.
- Existencia de implementaciones libres (mpich, LAM-MPI, ...).

Por todo lo comentado anteriormente y sabiendo que el algoritmo de blindaje va a procesar una gran cantidad de simulaciones de fotones (más de 1 millón), parece conveniente hacer uso de este estándar de comunicación entre procesadores.

5. Algoritmo de blindaje

En los apartados anteriores se ha hablado de los principios físicos básicos en los que se basa el algoritmo así como en métodos utilizados en su implementación (método de Monte Carlo y MPI). En este apartado se comentarán las características y desarrollos del algoritmo de blindaje.

5.1. Diagrama de flujo

En la Figura 9 se muestra el diagrama del flujo del algoritmo, sin tener en cuenta la paralelización. Se muestra el caso más completo en el que se tienen dos esferas concéntricas de distintos materiales y se considera la dispersión Rayleigh.

A pesar de que el diagrama de flujo se explica por sí solo a continuación se especifican los pasos seguidos de manera más extensa:

- El programa se inicia con unos parámetros de entrada elegidos por el usuario. Estos parámetros son: número de fotones, energía inicial de los fotones, material de la esfera 1, radio de la esfera 1, material de la esfera 2, radio de la esfera 2 y un parámetro para considerar o no la dispersión Rayleigh.
- El programa procesa estos parámetros y "crea" el sistema a partir de ellos.
- Hace un procesamiento de todos los fotones indicados, cada uno por separado. Este procesamiento es el siguiente:

    1. Se realiza el procesamiento si la distancia recorrida por el fotón es menor que el radio de la esfera externa y la energía es mayor de 1 keV y no ha sufrido efecto fotoeléctrico.
    2. En caso de cumplirse la condición anterior y si la distancia recorrida es menor que el radio de la esfera interna (esto es así siempre en la primera iteración), se calcula el efecto del fotón dentro de la esfera 1 (usando su número atómico). En caso contrario el fotón se encuentra en la esfera 2 y por tanto hay que calcular el efecto para dicha esfera.
    3. El efecto se calcula mediante probabilidades usando el método Monte Carlo, a partir de la generación de un número aleatorio y de las secciones eficaces del material para la energía del fotón.
    4. Después se calcula mediante números aleatorios y coordenadas esféricas el radio recorrido por el fotón.
    5. Dependiendo del efecto se realizan distintos procesamientos:
        a. En el caso de efecto fotoeléctrico se comprueba que se haya realizado dentro de las esferas. Si es así el fotón ha sido absorbido y en caso contrario ha salido de las esferas sin

perder energía. En ambos casos se acaba el procesamiento del fotón.
   b. En el caso de efecto Rayleigh, el fotón no pierde energía.
   c. En el caso de efecto Compton, se calcula la nueva energía del fotón con la fórmula vista en el apartado 2 a partir del ángulo hallado aleatoriamente.
6. Se realiza de nuevo el paso 1 con la nueva distancia y energía del fotón. Si no se cumple la condición del paso 1 se acaba el procesamiento de dicho fotón.

- Una vez finalizado el procesamiento de todos los fotones, se realiza el espectro energético de los fotones que han salido de las esferas. Para ello se suman todos los fotones que tengan una determinada energía, obteniendo por tanto el número de fotones de cada energía en el rango de energías de 0 a la energía inicial.
- Finalmente se calcula el blindaje con la siguiente ecuación:

$$Blindaje = \frac{nº\ fotones\ totales - nº\ fotones\ con\ energía\ inicial}{nº\ fotones\ totales}$$

considerando como fotones con energía inicial aquellos que salen de la esfera con una energía igual a la inicial. Se ha tomado esta fórmula de blindaje por simplicidad para obtener resultados cuantitativos que se puedan comparar.

Como se ha comentado, este algoritmo no tiene en cuenta la paralelización, pero puede valer perfectamente para el algoritmo paralelizado, puesto que lo que se hace en la paralelización es repartir los procesamientos de los fotones por separado entre los procesadores disponibles, de modo que cada uno lleve a cabo el mismo número de cuentas (o parecidas). Por ejemplo, supóngase que hay 500000 fotones a procesar y 6 procesadores, cada uno llevará a cabo $500000/6 = 83333$ cuentas excepto uno que realizará 83335 para completar todos los fotones. Así se consigue dividir la carga de trabajo equitativamente entre los procesadores. Después de realizar las cuentas un procesador se encargará de juntar todos los resultados para obtener el resultado final. En un apartado posterior se mostrará la mejora de tiempos al paralelizar el programa.

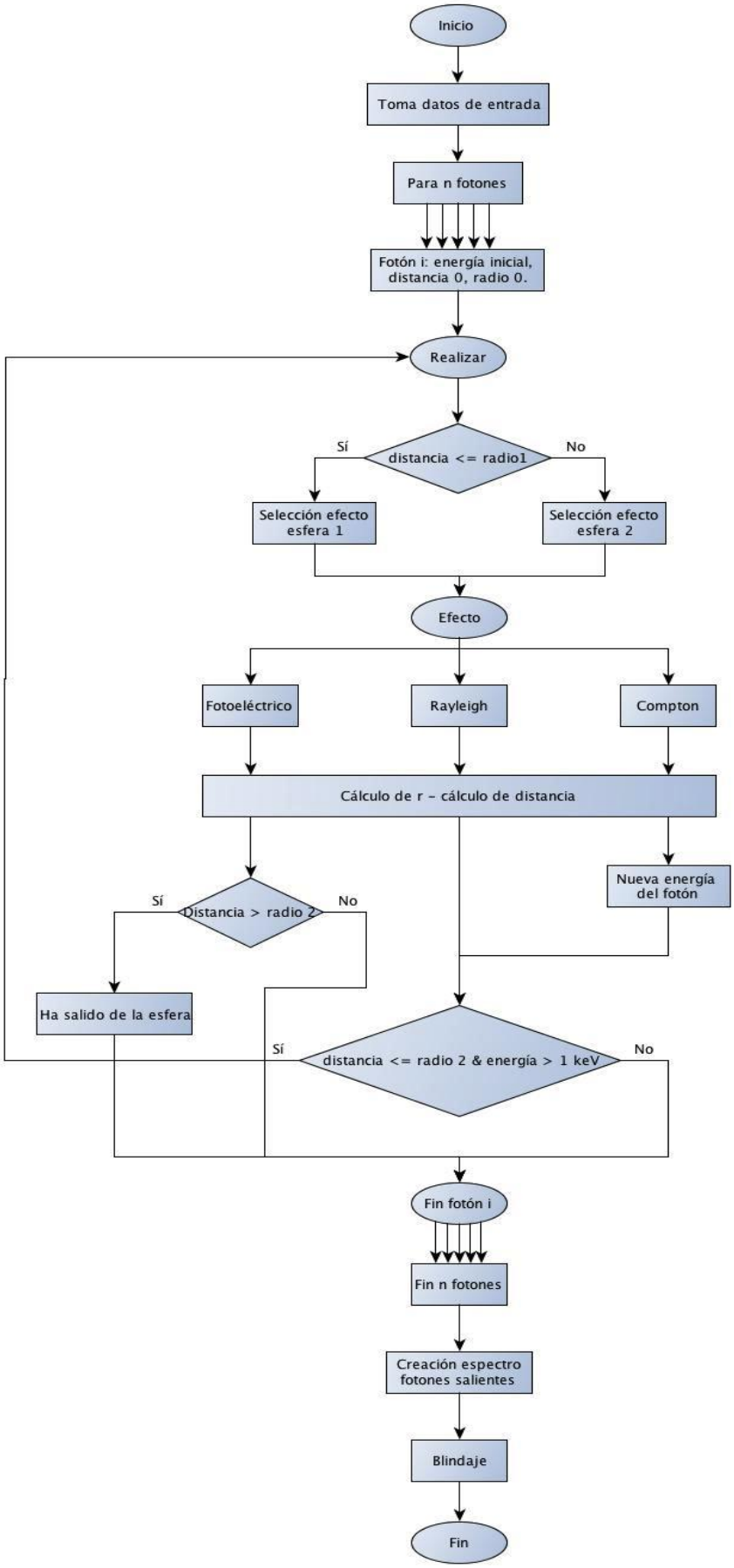

Figura 9. Diagrama de flujo del algoritmo de blindaje.

5.2. Código del programa

El algoritmo se desarrolló en un programa informático usando lenguaje C++. El código completo se encuentra en el Apéndice A de este documento. En este apartado se comentan algunas características importantes del código.

La entrada de parámetros se realiza a la hora de ejecutar el programa introduciéndolos mediante argumentos. Si no se realiza bien dicha introducción se indica qué se ha introducido mal o bien se indican las instrucciones de uso. Sólo un procesador realiza la lectura de dichos parámetros y las envía al resto de procesadores. Al ejecutar el programa también ha de introducirse el número de procesadores que lo ejecutarán, mediante el comando *mpirun –np num_procesadores*. También se puede ejecutar en uno o varios nodos concretos mediante *mpirun –np num_procesadores –hostfile fichero_nodos*, donde el fichero fichero_nodos contiene los nombres de los nodos y el número de procesadores que tienen, por ejemplo:

nodo01.dominio slots=16 max_slots=16
nodo02.dominio slots=16 max_slots=16
nodo03.dominio slots=16 max_slots=16

A partir del material de las esferas introducido se obtiene la densidad volumétrica del mismo guardadas en un mapa y también se carga el fichero de secciones eficaces del material para energías comprendidas entre 0 y 1 MeV. Este fichero se guarda en un vector de vectores. Esta operación se realiza en todos los procesadores. En el Apéndice B están los ficheros de las secciones eficaces de los materiales usados: ioduro de sodio, plomo y óxido de uranio. Estos archivos se obtuvieron de una página web del NIST [6], que es la base de datos XCOM: *Photon Cross Sections Database* .

Antes de iniciar el procesamiento de los fotones se realiza la división de los mismos entre los procesadores introducidos. El método es el indicado en el apartado anterior: se dividen los fotones por el número de procesadores, cada uno hace el número entero de la división anterior y el último los fotones sobrantes hasta completar el número total.

La función que halla el efecto sufrido por el fotón tiene en cuenta la energía del fotón, la esfera donde está y si se ha considerado el efecto Rayleigh. Primero se buscan las secciones eficaces de cada efecto para la energía dada. En caso de no tener la energía justa se hace el promedio de las energías entre las que se encuentre. Si se da el caso de que una misma energía tiene distintas secciones eficaces para el efecto fotoeléctrico (debido a las capas orbitales de los electrones), entonces se toma la de mayor sección eficaz. Una vez halladas las secciones eficaces se calcula la probabilidad de cada efecto dividiendo la sección eficaz del mismo entre la suma de todas. El número aleatorio generado indicará el efecto obtenido: Supongamos que sólo hay dos efectos, fotoeléctrico y Compton, cuyas probabilidades son a y 1-a, respectivamente, y que el número aleatorio generado es b. Si b está dentro del intervalo [0, a), entonces el efecto obtenido será el efecto fotoeléctrico y si b está dentro de [a, 1] entonces el efecto será el efecto Compton.

Para hallar el recorrido hecho por el fotón se usan números aleatorios para obtener las 3 componentes de las componentes esféricas (r, θ, Φ). Para θ el número aleatorio se desplaza al intervalo (0, π) y para Φ se hace lo mismo con (0, 2π). En el caso de r, se calcula el camino libre promedio con la fórmula indicada en el apartado 2:

$$\lambda = \frac{1}{\sigma(E)\rho}$$

y el camino recorrido por el fotón desde su última posición rantigua es Δr = -λlog(número aleatorio) y la nueva coordenada r es $r_{antigua}$+Δr.

Cuando acaba el procesamiento de los fotones se deja de usar la computación en paralelo, puesto que es el primer procesador el que realiza la unión de todos los resultados enviados por el resto y halla el espectro y la tasa de blindaje. Ya se comentó que la tasa de blindaje es la siguiente:

$$Blindaje = \frac{nº\ fotones\ totales - nº\ fotones\ con\ energía\ inicial}{nº\ fotones\ totales}$$

Se lleva a cabo una medición del tiempo que tarda cada procesador en realizar su parte del programa y se muestra el tiempo por pantalla al terminar los cálculos, junto con la tasa de blindaje.

Para hallar el espectro energético se suman todos los fotones de una energía dada. Para ello se guarda el espectro en un mapa donde el índice es la energía y el segundo dato es el número de fotones. El procedimiento es el siguiente: se va recorriendo el vector con todos los fotones salientes y la energía de cada uno se redondea a 3 decimales. Después se busca en el mapa del espectro esa energía de modo que si se encuentra esa energía se suma un fotón más al segundo dato, el número de fotones. Si no se encuentra se crea una nueva entrada en el mapa con la energía y un 1 de segundo dato (un fotón). Al finalizar el recorrido estará guardado en el mapa el espectro completo. El último paso es crear un fichero de texto con ese mapa. Para poder verlo hay que usar un programa de visualización de gráficos.

Por último, para que los resultados de los cálculos sean diferentes en los distintos procesadores, la semilla de los números pseudo-aleatorios se hizo que dependiera del número de procesador. Sin esta inicialización diferencial de las semillas, los resultados en cada uno de ellos se hubieran repetido, puesto que las series de números hubieran sido iguales en todos ellos.

5.3. Desarrollo y comprobación del programa

El desarrollo del programa se llevó a cabo en un *MacBook Air* con las siguientes características:

- Procesador *Intel Core* i5 a 1.7 GHz.
- Memoria RAM de 4 GB a 1333 MHz DDR3.

- Disco duro de 121 GB de almacenamiento.
- Gráficos *Intel HD Graphics 3000* con 384 MB de memoria.
- Sistema operativo *Mac OS X* versión 10.7.4.

Para la escritura del programa se hizo uso del software *Xcode*, con las librerías de MPI previamente instaladas.

El programa final no se realizó directamente, sino que se hizo en varios pasos:

- En primer lugar se realizó un programa que realizara el algoritmo simulando una esfera de ioduro de sodio (NaI) de 8 cm y una fuente de ^{137}Cs en el centro de la misma. Asimismo, se obtuvo el espectro energético de los fotones absorbidos en la esfera en vez de fuera. Con esto se intentó simular un detector de NaI, cuyo espectro es conocido, para comprobar que el espectro que se obtenía del programa era igual que el real de dichos detectores y, por tanto, comprobar el correcto funcionamiento del algoritmo. Este programa inicial de simulación del detector de NaI se encuentra en el Apéndice A.
- En segundo lugar, se llevaron a cabo algunas pruebas con las llamadas a funciones de las librerías de MPI, a modo entrenamiento, realizando pequeños programas para tal fin.
- En una fase posterior se modificó el programa de simulación del detector de NaI para el algoritmo final con una sola esfera sin paralelizar. Se programó la entrada y captura de los parámetros introducidos por el usuario y se cambió el mapa del espectro para tomar los fotones salientes y no los absorbidos en la esfera.
- Se paralelizó el programa introduciendo la librería MPI y sus llamadas.
- Se añadió una segunda esfera al algoritmo.
- Por último se repasó el código corrigiendo fallos y optimizando el tiempo.

6. Resultados y simulaciones

Una vez finalizado el programa el último paso y más importante es la obtención de simulaciones variando parámetros de dicha simulación. Estas simulaciones se llevaron a cabo en el ordenador mencionado y en el clúster larisa de la Facultad de Ciencias de la Universidad de Valladolid. Este ordenador tiene las siguientes características:

- Está formado por 14 nodos esclavos y un nodo principal o maestro.
- Disco duro: 920 GB el nodo principal, 10 nodos esclavos de 795 GB y cuatro de 860 GB.
- Procesadores: El nodo principal tiene ocho procesadores *Intel(R) Xeon(R) CPU E5410* a 2.33GHz, cuatro nodos esclavos tienen 16 procesadores *Intel(R) Xeon(R) CPU E7340* a 2.40GHz, seis nodos tienen 16 procesadores *Intel(R) Xeon(R) CPU X5550* a 2.67GHz y cuatro nodos tienen 24 procesadores *Intel(R) Xeon(R) CPU X5650* a 2.67GHz.
- Memoria RAM: 7982 MB el nodo principal, 80517 MB cuatro nodos esclavos, 96676 MB cuatro nodos esclavos y 48289 MB seis nodos esclavos.

- Todos los nodos usan el mismo sistema operativo: *Red Hat Enterprise Linux Server release 5.3.*

Aunque el programa puede realizar simulaciones con fotones de energía inicial en un rango entre 0 y 1 MeV, se simuló una fuente monoenergética de ^{137}Cs que emite fotones gamma a 662 keV, por lo que en las simulaciones no se cambió esta energía inicial. Asimismo, el programa puede simular esferas de Uranio, pero tampoco se hicieron simulaciones con este material.

6.1. Simulación de un detector de NaI

En primer lugar se simuló el programa de prueba que hacía la función de un detector de NaI. Este programa se simuló sólo en el *MacBook Air* y no está paralelizado.

Se usa un millón de fotones para la simulación, un radio de la esfera de 8 cm y una energía inicial de 662 keV, simulando una fuente de cesio, como ya se ha comentado. No se usa un número mayor de fotones debido a la limitada capacidad de memoria RAM del ordenador.

En la Figura 10 se muestra el espectro resultante de la simulación. En dicha figura se puede observar el fotopico a 0.662 MeV y el borde Compton a 477,65 keV. Estos datos y también la forma del espectro coinciden con los de la Figura 8. En la simulación no existe pico de retrodispersión dado que no hay un material rodeando al detector. Se observa también que alrededor de 600000 fotones han sido absorbidos mediante efecto fotoeléctrico del millón inicial. En comparación, el número de fotones absorbidos que han sufrido efecto Compton es mucho menor. En la Figura 11 se muestra una ampliación del espectro enfocado en la zona de la dispersión Compton para poder ver mejor esta dispersión.

Si se cambia el radio de la esfera a la mitad, 4 cm, se obtiene el espectro de la Figura 12, donde se observa perfectamente que ha disminuido el número de fotones fotoabsorbidos en la esfera. El número de fotones absorbidos que han sufrido dispersión Compton también es menor que en el caso anterior.

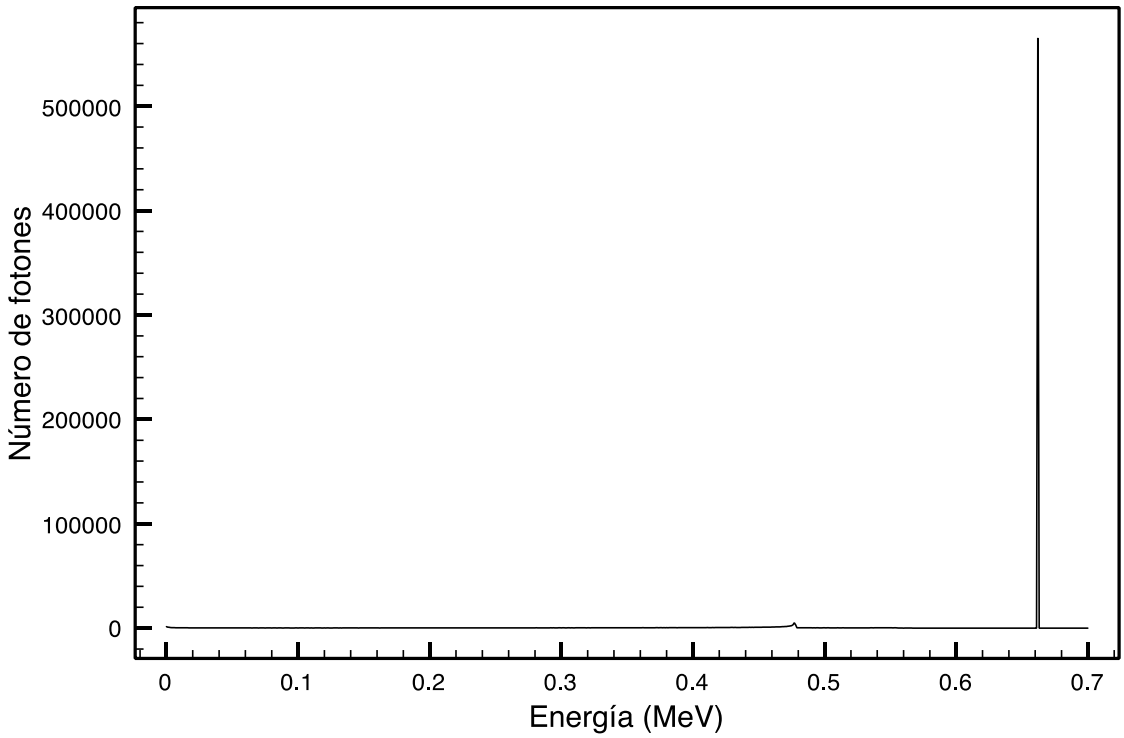

Figura 10. Simulación del espectro energético de los fotones en el interior de una esfera de 8 cm de radio de NaI.

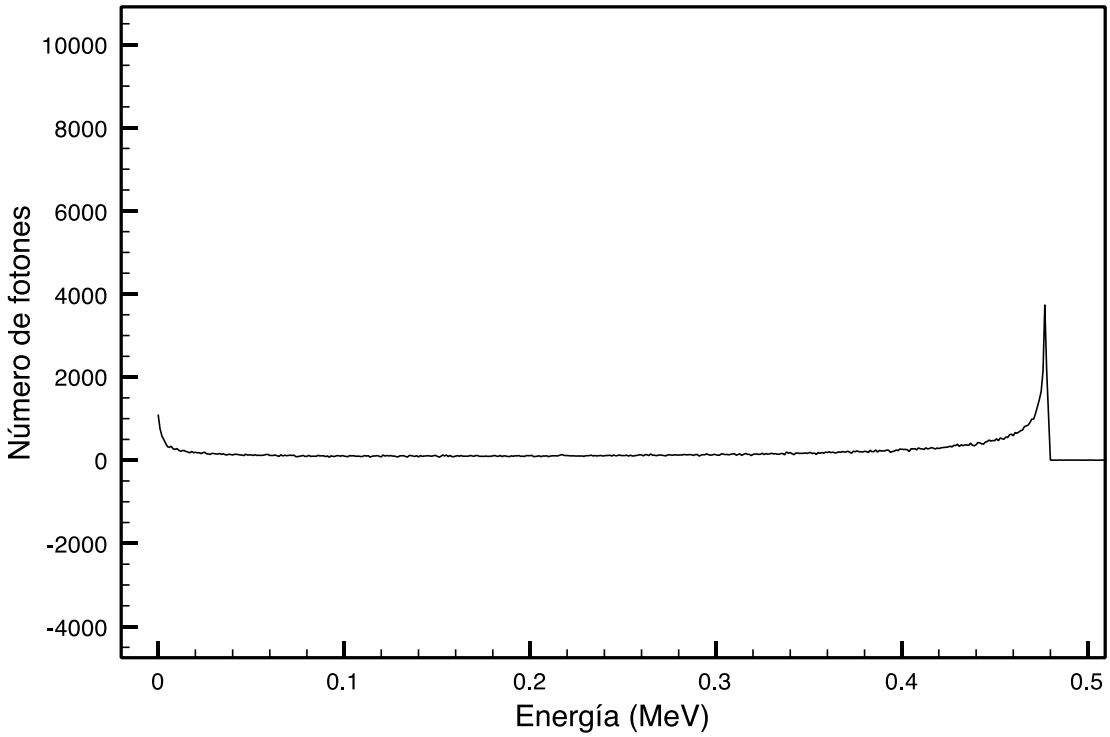

Figura 11. Ampliación en la zona de la dispersión Compton del espectro energético de los fotones en el interior de una esfera de 8 cm de radio de NaI.

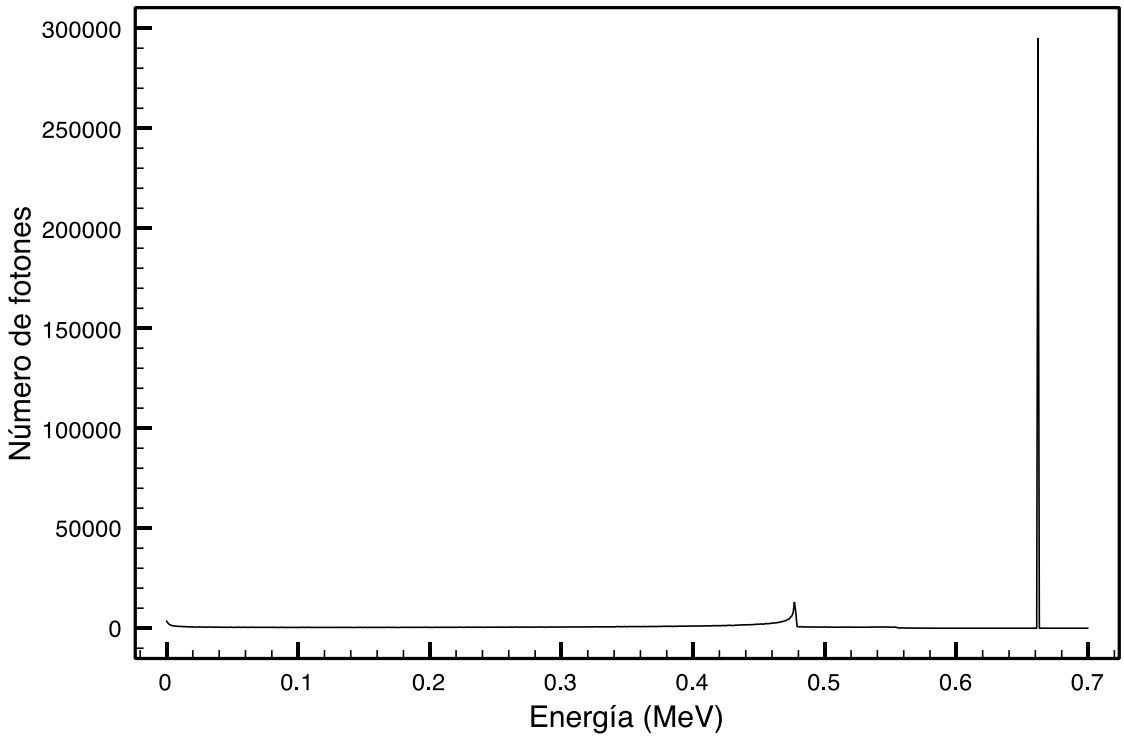

Figura 12. Simulación del espectro energético de los fotones en el interior de una esfera de 4 cm de radio de NaI.

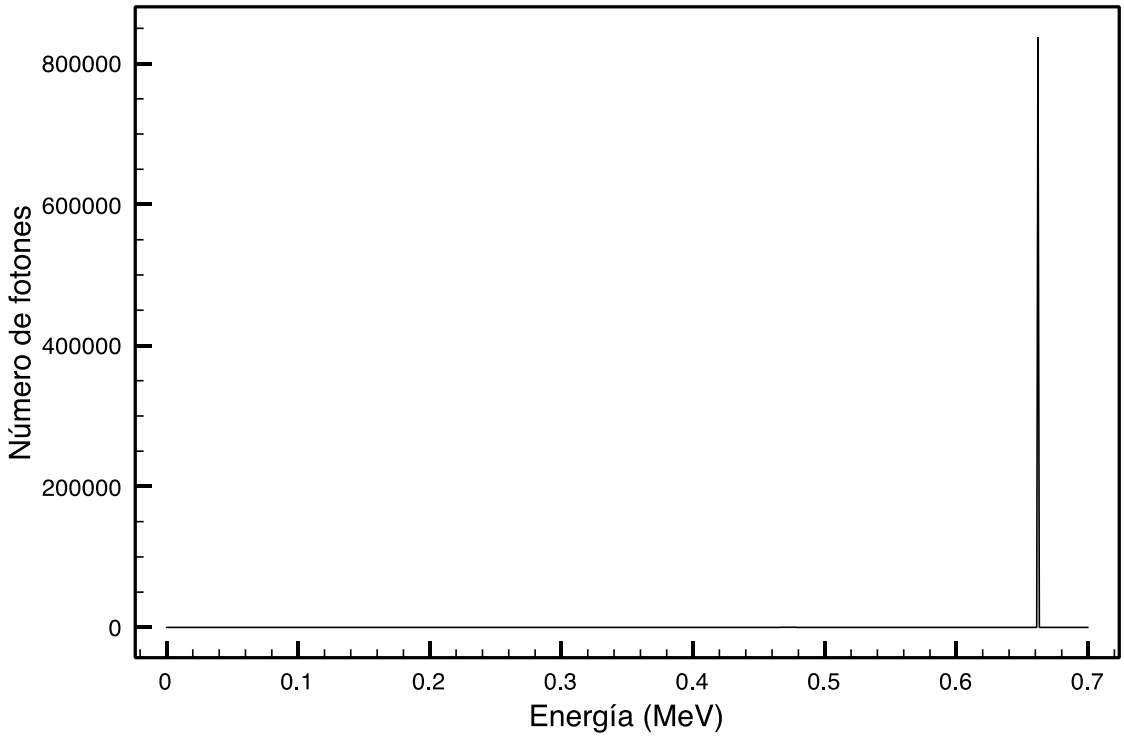

Figura 13. Simulación del espectro energético de los fotones en el interior de una esfera de 20 cm de radio de NaI.

En cambio, si se aumenta el radio de la esfera a 20 cm (Figura 13), se observa que aumenta el número de fotones absorbidos por efecto fotoeléctrico y también los

que han sufrido dispersión Compton. Más de 800000 fotones han sido fotoabsorbidos, que contrasta con los 600000 del caso de la esfera de radio de 8 cm y los 300000 de la esfera de 4 cm. Este aumento del fotopico con el radio o tamaño del detector de NaI es el resultado que esperábamos.

A la vista de los resultados, queda comprobado el correcto funcionamiento del algoritmo, por lo que se pudo pasar a la siguiente fase de desarrollo y completar el programa y la paralelización del mismo.

6.2. Simulación de una esfera de plomo

Para el resto de simulaciones se usó el programa final en el clúster larisa. Dada su gran capacidad de memoria RAM, se pudieron usar un número mayor de fotones en la simulación, lo que aumenta la precisión de la misma. Con el algoritmo se está simulando una muestra de ^{137}Cs que tiene una cierta actividad. La actividad es el número de fotones que emite por segundo y se calcula como el producto del número de emisores y la constante de desintegración λ. Esta constante es:

$$\lambda = \frac{ln2}{t_{1/2}}$$

donde $t_{1/2}$ es la semivida [10], que es el lapso de tiempo necesario para que se desintegren la mitad de los núcleos de una muestra inicial de una sustancia radiactiva. La semivida del ^{137}Cs es 30.17 años. Un gramo de ^{137}Cs tiene

$$\frac{1}{136.907}\ NAvogadro\ =\ 4.4 * 10^{21}\ núcleos\ de\ Cs.$$

Por tanto, la actividad de un gramo de ^{137}Cs es de 3.2 10^{12} desintegraciones/segundo o fotones/segundo.

Teniendo en mente esto, lo ideal hubiera sido realizar una simulación del orden de la actividad del ^{137}Cs, pero teniendo en cuenta el compromiso entre tiempo y precisión de la simulación se decidió realizar las simulaciones con 100 millones de fotones, puesto que de esta manera una ejecución del programa para una esfera de plomo tarda unos 124-143 segundos. Además, después de varias pruebas, con este número de fotones se obtiene una precisión muy alta, con un error de sólo un 0.01 en el blindaje obtenido.

A partir de este número de fotones y cambiando sólo el radio de la esfera de plomo se obtiene la Figura 14 que muestra la gráfica del blindaje sin tener en cuenta la dispersión Rayleigh en función del radio de la esfera y la Figura 15 donde se muestran los tiempos de ejecución en función del radio. En la Figura 14 se observa perfectamente cómo el blindaje aumenta con el radio de la esfera de una manera logarítmica mientras que en la Figura 15 se observa cómo en general los tiempos de ejecución aumentan con el radio. Esto es debido a que cuanto mayor es la esfera mayor es, en promedio, el recorrido o número de pasos de cada fotón dentro de la esfera y, por tanto, mayor es el tiempo de ejecución total del programa. Los picos e

irregularidades que se observan son explicados por la concurrencia de otros procesos simultáneos en el clúster.

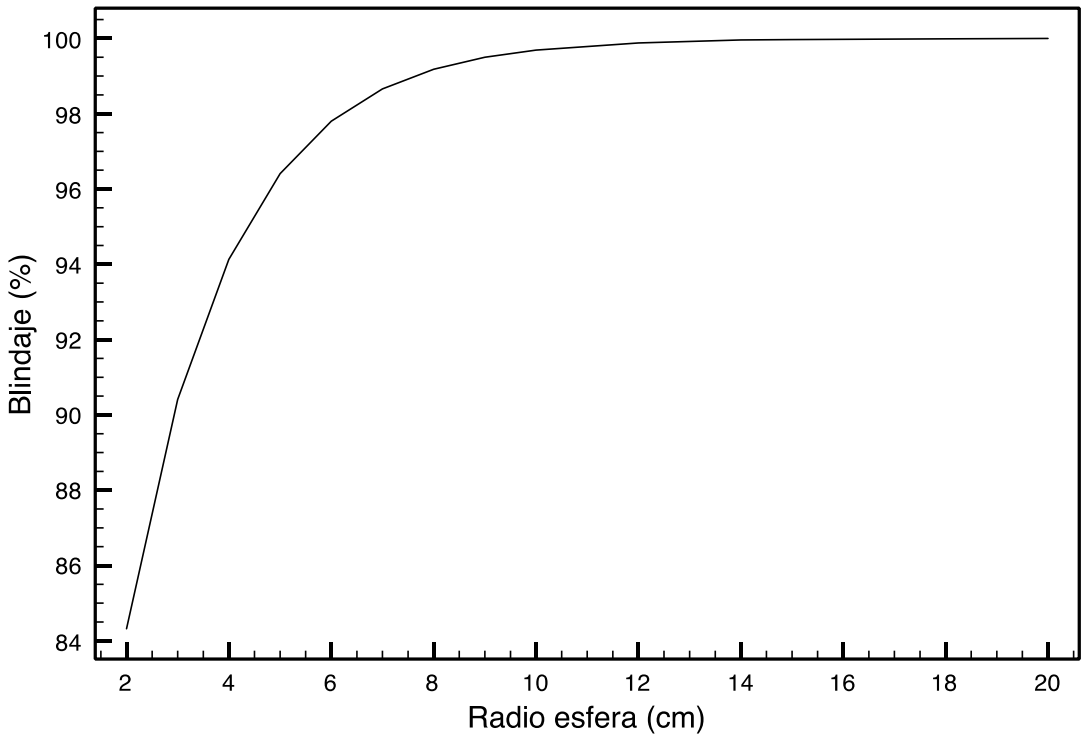

Figura 14. Blindaje en función del radio de una esfera de plomo sin tener en cuenta la dispersión Rayleigh.

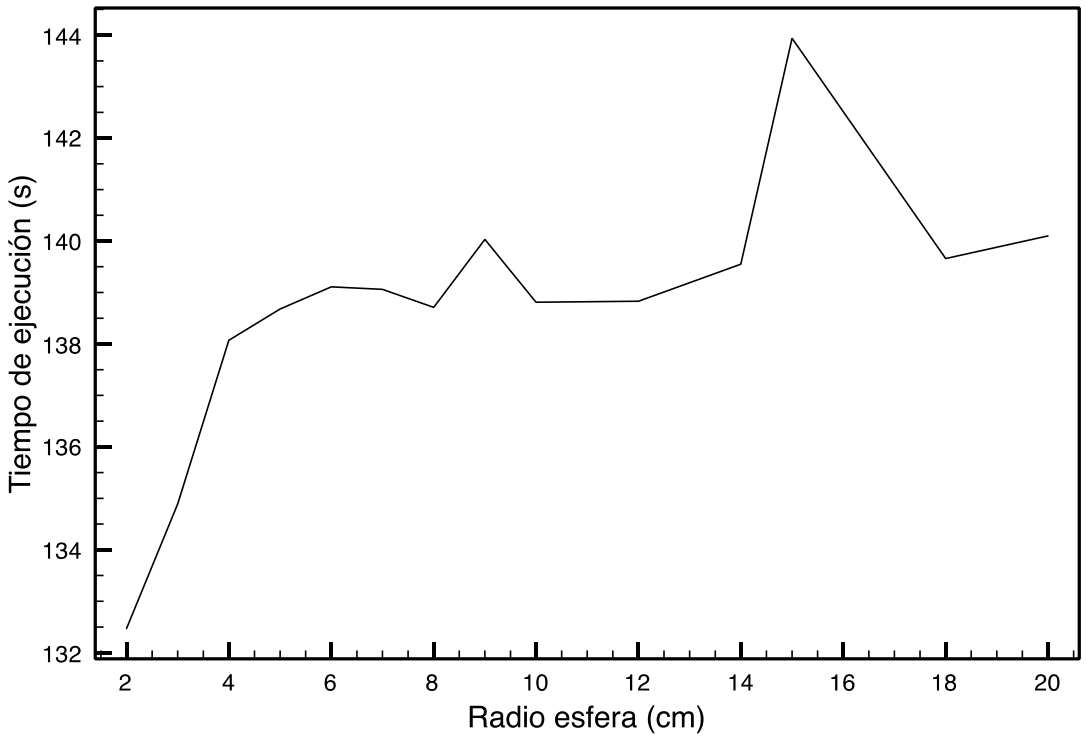

Figura 15. Tiempo de ejecución del programa en función del radio de una esfera de plomo sin tener en cuenta la dispersión Rayleigh.

En la Tabla 1 se muestran los blindajes y tiempos de ejecución obtenidos para ciertos radios teniendo y no teniendo en cuenta la dispersión Rayleigh:

	Con dispersión Rayleigh		**Sin dispersión Rayleigh**	
Radio (cm)	Blindaje (%)	Tiempo ejec. (s)	Blindaje (%)	Tiempo ejec. (s)
1 cm	74.29	122	74.37	124
3 cm	90.31	135	90.41	135
5 cm	96.32	138	96.41	139
8 cm	99.10	141	99.18	141
12 cm	99.81	142	99.88	139

Tabla 1. Blindajes y tiempos de ejecución en función del radio con y sin dispersión Rayleigh.

Si se comparan los blindajes de la misma esfera teniendo y sin tener en cuenta la dispersión elástica, se observa que en el primer caso el blindaje es siempre menor, aunque las diferencias son muy pequeñas:

- En el caso de una esfera de 1 cm la diferencia es de 0.08.
- En el caso de una esfera de 3 cm esta diferencia es de 0.10.
- Si la esfera es de 5 cm de radio la diferencia disminuye a 0.09.
- Si se aumenta el radio a 8 cm la diferencia es de 0.08.

La diferencia entre los blindajes con y sin dispersión coherente es prácticamente constante. Además, esta diferencia es muy pequeña por lo que puede ser omitida esta dispersión a la hora de hacer las simulaciones.

En la Figura 16 se muestra el espectro de la simulación de la esfera de plomo para 2 cm. Se puede ver cómo la gran mayoría de los fotones que escapan lo hacen con la energía inicial, siendo los fotones peligrosos que se tienen en cuenta para el blindaje.

En la Figura 17 se muestra el espectro de la misma simulación para un radio de 6 cm. Comparándola con la Figura 16 se puede ver que la cantidad de fotones peligrosos ha disminuido considerablemente. Por último, en la Figura 18 se observa un espectro de una esfera de 12 cm de radio.

Al aumentar el grosor aumenta el blindaje y por tanto disminuye la cantidad de fotones peligrosos que escapan de la esfera. Se ha pasado a tener más de $1.6*10^7$ fotones peligrosos que escapan de una esfera de 2 cm de radio a tan solo 300000 en una esfera de 12 cm de radio de los 100 millones totales. De ahí la diferencia de blindajes: 84.32% con 2 cm de radio frente a 99.88% en el caso de 12 cm.

En la Figura 19 se muestran los espectros de las esferas anteriores con el eje Y en escala logarítmica, donde se puede apreciar mejor la diferencia de blindajes al aumentar el radio. La línea azul corresponde al espectro de salida de una esfera de 2 cm, la verde al de una de 6 cm y la roja al de una de 12 cm.

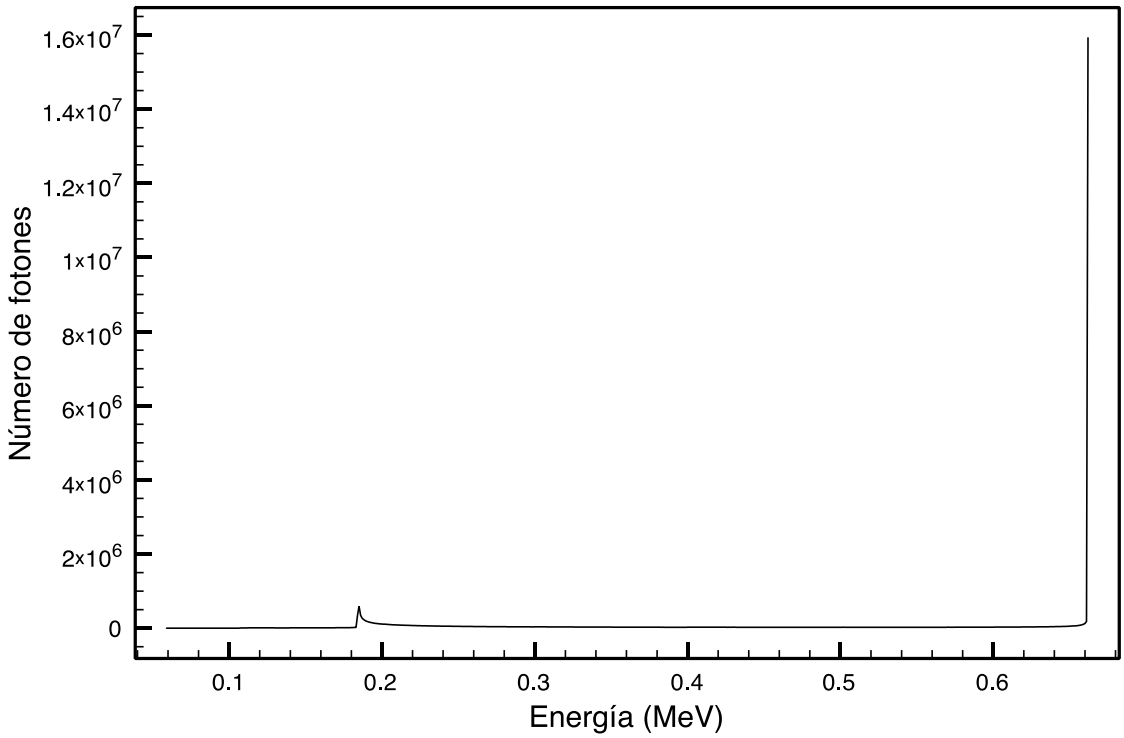

Figura 16. Simulación del espectro energético de los fotones que salen de una esfera de 2 cm de radio de Pb.

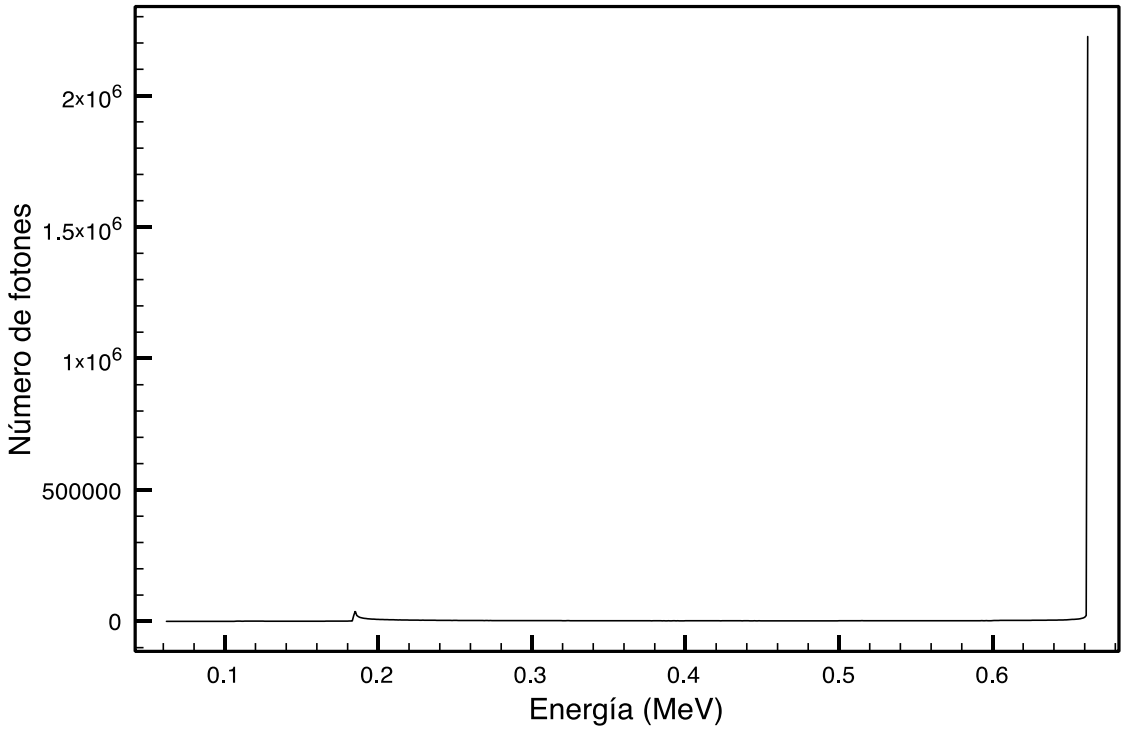

Figura 17. Simulación del espectro energético de los fotones que salen de una esfera de 6 cm de radio de Pb.

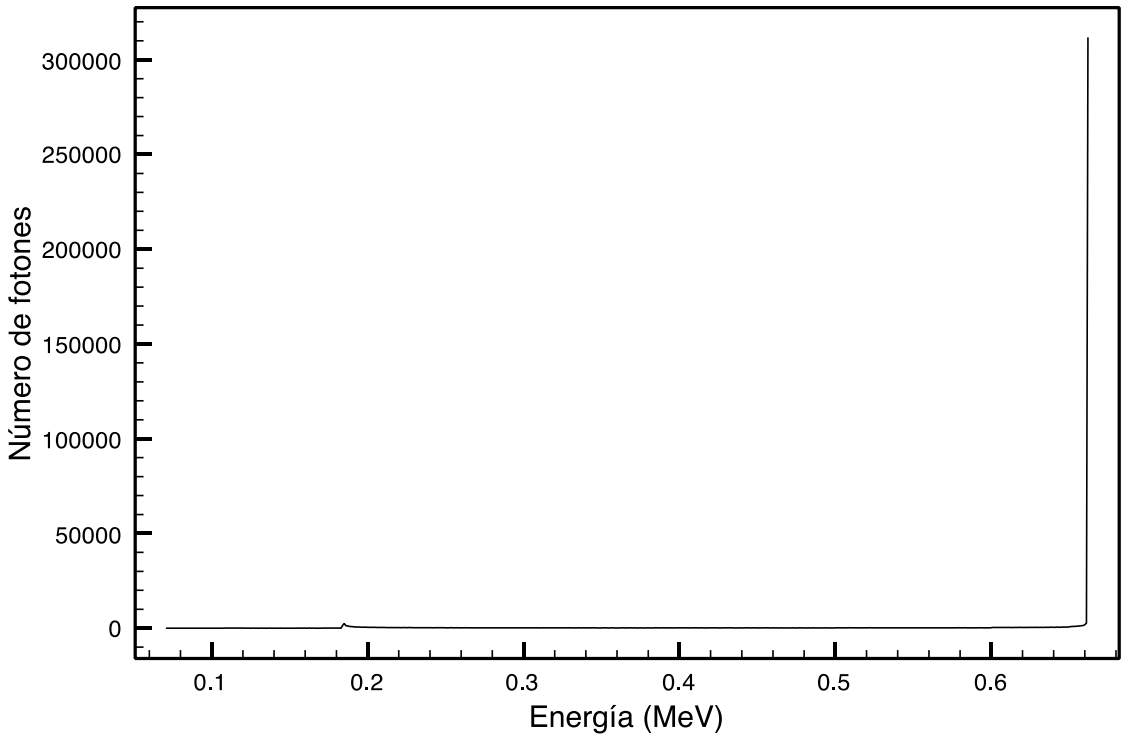

Figura 18. Simulación del espectro energético de los fotones que salen de una esfera de 12 cm de radio de Pb.

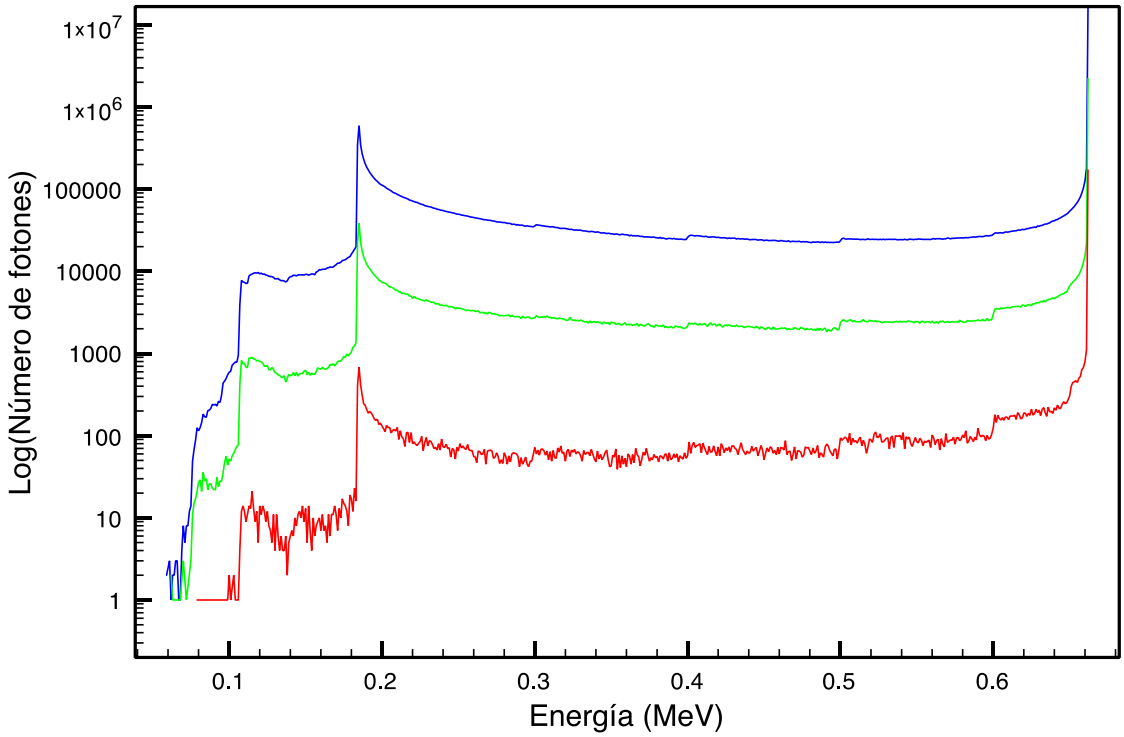

Figura 19. Espectros de salida en escala logarítmica de esferas de plomo de 2 cm (azul), 6 cm (verde) y 12 cm (rojo).

Aparte de los resultados de la simulación del blindaje y del espectro energético, interesa también calcular la mejora de velocidad al paralelizar el programa, así como el grado de paralelización del mismo.

La velocidad S_p es, por definición, T_1/T_P, donde T_1 es el tiempo de computación para un solo procesador y T_P es el tiempo de computación para P procesadores. Por otra parte, la ley de Amdahl [11] predice la mejora máxima esperada dentro de un sistema completo cuando sólo se ha mejorado parte del sistema. Esta ley es muy usada en computación en paralelo para predecir el incremento de velocidad usando múltiples procesadores, lo cual se atañe perfectamente al caso del algoritmo de blindaje. Según esta ley, el incremento de velocidad es el siguiente:

$$S_p = \frac{P}{1 + (P-1)S}$$

Donde S_p es el incremento de velocidad,
P es el número de procesadores,
y S es el porcentaje no paralelizado o no paralelizable del sistema y varía entre 0 y 1.

Usando una esfera de plomo de 6 cm de radio se obtiene para varios procesadores la Tabla 2, donde se muestra el número de procesadores usados, el tiempo de ejecución del programa y el incremento de velocidad.

Nº procesadores	**Tiempo ejecución**	**S_p**
1	4131.74 s	1
2	1853.26 s	2.229
4	954.71 s	4.328
8	492.04 s	8.397
16	399.37 s	10.346
32	206.89 s	19.97
48	139.11 s	29.7

Tabla 2. Tiempos de ejecución y velocidad del programa en función del número de procesadores usados.

Ajustando los datos de la Tabla 2 de S_p a la ley de Amdahl, obtenemos del ajuste un valor de S=0.013, lo que significa que el 98.7 % del programa está paralelizado, un alto grado de paralelización.

6.3. Simulación de una esfera de óxido de uranio

En este apartado se muestran los resultados de la simulación de una esfera de óxido de uranio para distintos radios de la misma. En la Figura 20 se muestra una gráfica con el blindaje de la esfera para distintos radios. El tiempo de computación varía entre 132 segundos para radios y por tanto blindajes bajos y 152 para radios y blindajes altos.

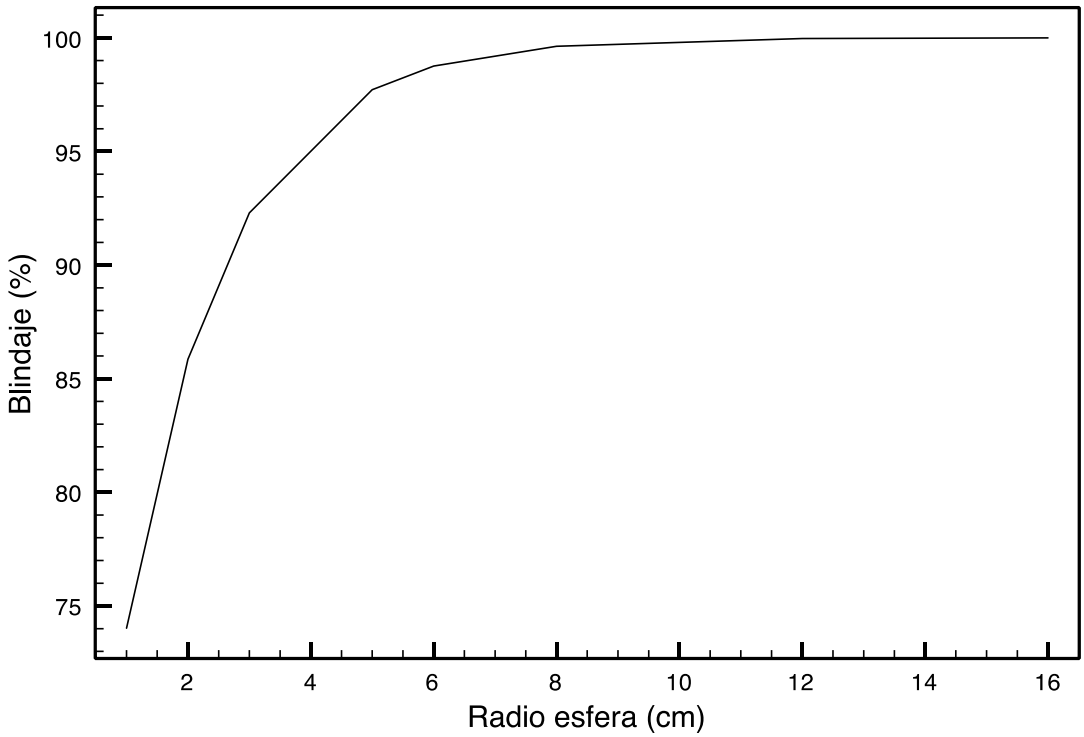

Figura 20. Blindaje en función del radio de una esfera de óxido de uranio sin tener en cuenta la dispersión Rayleigh.

Dado que en el apartado anterior ya se vio cómo el blindaje iba aumentando con el radio de la esfera, en este apartado solamente se muestran algunos radios considerados representativos. Además, como ya se comentó, se ha obviado el efecto Rayleigh debido a su mínima contribución en el blindaje radiactivo.

Lo primero que se observa es que para una esfera del mismo radio, si está constituida por óxido de uranio tiene mejor blindaje que una de plomo, para todos los radios excepto para radios pequeños en torno a 1 cm. Esta diferencia es menos evidente con radios mayores, pero no deja de haber una mejora en blindaje al usar óxido de uranio frente a plomo para radios mayores de 1 cm. Es lógico que el blindaje sea mayor para óxido de uranio que para el plomo puesto que las secciones eficaces para aquel son mayores que para éste y, a pesar de que la densidad volumétrica es mayor para el plomo (11.34 g/cm^3 frente a 10.97 g/cm^3), en la fórmula del camino libre promedio pesan más las diferencias entre secciones eficaces que la densidad de los materiales.

Lo comentado anteriormente choca de forma directa con lo observado en el radio de 1 cm: ¿Por qué para radios en torno a 1 cm es mejor el blindaje en una esfera de plomo? La explicación radica en el código del programa. Cuando se procesa un fotón se calcula su efecto y la distancia a la que se ha producido. Si se da el caso de sufrir un efecto fotoeléctrico pero a una distancia mayor que el radio de la esfera es lo mismo que decir que el efecto se ha producido fuera de la misma. Por tanto, el programa considera que el fotón ha salido de la esfera sin perder energía. Haciendo las cuentas, para una energía inicial de 662 keV, se obtiene que hay una probabilidad del 41.88% de que se produzca un efecto fotoeléctrico en el plomo frente a un 47.7% de que se produzca en el óxido de uranio. Además, para el plomo

el camino libre promedio es de 2.04 cm mientras que para el óxido de uranio es de 1.64 cm para la energía inicial. A la luz de estos resultados, para radios menores de 1.64 cm el programa obtiene un mejor blindaje para el plomo que para el óxido de uranio.

Otro aspecto importante es el aumento del tiempo de ejecución con respecto al caso de la esfera de plomo. Esto se debe a que el fichero de secciones eficaces del UO_2 tiene más entradas que el del plomo, por lo que en el vector se guardan más componentes que van a producir más iteraciones a la hora de obtener la sección eficaz de una energía, con el consiguiente aumento del tiempo de ejecución total del programa. Se sigue observando el aumento del tiempo de ejecución con el aumento del radio, como ya se comentó.

En la Figura 21 se muestran los espectros en escala logarítmica de esferas de óxido de uranio de radios 2, 6 y 12 cm. Como en el caso anterior se observa perfectamente cómo según se va aumentando el radio de la esfera disminuye el pico en la energía inicial y por tanto aumenta el blindaje. Saliendo de la esfera de 2 cm de radio hay más de $1.4*10^7$ fotones peligrosos, en la de 6 cm salen más de $1.2*10^6$ y en la de 12 cm salen menos de 35000. De hecho, si se compara esta última esfera con una de 12 cm de plomo, se observa que de esta última salen 300000 fotones peligrosos, unos 8.5 veces el número de fotones que salen de la de óxido de uranio. A pesar de la diferencia en las alturas de los picos, la forma del espectro es la misma que en el caso del plomo, dado que es característica de la radiación gamma.

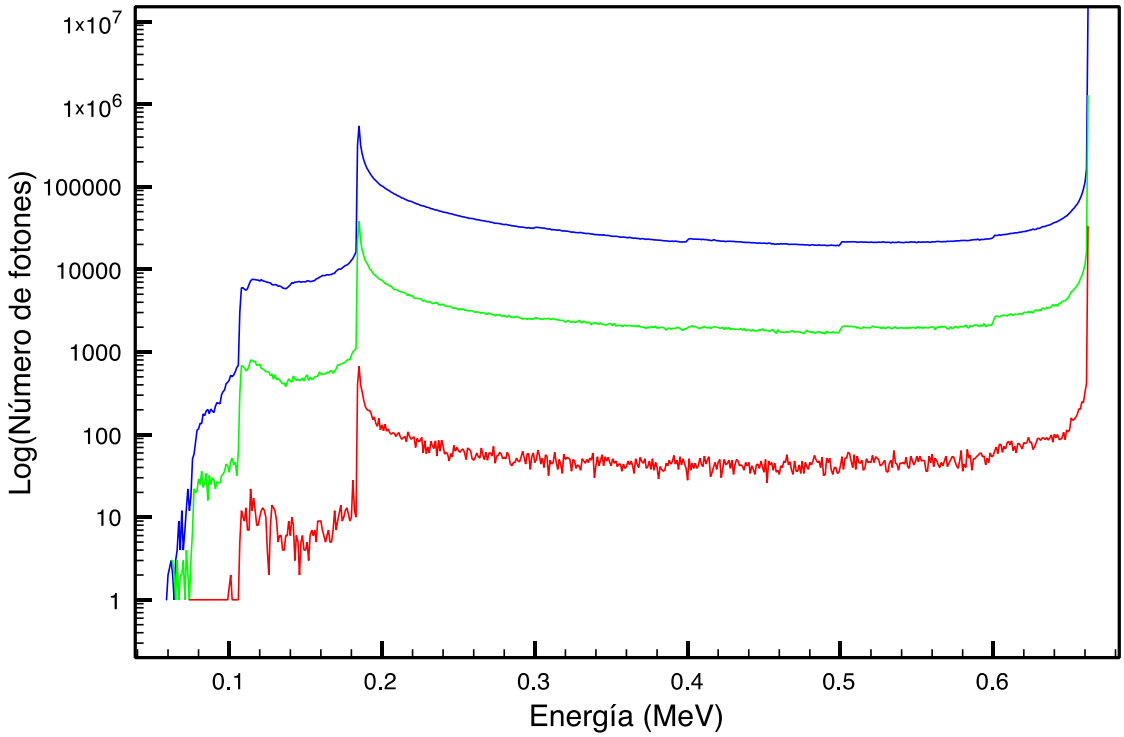

Figura 21. Espectros de salida en escala logarítmica de esferas de óxido de uranio de 2 cm (azul), 6 cm (verde) y 12 cm (rojo).

6.4. Simulación de una esfera de plomo y óxido de uranio

Si se simula una doble esfera con una esfera interna de plomo y una externa de óxido de uranio se obtienen los blindajes de la Tabla 3, escritos en función de los radios de las 2 esferas. Los tiempos de ejecución oscilaron entre 132 y 143 segundos, dependiendo de los radios de las esferas.

Radio esfera interna (cm)	Radio esfera externa (cm)	Blindaje (%)
1	2	84.32
1	6	97.80
1	12	99.88
5	6	97.80
11	12	99.88
3	6	97.80

Tabla 3. Blindajes de una doble esfera de plomo (esfera interna) y óxido de uranio (esfera externa) en función de sus radios.

Estos blindajes son iguales a los blindajes del plomo en el caso de una esfera de radio igual al radio de la esfera externa. Es decir, el blindaje en el caso de una esfera externa de 6 cm por ejemplo es igual al blindaje de una esfera de plomo de 6 cm independientemente del radio de la esfera interna. Esto, que parece no tener lógica, tiene una explicación que radica en la fórmula usada para hallar el blindaje. Esta fórmula, como ya se ha comentado es la siguiente:

$$Blindaje = \frac{nº\ fotones\ totales - nº\ fotones\ con\ energía\ inicial}{nº\ fotones\ totales}$$

En el algoritmo, si no se considera la dispersión elástica, los fotones que salen con energía inicial sólo son los que han sufrido un efecto fotoeléctrico fuera de la esfera, a una distancia mayor al radio de la esfera, puesto que un efecto Compton lleva asociado una pérdida de energía. Al situarse la fuente emisora de fotones en el centro de la esfera interna, el primer efecto se va a calcular usando las secciones eficaces del material de dicha esfera, en este caso el plomo, independientemente del radio de la misma. Por esto, los blindajes son iguales a los obtenidos en la esfera simple de plomo.

Sin embargo, los espectros energéticos dependen de los radios de la doble esfera mixta. En la Figura 22 se muestran los espectros en escala logarítmica de una esfera simple de plomo de 2 cm de radio y una doble esfera de 1 cm de radio la esfera interna de plomo y 1 cm de radio la externa de óxido de uranio. Apenas hay diferencias a energías mayores de 0.2 MeV, aunque a energías menores sí hay alguna diferencia aunque pequeña. Esta semejanza de espectros se debe a que estamos considerando radios de 2 cm y el camino libre promedio del plomo es de 2.04 cm en el caso del efecto fotoeléctrico. Además, los dos materiales son muy parecidos en propiedades.

Si se sigue ampliando la diferencia de radios se observan más diferencias. En la Figura 23 se muestran los mismos espectros pero con radios en la doble esfera de

1+5 cm. Aquí se diferencian 2 líneas distintas a altas y bajas energías. Éstas se diferencian aún más en la Figura 24, donde la doble esfera es de 1+11 cm.

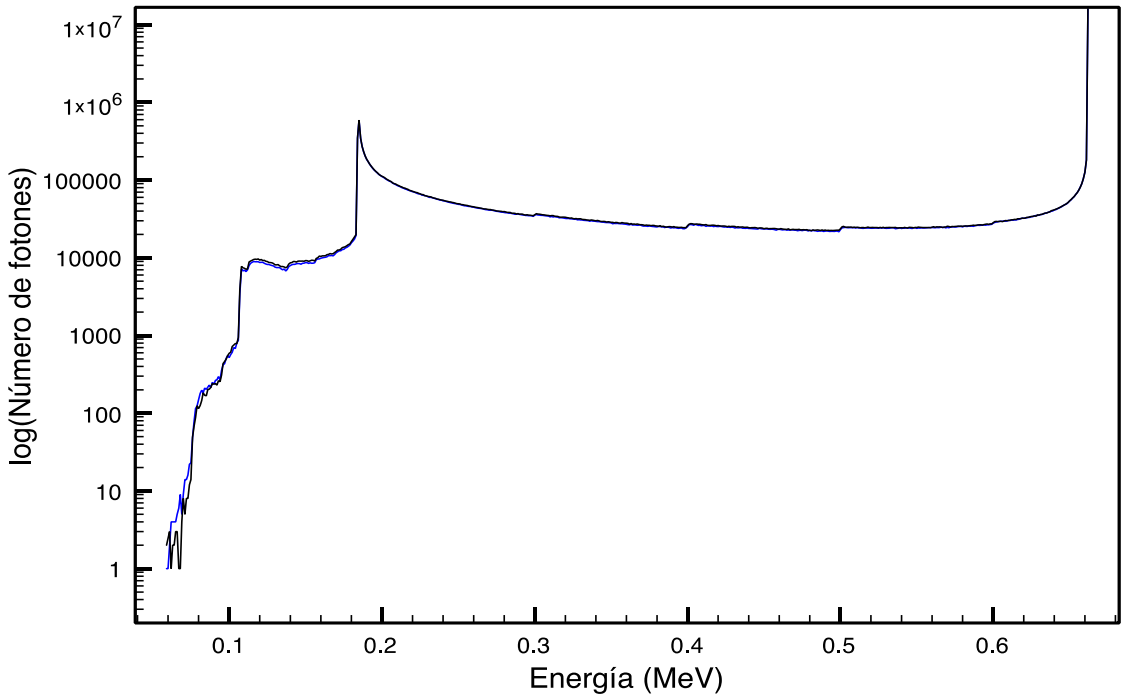

Figura 22. Espectros energéticos en escala logarítmica de una esfera simple de plomo de 2 cm de radio (negro) y de una esfera doble de plomo y óxido de uranio de radios 1+1 cm (azul).

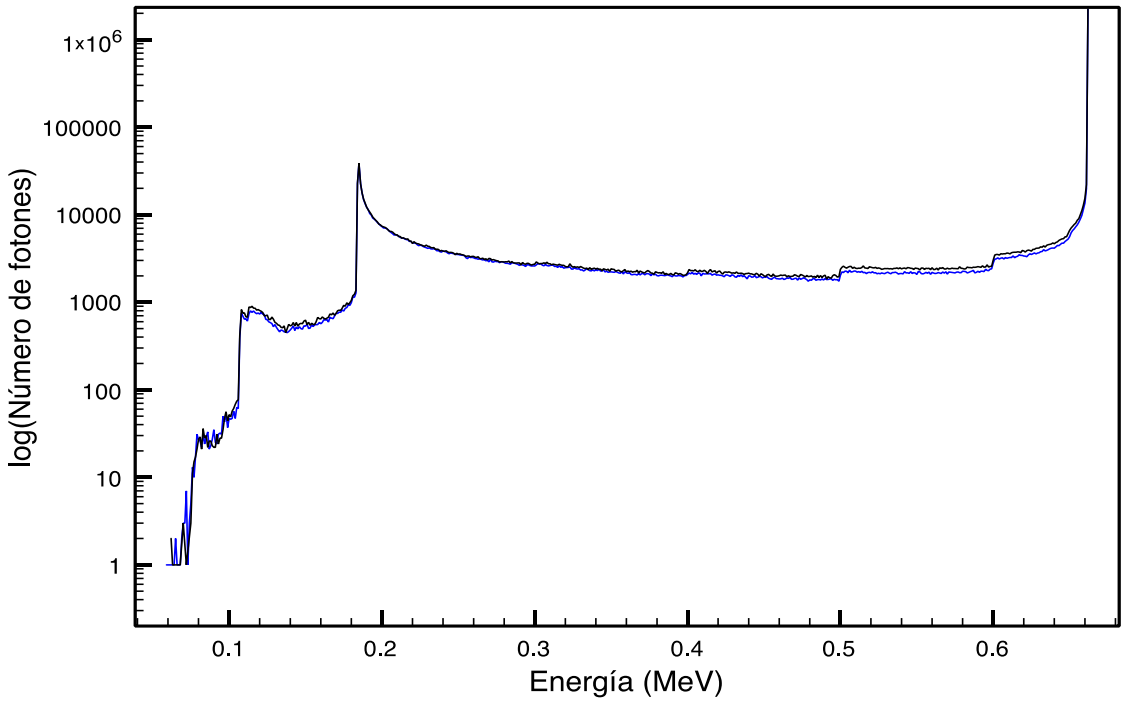

Figura 23. Espectros energéticos en escala logarítmica de una esfera simple de plomo de 6 cm de radio (negro) y de una esfera doble de plomo y óxido de uranio de radios 1+5 cm (azul).

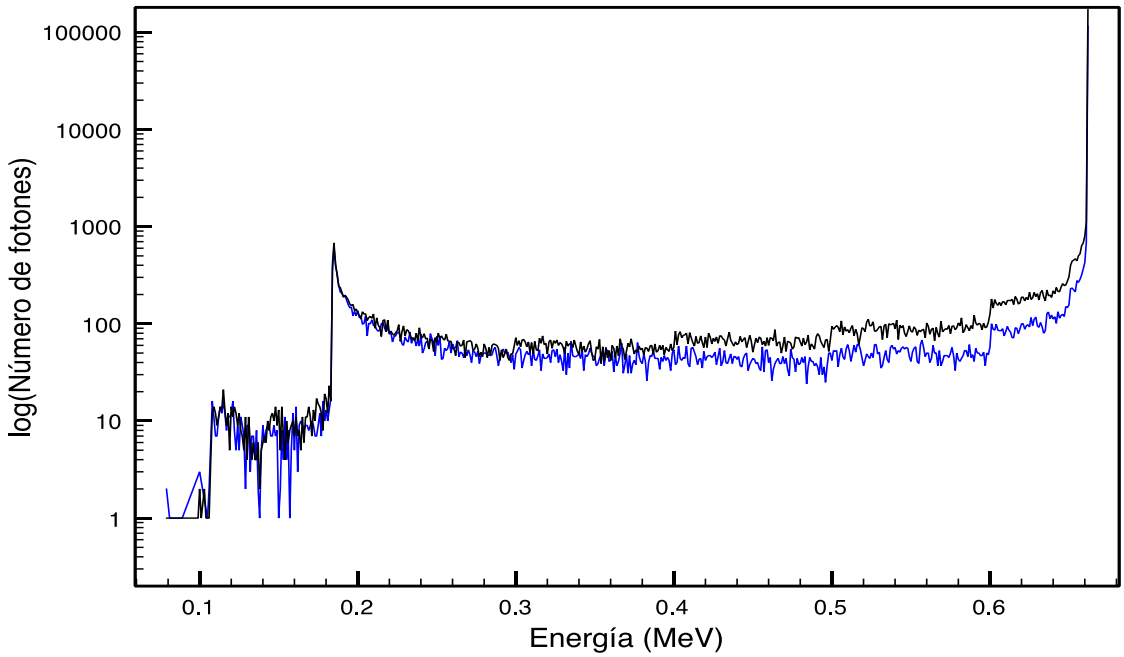

Figura 24. Espectros energéticos en escala logarítmica de una esfera simple de plomo de 12 cm de radio (negro) y de una esfera doble de plomo y óxido de uranio de radios 1+11 cm (azul).

Si además, en el caso de la Figura 23 se añade el espectro de una esfera simple de óxido de uranio de 6 cm de radio se obtiene la Figura 25. En ella se ha hecho una ampliación en ciertas frecuencias para una mejor visualización de cómo el espectro de la doble esfera se encuentra entre los espectros de las esferas simples.

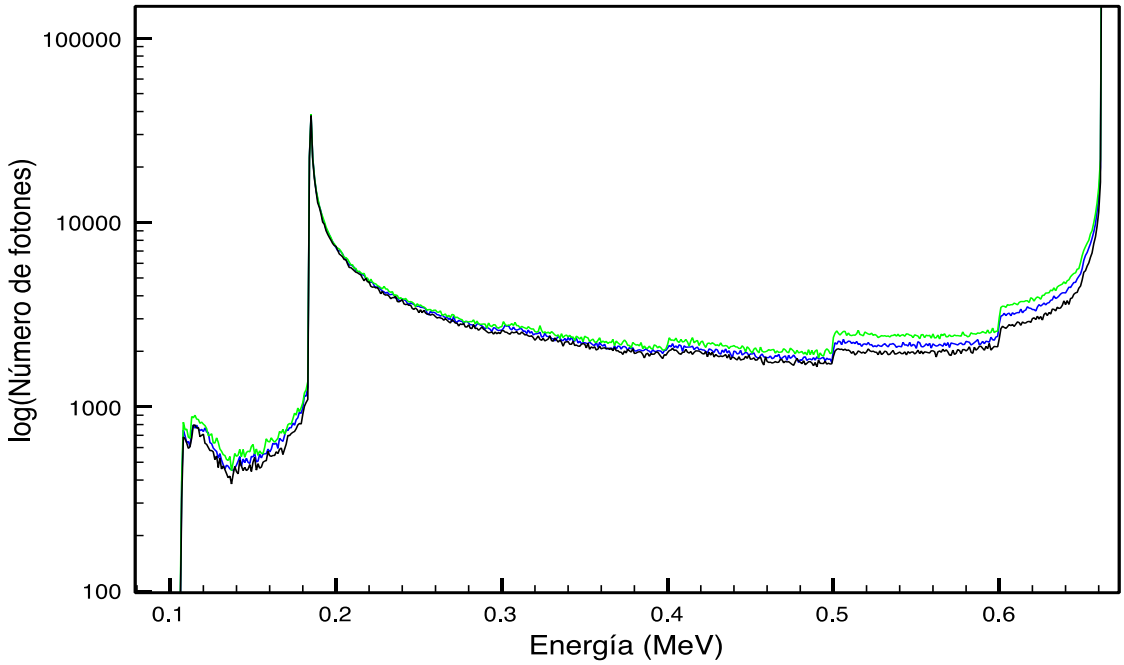

Figura 25. Espectros energéticos en escala logarítmica de una esfera simple de plomo de 6 cm de radio (verde), de una esfera doble de plomo y óxido de uranio de radios 1+5 cm (azul) y una esfera simple de UO_2 de 6 cm (negro).

Por último, en la Figura 26 se muestran los espectros en escala logarítmica de una doble esfera de plomo y óxido de uranio de radios 1+5, 3+3 y 5+1 cm. Se ha ampliado la imagen para poder visualizar cómo el espectro de la esfera con más plomo queda por encima del de la esfera con mismo radio para plomo y óxido de uranio, que a su vez está por encima de la que tiene más óxido de uranio. Estos resultados son coherentes puesto que se ha visto que el óxido de uranio blinda mejor que el plomo.

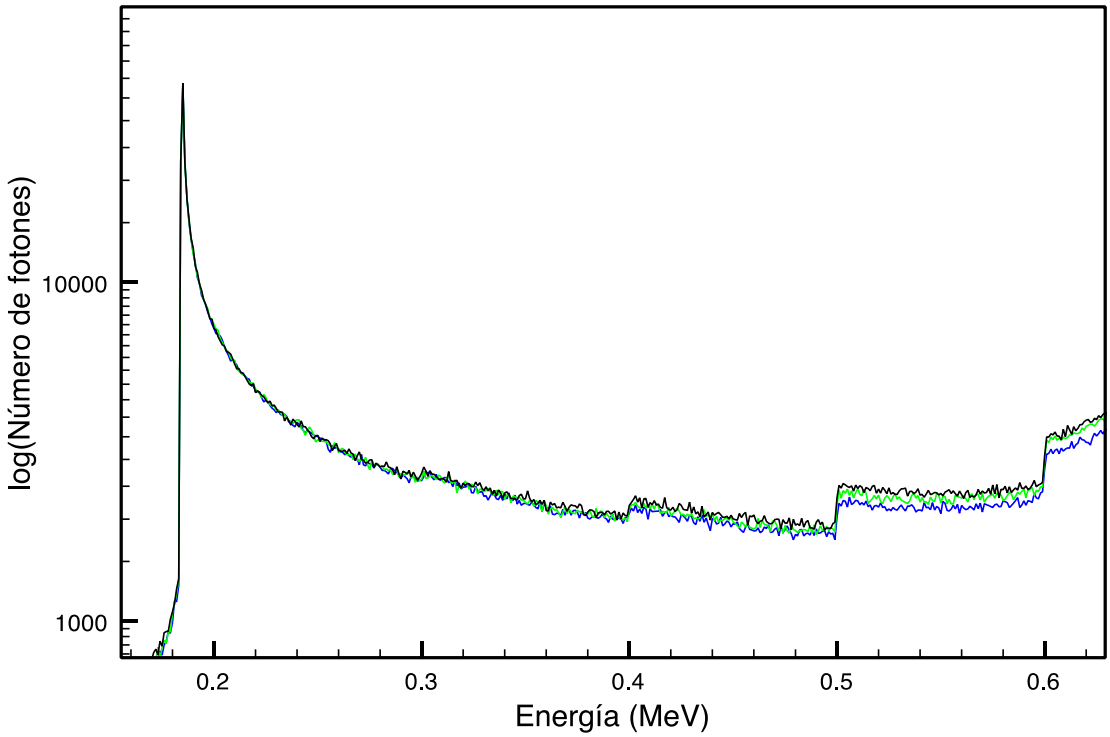

Figura 26. Espectros energéticos en escala logarítmica de tres esferas dobles de plomo y óxido de uranio de radios 1+5 cm (azul), 3+3 cm (verde) y 5+1 cm (negro).

7. Conclusiones y líneas futuras

En este documento se ha explicado un algoritmo de simulación del blindaje radiactivo que ofrece una esfera formada por uno o dos materiales de distintos radios, teniendo en el centro de la misma una fuente de ^{137}Cs que emite fotones gamma de 662 keV. Los materiales usados fueron plomo y óxido de uranio. Este algoritmo se programó en C++.

Para ello, en primera instancia se comentaron los principios físicos básicos en los que se basa el algoritmo de simulación, se habló del método de Monte Carlo, fundamental para el desarrollo de este tipo de algoritmos y, por último, de la librería MPI, usada para paralelizar el programa, optimizando los recursos y disminuyendo, por tanto, los tiempos de ejecución del programa.

Después se profundizó en el algoritmo y en el programa en cuestión: se comentó el diagrama de flujo del mismo así como el código del programa, se explicó cómo se desarrolló el mismo y se finalizó hablando de la comprobación del correcto funcionamiento del programa.

Se realizaron varias simulaciones para obtener datos representativos y poder comparar los blindajes de esferas de distintos radios y/o materiales. Se vio que en general para esferas de mayor tamaño mayor es el blindaje radiactivo y que la forma del espectro energético es la misma y lo que cambia esencialmente es el número de fotones de los picos (observado especialmente en la energía inicial de 662 keV). Se comprobó el grado de paralelización del programa obteniendo que según la ley de Amdahl el programa está paralelizado en un 98.7 %. También se comprobó que el óxido de uranio blinda mejor que el plomo para esferas de igual tamaño. En el caso de la doble esfera de plomo recubierta de óxido de uranio, se observó que los blindajes obtenidos eran iguales a los de una esfera simple de plomo del mismo radio debido a la definición del blindaje, pero que los espectros variaban ligeramente respecto del caso de la esfera simple y también con variaciones de los radios dentro de la doble esfera.

A pesar de que el programa es un programa completo, ni mucho menos es un programa acabado. En un trabajo futuro se podría considerar ampliar el programa para el uso de más materiales, para lo cual simplemente sería cuestión de introducir unas pocas líneas de código introduciendo estos materiales y su densidad en un mapa de datos y descargando los ficheros de las secciones eficaces de estos materiales de [6]. Aunque con el programa actual se puede cambiar la energía inicial de los fotones, otra mejora sería ampliar el rango de la energía inicial puesto que ahora el rango es de 0 a 1 MeV. Una última actualización podría ser cambiar la forma del material, simulando en vez de una esfera el cuerpo de un paciente, por ejemplo. Sin embargo, este cambio conllevaría una modificación importante del programa aunque la base seguiría siendo la misma.

Apéndice A. Códigos de programas.

Código del programa de simulación de un detector de radiación de NaI

```
//
//  main.cpp
//  NaI
//
//  Created by Borja Hawk on 10/04/12.
//  Copyright (c) 2012 BHAWK. All rights reserved.
//

#include <iostream>
#include <fstream>
#include <math.h>
#include <vector>
#include <map>
usingnamespacestd;
vector<vector<float>> vcSecciones;
float round(float r,int n_digit);
int effectfunction(float energia, float *seccionEficaz, bool rayleigh);
float rfunction(float seccionEficaz, float density);
float thetafunction();
float phifunction();
float Efunction(float theta, float E);
float dfunction(float x, float y, float z);
bool rayleigh = false;  //Variable que tiene en cuenta o no la dispersión Rayleigh


int main(int argc, constchar * argv[])
{
//Se carga el fichero de las secciones eficaces del NaI
ifstream fic("/Users/BorjaHawk/Downloads/cross-sections-nai.dat");
if (fic == NULL)
perror ("Error al abrir el archivo");
char buffer[100];
int i = 0;

while(!fic.eof())
{
        fic.getline(buffer,100);

if(i>2) //No se tienen en cuenta las 2 primeras líneas
{
char * pch;
pch = strtok (buffer," ");
int j = 0;
vector<float> seccion;
while (pch != NULL)
{
seccion.push_back(atof(pch));
pch = strtok (NULL, " ");
j++;
            }
vcSecciones.push_back(seccion);
}
        i++;
    }
    fic.close();
vcSecciones.pop_back(); //Por razones del strtok se añade un último elemento vacío y hay que quitarlo


//Se realizan las iteraciones
int iteraciones = 1000000;
int radio = 8;
typedefmap<float, int> MAP_ESPECTRO;   //Se crea el mapa que almacenará el espectro energético
MAP_ESPECTRO mEspectro;
float density = 3.67;
int numPE = 0;
srand(time(NULL));          //Se inicializa el número aleatorio

for (int k=0; k<iteraciones; k++)
    {
float E = 0.662;      //energía del fotón de Cs137
float x, y, z;
        x=y=z=0;
int effect = -1;
float distance = 0;
```

```
//Este bucle se realiza siempre que no haya efecto fotoeléctrico y la distancia recorrida sea menor que el radio
do {
float seccionEficaz = 0;
        effect = effectfunction(E, &seccionEficaz, rayleigh);              //Se obtiene el efecto a partir de la energía del fotón
//y la sección eficaz. Se indica si se quiere Rayleigh
float r = rfunction(seccionEficaz, density);       //Se halla r en función de la sección eficaz y de la densidad
float theta = thetafunction();                  //Se halla aleatoriamente el ángulo theta
float phi = phifunction();                      //Se halla phi aleatoriamente
float xold = x; float yold = y; float zold = z;       //Se actualizan las coordenadas anteriores
                                float dx = r*sin(theta)*cos(phi);              //Se calcula el nuevo vector recorrido por el fotón
                                float dy = r*sin(theta)*sin(phi);
                                float dz = r*cos(theta);
                                x = xold + dx;                 //Se calculan las nuevas coordenadas respecto al centro de la circunferencia
                                y = yold + dy;
                                z = zold + dz;
                                distance = dfunction(x,y,z);           //Halla la distancia del fotón respecto al centro de la circunferencia
if(distance > radio && effect == 0) //Si el fotoeléctrico sale fuera de la esfera, en realidad el fotón ha salido
        {
           effect = 5;                //Se trata como un fotón que no ha salido y no ha perdido energía
break;
        }
elseif(effect == 1)              //Si el efecto es Compton cambia la energía
        {
float Enew = Efunction(theta, E);   //Se halla la energía del fotón resultante a partir de theta y E
E = Enew;
        }
      } while (distance <= radio && effect != 0);

if(effect==0)   //Si se ha producido una absorción se incrementa el número de absorciones
        numPE++;
elseif(effect == 1)                     //efecto Compton -> Hay que obtener el espectro
      {
//Se calcula la energía depositada redondeando a 3 decimales
float Edepositada = round(0.662 - E,3);
//Se suma un fotón a la energía depositada dentro del mapa del espectro
MAP_ESPECTRO::iterator it = mEspectro.find(Edepositada);
if(it != mEspectro.end())
            (*it).second++;
else
mEspectro.insert(MAP_ESPECTRO::value_type(Edepositada,1));
}

   }
//Se copia en el mapa la energía de fotoabsorción
   mEspectro.insert(MAP_ESPECTRO::value_type(0.661,0));    //Se introduce este punto para que salga el fotopico como una delta
mEspectro.insert(MAP_ESPECTRO::value_type(0.662,numPE));
//Finalmente se crea el archivo con el espectro
ofstream f2;
f2.open("/Users/BorjaHawk/Documents/ficheroEspectro.txt", ofstream::out);
MAP_ESPECTRO::const_iterator it;
for (it=mEspectro.begin() ; it != mEspectro.end(); it++)
      f2 << (*it).first<<" "<< (*it).second<<endl;
for(float i = 0.663; i<= 0.7; i+=0.001)
      f2 << i <<" "<<0<<endl;
f2.close();

return0;
}

//Función que hace el redondeo de un número con el número de decimales requerido
float round(float r,int n_digit)
{
int n=pow(10,n_digit);
r=((float)((int)(r*n+0.5)))/n;
return(r);
}

//Función que obtiene el efecto probabilísticamente y además devuelve su sección eficaz a la energía dada
int effectfunction(float energia, float *seccionEficaz, bool rayleigh)
{
float nr = ((float)rand())/(float)RAND_MAX;
//se buscan las secciones eficaces correspondientes a la energía del fotón
for (size_t i=0; i<vcSecciones.size(); i++)
{
if(energia <vcSecciones[i][0] && i == 0)  //la energía es más baja que la primera, consideramos absorción
      {
         *seccionEficaz = vcSecciones[i][3];
return0;
      }
elseif(energia == vcSecciones[i][0])      //La energía coincide con una de las guardadas
      {
float sumaSecciones = vcSecciones[i][2] + vcSecciones[i][3];
if(rayleigh)
            sumaSecciones = sumaSecciones + vcSecciones[i][1];
float probPE = vcSecciones[i][3] / sumaSecciones;
float probCO = vcSecciones[i][2] / sumaSecciones;
if (nr < probPE)                                //fotoeléctrico
```

```
            {
                *seccionEficaz = vcSecciones[i][3];
return0;
            }
elseif(nr>=probPE && nr <= probPE + probCO)          //Compton
            {
                *seccionEficaz = vcSecciones[i][2];
return1;
            }
else//Rayleigh
            {
                *seccionEficaz = vcSecciones[i][1];
return2;
            }
        }
elseif(energia >vcSecciones[i][0])
{
if(i == vcSecciones.size()-1)   //la energía es más alta que todas, cogemos la última
            {
float sumaSecciones = vcSecciones[i][2] + vcSecciones[i][3];
if(rayleigh)
                    sumaSecciones = sumaSecciones + vcSecciones[i][1];
float probPE = vcSecciones[i][3] / sumaSecciones;
float probCO = vcSecciones[i][2] / sumaSecciones;
if (nr < probPE)                                //fotoeléctrico
                {
                    *seccionEficaz = vcSecciones[i][3];
return0;
                }
elseif(nr>=probPE && nr <= probPE + probCO)       //Compton
                {
                    *seccionEficaz = vcSecciones[i][2];
return1;
                }
else//Rayleigh
{
                    *seccionEficaz = vcSecciones[i][1];
return2;
                }
            }
        }
elseif(energia <vcSecciones[i][0])    //La energía es menor que la iteración de la energía
        {
//Si la energía está entre dos de la lista, se hace el promedio de secciones eficaces
float promedioPE = vcSecciones[i-1][3];
float promedioCO = vcSecciones[i-1][2];
float promedioRA = vcSecciones[i-1][1];

//Hay que comprobar que no haya más energías iguales a la superior
size_t k = i;
float seccionAltaPE = vcSecciones[k][3];
float seccionAltaCO = vcSecciones[k][2];
float seccionAltaRA = vcSecciones[k][1];
while(k <vcSecciones.size()-1&&vcSecciones[k][0] == vcSecciones[k+1][0])
            {
//En caso de haber distintas secciones eficaces con la misma energía se toma la sección más grande
if (vcSecciones[k+1][3]>seccionAltaPE)
                    seccionAltaPE = vcSecciones[k+1][3];
if (vcSecciones[k+1][2]>seccionAltaCO)
                    seccionAltaCO = vcSecciones[k+1][2];
if (vcSecciones[k+1][1]>seccionAltaRA)
                    seccionAltaRA = vcSecciones[k+1][1];
                k++;
            }
//Se promedian las secciones
            promedioPE+=seccionAltaPE;
            promedioCO+=seccionAltaCO;
            promedioRA+=seccionAltaRA;
            promedioPE = promedioPE/2;
            promedioCO = promedioCO/2;
            promedioRA = promedioRA/2;

float sumaSecciones = promedioPE + promedioCO;
if(rayleigh)
                sumaSecciones = sumaSecciones + promedioRA;
float probPE = promedioPE/sumaSecciones;
float probCO = promedioCO/sumaSecciones;
if (nr < probPE)                                    //fotoeléctrico
            {
                *seccionEficaz = promedioPE;
return0;
            }
elseif(nr>=probPE && nr <= probPE + probCO)          //Compton
{
                *seccionEficaz = promedioCO;
return1;
            }
```

```
else//Rayleigh
        {
           *seccionEficaz = promedioRA;
return2;
        }
      }
   }
return -1;
}

//Función que calcula el radio del nuevo vector del fotón
float rfunction(float seccionEficaz, float density)
{
        float mfpath = 1.0 / (seccionEficaz * density);          //fórmula del camino medio libre
        float nr = ((float)rand())/(float)RAND_MAX;                    //nº aleatorio entre 0 y 1
        float r = -1 * log(nr) * mfpath;                  //trasladamos el nº aleatorio usando logaritmo
        return r;
}

//Función que calcula theta
float thetafunction()
{
        float nr = ((float)rand())/(float)RAND_MAX;                    //nº aleatorio entre 0 y 1
        float theta = nr * M_PI;                          //se traslada al intervalo (0, pi)
return theta;
}

//Función que calcula phi
float phifunction()
{
        float nr = ((float)rand())/(float)RAND_MAX;              //nº aleatorio entre 0 y 1
        float phi = nr * 2 * M_PI; //se traslada al intervalo (0, 2*pi)
return phi;
}

//Función que calcula la energía nueva después de un efecto Compton
float Efunction(float theta, float E)
{
        float Enew = E / (1 + ((1 - cos(theta)) * E/0.511));                    //fórmula de la energía
        return Enew;
}

//Función que calcula la distancia del fotón al centro de la esfera
float dfunction(float x, float y, float z)
{
        float d = sqrt(pow(x,2) + pow(y,2) + pow(z,2));
        return d;
}
```

Código del programa de simulación de blindaje que ofrece una esfera

```
//
//  main.cpp
//  Blindaje
//
//  Created by Borja Hawk on 28/04/12.
//  Copyright (c) 2012 BHAWK. All rights reserved.
//

#include <iostream>
#include <fstream>
#include <math.h>
#include <vector>
#include <map>
#include <sstream>
#include <mpi.h>
usingnamespace std;
vector<vector<float>> vcSecciones1;
vector<vector<float>> vcSecciones2;
float round(float r,int n_digit);
int effectfunction(float energia, float *seccionEficaz, bool rayleigh, int tipoEsfera);
float rfunction(float seccionEficaz, float density);
float thetafunction();
float phifunction();
float Efunction(float theta, float E);
float dfunction(float x, float y, float z);
bool cargarFicheroSeccionesEficazes(string composicion, int tipoEsfera);
bool rayleigh = false;
int parserParametros(int argc, char *argv[], int *iteraciones, float *energia, float *radio1, float *radio2,
string * material1, string * material2);
unsigned time_seed(int rank);


int main(int argc, char * argv[])
{
int rank, size;

//Se inicializa el procesamiento en paralelo
   MPI_Init(&argc, &argv);
double timeInit = MPI_Wtime();      //Se toma el tiempo al inicio del programa
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
   MPI_Comm_size(MPI_COMM_WORLD,&size);

int parser = 1;
int iteraciones = 1000000;          //El número de iteraciones es de 1 millón por defecto
float energia = 1;                  //La energía inicial es de 1 MeV por defecto
string composicion1 = "";
float radio1;
float density1;
string composicion2 = "";
float radio2;
float density2 = 0;

//Se obtienen los parámetros introducidos por el usuario, sólo lo hace un nodo
if(rank == 0)
        parser = parserParametros(argc, argv, &iteraciones, &energia, &radio1, &radio2, &composicion1, &composicion2);

MPI_Bcast(&parser, 1, MPI_INT, 0, MPI_COMM_WORLD);
if(!parser)                         //En caso de no ser correctos los parámetros se sale del programa
   {
      MPI_Finalize();
return -1;
   }

//Se envían las composiciones a todos los nodos
char m1[3], m2[3];
strcpy(m1, composicion1.c_str());
   MPI_Bcast(&m1, 3, MPI_CHAR, 0, MPI_COMM_WORLD);
strcpy(m2, composicion2.c_str());
   MPI_Bcast(&m2, 3, MPI_CHAR, 0, MPI_COMM_WORLD);

//Se inicializa un mapa con los posibles materiales de la esfera y sus densidades
typedef map<string, float> MAP_MATERIALES;
   MAP_MATERIALES mMateriales;
mMateriales.insert(MAP_MATERIALES::value_type("Pb",11.34));
mMateriales.insert(MAP_MATERIALES::value_type("U",19.05));
mMateriales.insert(MAP_MATERIALES::value_type("UO2",10.97));
//Se obtiene la densidad del material 1 a usar
MAP_MATERIALES::iterator it1 = mMateriales.find(m1);
if(it1 != mMateriales.end())
       density1 = (*it1).second;
//Se carga el fichero de las secciones eficaces de la composición de la esfera concéntrica
   cargarFicheroSeccionesEficazes(m1, 1);
//Se obtiene la densidad del material 2 a usar si se ha introducido alguno
```

```
it1 = mMateriales.find(m2);
if(it1 != mMateriales.end())
   {
      density2 = (*it1).second;
cargarFicheroSeccionesEficazes(m2, 2);
}

//Se envían los parámetros a todos los nodos
MPI_Bcast(&iteraciones, 1, MPI_INT, 0, MPI_COMM_WORLD);
   MPI_Bcast(&energia, 1, MPI_FLOAT, 0, MPI_COMM_WORLD);
   MPI_Bcast(&radio1, 1, MPI_FLOAT, 0, MPI_COMM_WORLD);
   MPI_Bcast(&radio2, 1, MPI_FLOAT, 0, MPI_COMM_WORLD);

//Se reparten el número de fotones a procesar entre los nodos
int cuentas = iteraciones / size;
int inicial = 1 + rank*cuentas;
int final = (rank+1)*cuentas;
if(rank == size - 1)
final = iteraciones;


   srand(time_seed(rank+1));          //Se inicializa el número aleatorio utilizando el nº de procesador para que los cálculos sean distintos
int numFotonesPeligrosos = 0;
float energiasFotones[iteraciones];             //Vector donde se guardará la energía final de cada fotón
   std::fill_n(energiasFotones, iteraciones, 0);   //Se inicializa con ceros

//Se realizan las iteraciones
for (int k=inicial; k<=final; k++)
   {
float E = energia;      //energía introducida por el usuario
//Se inicializa el punto del fotón, el efecto sufrido y la distancia recorrida
float x, y, z;
      x=y=z=0;
int effect = -1;
float distance = 0;
//Este bucle se realiza siempre que no haya efecto fotoeléctrico, la distancia recorrida sea menor que el radio y la energía mayor de 1
keV
do {
float seccionEficaz = 0;
float r = 0;
//Si la distancia es menor del radio 1 se toman los datos de la esfera interna
if(distance <= radio1)
         {
            effect = effectfunction(E, &seccionEficaz, rayleigh, 1);        //Se obtiene el efecto a partir de la energía
            r = rfunction(seccionEficaz, density1);     //Se halla r en función de la sección eficaz y de la densidad
         }
else////Si la distancia es mayor del radio 1 se toman los datos de la esfera externa
         {
            effect = effectfunction(E, &seccionEficaz, rayleigh, 2);        //Se obtiene el efecto a partir de la energía
            r = rfunction(seccionEficaz, density2);     //Se halla r en función de la sección eficaz y de la densidad
         }

float theta = thetafunction();                  //Se halla aleatoriamente el ángulo theta
float phi = phifunction();                     //Se halla phi aleatoriamente
float xold = x; float yold = y; float zold = z;       //Se actualizan las coordenadas anteriores
                              float dx = r*sin(theta)*cos(phi);              //Se calcula el nuevo vector recorrido por el fotón
                              float dy = r*sin(theta)*sin(phi);
                              float dz = r*cos(theta);
                              x = xold + dx;                 //Se calculan las nuevas coordenadas respecto al centro de la circunferencia
                              y = yold + dy;
                              z = zold + dz;
                              distance = dfunction(x,y,z);           //Halla la distancia del fotón respecto al centro de la circunferencia
if(distance > radio2 && effect == 0)    //Si el fotoeléctrico sale fuera de la esfera, en realidad el fotón ha salido
         {
            effect = 1;                //Se trata como un Compton que no ha perdido energía
break;
         }
elseif(effect == 1)                    //Si el efecto es Compton cambia la energía
         {
float Enew = Efunction(theta, E);   //Se halla la energía del fotón resultante a partir de theta y E
E = Enew;
         }
      } while (distance <= radio2 && effect != 0&& E >0.001);

//Si la energía del fotón es la inicial y ha salido de la esfera se incrementa en 1 el número de fotones peligrosos
if(E == energia && effect != 0)
         numFotonesPeligrosos++;
//Se calcula la energía del fotón redondeando a 3 decimales si no ha sido efecto fotoeléctrico
if(effect!=0)
      {
float Efinal = round(E,3);
energiasFotones[k-1] = Efinal;
      }
else
         energiasFotones[k-1] = 0;
   }
```

```
//Se suman el número de fotones peligrosos y los vectores de energías de cada nodo, juntando todos los resultados en el nodo 0
int numFotonesPeligrososTotal = 0;
float energiasFotonesTotal[iteraciones];
   MPI_Reduce(&numFotonesPeligrosos, &numFotonesPeligrososTotal, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
   MPI_Reduce(&energiasFotones, &energiasFotonesTotal, iteraciones, MPI_FLOAT, MPI_SUM, 0, MPI_COMM_WORLD);

//Finalmente se crea el archivo con el espectro en el nodo 0
if (rank==0)
   {
//Primero se halla el mapa con el espectro resultante
typedef map<float, int> MAP_ESPECTRO;
      MAP_ESPECTRO mEspectro;
for(int i=0; i<iteraciones; i++)
{
if(energiasFotonesTotal[i] != 0)
         {
//Se suma un fotón a la energía saliente dentro del mapa del espectro
MAP_ESPECTRO::iterator it = mEspectro.find(energiasFotonesTotal[i]);
if(it != mEspectro.end())
               (*it).second++;
else
mEspectro.insert(MAP_ESPECTRO::value_type(energiasFotonesTotal[i],1));
}
      }
//Después se crea el fichero en el que escribimos el mapa guardado
      ofstream f2;
      stringstream ss;
      ss <<"/home/borja/simulaciones/sim_"<< composicion1 <<"_radio"<< radio1 << composicion2 <<"_radio"<< radio2 <<"_energia"<<
energia <<"_fotones"<< iteraciones <<".txt";
//ss << "/Users/BorjaHawk/Desktop/Trabajo Master/Algoritmo de blindaje/Simulaciones/sim_" << composicion1 <<"_radio" << radio1 <<
composicion2 <<"_radio" << radio2 << "_energia" << energia << "_fotones" << iteraciones << ".txt";
f2.open(ss.str().c_str(), ofstream::out);
      MAP_ESPECTRO::const_iterator it;
for (it=mEspectro.begin() ; it != mEspectro.end(); it++)
         f2 << (*it).first <<" "<< (*it).second << endl;
f2.close();

//Se escriben los resultados de blindaje y tiempo de ejecución por pantalla
double timeEnd = MPI_Wtime();          //Se guarda el tiempo de finalización
double timeTotal = timeEnd-timeInit;   //Se halla el tiempo total de ejecución del nodo
      cout <<"Tiempo de ejecución de nodo "<< rank <<": "<< timeTotal <<" segundos."<< endl;
float blindaje = round((float)(iteraciones - numFotonesPeligrososTotal)/iteraciones*100, 2);
      cout <<"Blindaje (porcentaje de fotones retenidos) = "<< blindaje << endl;
      cout <<"Archivo de espectro: sim_"<< composicion1 <<"_radio"<< radio1 << composicion2 <<"_radio"<< radio2 <<"_energia"<<
energia <<"_fotones"<< iteraciones <<".txt"<< endl;
   }
else//En el resto de nodos solo se escribe el tiempo de ejecución
{
double timeEnd = MPI_Wtime();
double timeTotal = timeEnd-timeInit;
cout <<"Tiempo de ejecución de nodo "<< rank <<": "<< timeTotal <<" segundos."<< endl;
   }

   MPI_Finalize();    //Se termina la parelelización
return1;
}

//Función que hace el redondeo de un número con el número de decimales requerido
float round(float r,int n_digit)
{
int n=pow(10,n_digit);
r=((float)((int)(r*n+0.5)))/n;
return(r);
}

//Función que obtiene el efecto probabilísticamente y además devuelve su sección eficaz a la energía dada
int effectfunction(float energia, float *seccionEficaz, bool rayleigh, int tipoEsfera)
{
//Dependiendo de la esfera donde esté el fotón se usan las secciones eficaces de una u otra
   vector<vector<float>> vcSecciones;
if(tipoEsfera == 1)
      vcSecciones = vcSecciones1;
elseif (tipoEsfera == 2)
vcSecciones = vcSecciones2;

float nr = ((float)rand())/(float)RAND_MAX;
//se buscan las secciones eficaces correspondientes a la energía del fotón
for (size_t i=0; i<vcSecciones.size(); i++)
{
if(energia < vcSecciones[i][0] && i == 0)  //la energía es más baja que la primera, consideramos absorción
      {
         *seccionEficaz = vcSecciones[i][3];
return0;
      }
elseif(energia == vcSecciones[i][0])      //La energía coincide con una de las guardadas
      {
float sumaSecciones = vcSecciones[i][2] + vcSecciones[i][3];
```

```
if(rayleigh)
            sumaSecciones = sumaSecciones + vcSecciones[i][1];
float probPE = vcSecciones[i][3] / sumaSecciones;
float probCO = vcSecciones[i][2] / sumaSecciones;
if (nr < probPE)                                    //fotoeléctrico
        {
            *seccionEficaz = vcSecciones[i][3];
return0;
        }
elseif(nr>=probPE && nr <= probPE + probCO)          //Compton
        {
            *seccionEficaz = vcSecciones[i][2];
return1;
        }
else//Rayleigh
        {
            *seccionEficaz = vcSecciones[i][1];
return2;
        }
    }
elseif(energia > vcSecciones[i][0])
{
if(i == vcSecciones.size()-1)   //la energía es más alta que todas, cogemos la última
        {
float sumaSecciones = vcSecciones[i][2] + vcSecciones[i][3];
if(rayleigh)
                sumaSecciones = sumaSecciones + vcSecciones[i][1];
float probPE = vcSecciones[i][3] / sumaSecciones;
float probCO = vcSecciones[i][2] / sumaSecciones;
if (nr < probPE)                                    //fotoeléctrico
            {
                *seccionEficaz = vcSecciones[i][3];
return0;
            }
elseif(nr>=probPE && nr <= probPE + probCO)        //Compton
            {
                *seccionEficaz = vcSecciones[i][2];
return1;
            }
else//Rayleigh
{
                *seccionEficaz = vcSecciones[i][1];
return2;
            }
        }
    }
elseif(energia < vcSecciones[i][0])        //La energía es menor que la iteración de la energía
    {
//Si la energía está entre dos de la lista, se hace el promedio de secciones eficaces
float promedioPE = vcSecciones[i-1][3];
float promedioCO = vcSecciones[i-1][2];
float promedioRA = vcSecciones[i-1][1];

//Hay que comprobar que no haya más energías iguales a la superior
size_t k = i;
float seccionAltaPE = vcSecciones[k][3];
float seccionAltaCO = vcSecciones[k][2];
float seccionAltaRA = vcSecciones[k][1];
while(k < vcSecciones.size()-1&& vcSecciones[k][0] == vcSecciones[k+1][0])
        {
//En caso de haber distintas secciones eficaces con la misma energía se toma la sección más grande
if (vcSecciones[k+1][3]>seccionAltaPE)
                seccionAltaPE = vcSecciones[k+1][3];
if (vcSecciones[k+1][2]>seccionAltaCO)
                seccionAltaCO = vcSecciones[k+1][2];
if (vcSecciones[k+1][1]>seccionAltaRA)
                seccionAltaRA = vcSecciones[k+1][1];
            k++;
        }
//Se promedian las secciones
        promedioPE+=seccionAltaPE;
        promedioCO+=seccionAltaCO;
        promedioRA+=seccionAltaRA;
        promedioPE = promedioPE/2;
        promedioCO = promedioCO/2;
        promedioRA = promedioRA/2;

float sumaSecciones = promedioPE + promedioCO;
if(rayleigh)
            sumaSecciones = sumaSecciones + promedioRA;
float probPE = promedioPE/sumaSecciones;
float probCO = promedioCO/sumaSecciones;
if (nr < probPE)                                    //fotoeléctrico
        {
            *seccionEficaz = promedioPE;
return0;
        }
```

```
elseif(nr>=probPE && nr <= probPE + probCO)           //Compton
{
            *seccionEficaz = promedioCO;
return1;
        }
else//Rayleigh
        {
            *seccionEficaz = promedioRA;
return2;
        }
      }
   }
return -1;
}

//Función que calcula el radio del nuevo vector del fotón
float rfunction(float seccionEficaz, float density)
{
            float mfpath = 1.0 / (seccionEficaz * density);          //fórmula del camino medio libre
            float nr = ((float)rand())/(float)RAND_MAX;                        //nº aleatorio entre 0 y 1
            float r = log(nr) * -1 * mfpath;                        //trasladamos el nº aleatorio al intervalo (0, 2*mfpath)
            return r;
}

//Función que calcula theta
float thetafunction()
{
            float nr = ((float)rand())/(float)RAND_MAX;                        //nº aleatorio entre 0 y 1
            float theta = nr * M_PI;   //se traslada al intervalo (0, pi)
return theta;
}

//Función que calcula phi
float phifunction()
{
            float nr = ((float)rand())/(float)RAND_MAX;              //nº aleatorio entre 0 y 1
            float phi = nr * 2 * M_PI; //se traslada al intervalo (0, 2*pi)
return phi;
}

//Función que calcula la energía nueva después de un efecto Compton
float Efunction(float theta, float E)
{
            float Enew = E / (1 + ((1 - cos(theta)) * E/0.511));                        //fórmula de la energía
            return Enew;
}

//Función que calcula la distancia del fotón al centro de la esfera
float dfunction(float x, float y, float z)
{
            float d = sqrt(pow(x,2) + pow(y,2) + pow(z,2));
            return d;
}

//Función usada en parserParametros para obtener los parámetros
char* getCmdOption(char ** begin, char ** end, const std::string & option)
{
char ** itr = std::find(begin, end, option);
if (itr != end && ++itr != end)
{
return *itr;
   }
return0;
}

//Función usada en parserParametros para comprobar la introducción de los parámetros
bool cmdOptionExists(char** begin, char** end, const std::string& option)
{
return std::find(begin, end, option) != end;
}

//Función usada en parserParametros para comprobar si el parámetro introducido es un número
template<class T>
bool from_string(T& t, const string& s, ios_base& (*f)(ios_base&))
{
istringstream iss(s);
return !(iss >> f >> t).fail();
}

//Función que recoge los parámetros introducidos y los guarda en variables
int parserParametros(int argc, char *argv[], int *iteraciones, float *energia, float *radio1, float *radio2,
string * material1, string * material2)
{
//Se comprueba que al menos se ha introducido el material y el radio de la esfera 1
if(cmdOptionExists(argv, argv+argc, "-m1") && cmdOptionExists(argv, argv+argc, "-r1"))
{
char * charIteraciones = getCmdOption(argv, argv + argc, "-p");     //número de iteraciones
```

```
if (charIteraciones)
	{
if(!from_string<int>(*iteraciones, charIteraciones, std::dec))
{
		cout <<"El formato de n no es válido."<< std::endl;
return0;
	}
else
if(iteraciones == 0)
{
		cout <<"El número de fotones no puede ser 0."<< std::endl;
return0;
		}
	}
char * charEnergia = getCmdOption(argv, argv + argc, "-e");		//energía inicial
if (charEnergia)
	{
if(!from_string<float>(*energia, charEnergia, std::dec))
{
		cout <<"El formato de e no es válido."<< std::endl;
return0;
	}
else
	{
if(energia == 0)
{
		cout <<"Error - La energía no puede ser 0."<< std::endl;
return0;
		}
elseif(*energia >1)
		{
		cout <<"Error - La energía no puede ser mayor de 1 MeV."<< std::endl;
return0;
		}
	}
	}
char * charMaterial = getCmdOption(argv, argv + argc, "-m1");   //material esfera interna
if (charMaterial)
	{
if(strcmp(charMaterial, "Pb") == 0 || strcmp(charMaterial, "U") == 0 || strcmp(charMaterial, "UO2") == 0)
*material1 = charMaterial;
else
	{
		cout <<"Error - El material de la esfera concéntrica debe ser Pb, U o UO2."<<endl;
return0;
	}

	}
char * charRadio = getCmdOption(argv, argv + argc, "-r1");	//radio esfera interna
if (charRadio)
	{
if(!from_string<float>(*radio1, charRadio, std::dec))
{
		cout <<"El formato de r1 no es válido."<< std::endl;
return0;
	}
else
if(radio1 == 0)
{
		cout <<"Error - r1 no puede ser 0."<< std::endl;
return0;
		}
	}
charMaterial = getCmdOption(argv, argv + argc, "-m2");		//material esfera externa
if (charMaterial)
	{
if(strcmp(charMaterial, "Pb") == 0 || strcmp(charMaterial, "U") == 0 || strcmp(charMaterial, "UO2") == 0)
	{
		*material2 = charMaterial;
charRadio = getCmdOption(argv, argv + argc, "-r2");	//radio esfera externa
if (charRadio)
		{
if(!from_string<float>(*radio2, charRadio, std::dec))
{
			cout <<"El formato de r2 no es válido."<< std::endl;
return0;
		}
else
if(*radio2 <= *radio1)
			{
cout<<"Error - r2 debe ser mayor que r1."<< std::endl;
return0;
			}
		}
else
		{
		cout <<"Error - Se ha definido m2 pero no r2."<< std::endl;
```

```
return0;
            }
        }
else
        {
            cout <<"Error - El material de la esfera excéntrica debe ser Pb, U o UO2."<<endl;
return0;
        }
    }
else
        *radio2 = *radio1;
char * charRayleigh = getCmdOption(argv, argv + argc, "-r");    //Con o sin dispersión Rayleigh
if (charRayleigh)
    {
if(strcmp(charRayleigh, "s") == 0)
rayleigh = true;
elseif(strcmp(charRayleigh, "n") != 0)
{
            cout <<"Error - El efecto Rayleigh debe ser s (si) o n (no)."<< endl;
return0;
        }

    }
  }
else//Si no se han introducido bien los parámetros salta el texto de ayuda
  {
    cout <<"Los parámetros que acepta el programa son los siguientes:"<<endl;
    cout <<"-p número de fotones a simular (1 millón si se omite),"<< endl;
    cout <<"-e energía inicial de los fotones, hasta 1 MeV (1 MeV si se omite)"<< endl;
    cout <<"-m1 composición de la esfera concéntrica de blindaje (obligatorio), opciones"<< endl;
    cout <<"    Pb: plomo,"<< endl <<"    U: uranio empobrecido,"<<endl <<"    UO2: dióxido de uranio;"<< endl;
    cout <<"-m2 composición de la esfera excéntrica de blindaje (no hay si se omite),"<< endl;
    cout <<"-r1 radio de la esfera concéntrica de blindaje (obligatorio)."<<endl;
    cout <<"-r2 radio de la esfera excéntrica de blindaje (obligatorio si se usa dicha esfera)."<<endl;
    cout <<" -r s/n para tener en cuenta o no respectivamente la dispersión de Rayleigh (no si se omite)."<< endl;
return0;
  }
return1;
}

//Función que carga las secciones eficaces en vectores a partir de los ficheros de texto
bool cargarFicheroSeccionesEficazes(string composicion, int tipoEsfera)
{
  vector<vector<float>> vcSecciones;
  string cadenaFichero = "/home/borja/seccioneseficaces/cross-sections-" + composicion + ".dat";
//string cadenaFichero = "/Users/BorjaHawk/Desktop/Trabajo Master/Algoritmo de blindaje/secciones_eficaces/cross-sections-" +
composicion + ".dat";

ifstream fic(cadenaFichero.c_str());
if (fic == NULL)
  {
    perror ("Error al abrir el archivo");
fic.close();
returnfalse;
  }
char buffer[100];
int i = 0;

while(!fic.eof())
  {
fic.getline(buffer,100);

    {
char * pch;
pch = strtok (buffer," ");
int j = 0;
vector<float> seccion;
while (pch != NULL)
{
seccion.push_back(atof(pch));
pch = strtok (NULL, " ");
j++;
        }
vcSecciones.push_back(seccion);
}
    i++;
  }
  fic.close();
  vcSecciones.pop_back(); //Por razones del strtok se añade un último elemento vacío y hay que quitarlo
if(tipoEsfera == 1)
    vcSecciones1 = vcSecciones;
elseif (tipoEsfera == 2)
    vcSecciones2 = vcSecciones;

returntrue;
}
```

```
//Función que inicia la semilla de los números pseudoaleatorios a partir del número de nodo
unsigned time_seed(int rank)
{
time_t now =time(0);
unsignedchar *p = (unsignedchar *)&now;

unsigned seed=rank;
size_t i;

for (i=0;i<sizeof now;i++)
seed = seed * (UCHAR_MAX+2U) + p[i];
return seed;
}
```

Apéndice B. Ficheros de secciones eficaces.

Fichero de secciones eficaces del ioduro de sodio (NaI)

Photon Energy	Coherent Scatter.	Incoher. Scatter.	Photoel. Absorb.
1.000E-03	7.422E+00	5.916E-03	7.794E+03
1.035E-03	7.394E+00	6.244E-03	7.245E+03
1.072E-03	7.366E+00	6.586E-03	6.736E+03
1.072E-03	7.366E+00	6.586E-03	7.021E+03
1.072E-03	7.366E+00	6.586E-03	7.925E+03
1.072E-03	7.366E+00	6.587E-03	7.021E+03
1.072E-03	7.366E+00	6.587E-03	7.924E+03
1.500E-03	7.010E+00	1.065E-02	3.801E+03
2.000E-03	6.555E+00	1.541E-02	1.917E+03
3.000E-03	5.687E+00	2.459E-02	7.003E+02
4.000E-03	4.947E+00	3.292E-02	3.351E+02
4.557E-03	4.592E+00	3.712E-02	2.387E+02
4.557E-03	4.592E+00	3.712E-02	6.585E+02
4.702E-03	4.506E+00	3.816E-02	6.166E+02
4.852E-03	4.419E+00	3.923E-02	5.774E+02
4.852E-03	4.419E+00	3.923E-02	7.719E+02
5.000E-03	4.335E+00	4.026E-02	7.277E+02
5.188E-03	4.229E+00	4.155E-02	6.611E+02
5.188E-03	4.229E+00	4.155E-02	7.604E+02
6.000E-03	3.816E+00	4.679E-02	5.297E+02
8.000E-03	3.009E+00	5.799E-02	2.489E+02
1.000E-02	2.437E+00	6.731E-02	1.375E+02
1.500E-02	1.598E+00	8.410E-02	4.570E+01
2.000E-02	1.136E+00	9.463E-02	2.062E+01
3.000E-02	6.443E-01	1.067E-01	6.607E+00
3.317E-02	5.556E-01	1.090E-01	4.972E+00
3.317E-02	5.556E-01	1.090E-01	2.976E+01
4.000E-02	4.196E-01	1.127E-01	1.824E+01
5.000E-02	2.978E-01	1.157E-01	1.006E+01
6.000E-02	2.221E-01	1.169E-01	6.111E+00
8.000E-02	1.365E-01	1.165E-01	2.746E+00
1.000E-01	9.243E-02	1.144E-01	1.462E+00
1.500E-01	4.491E-02	1.071E-01	4.592E-01
2.000E-01	2.654E-02	1.000E-01	2.019E-01
3.000E-01	1.238E-02	8.860E-02	6.481E-02
4.000E-01	7.126E-03	8.009E-02	2.987E-02
5.000E-01	4.622E-03	7.351E-02	1.684E-02
6.000E-01	3.238E-03	6.822E-02	1.079E-02
6.500E-01	2.769E-03	6.592E-02	8.919E-03
6.620E-01	2.671E-03	6.540E-02	8.544E-03

Fichero de secciones eficaces del plomo

1.000E-03 1.251E+01 3.587E-03 5.197E+03
1.500E-03 1.201E+01 6.601E-03 2.344E+03
2.000E-03 1.144E+01 9.620E-03 1.274E+03
2.484E-03 1.088E+01 1.240E-02 7.900E+02
2.484E-03 1.088E+01 1.240E-02 1.385E+03
2.534E-03 1.082E+01 1.268E-02 1.636E+03
2.586E-03 1.076E+01 1.297E-02 1.933E+03
2.586E-03 1.076E+01 1.297E-02 2.439E+03
3.000E-03 1.027E+01 1.525E-02 1.955E+03
3.066E-03 1.019E+01 1.560E-02 1.847E+03
3.066E-03 1.019E+01 1.560E-02 2.136E+03
3.301E-03 9.929E+00 1.684E-02 1.782E+03
3.554E-03 9.652E+00 1.813E-02 1.486E+03
3.554E-03 9.652E+00 1.813E-02 1.575E+03
3.699E-03 9.496E+00 1.887E-02 1.432E+03
3.851E-03 9.335E+00 1.963E-02 1.302E+03
3.851E-03 9.335E+00 1.963E-02 1.358E+03
4.000E-03 9.179E+00 2.037E-02 1.242E+03
5.000E-03 8.208E+00 2.516E-02 7.222E+02
6.000E-03 7.362E+00 2.970E-02 4.598E+02
8.000E-03 6.002E+00 3.807E-02 2.226E+02
1.000E-02 4.982E+00 4.540E-02 1.256E+02
1.304E-02 3.851E+00 5.435E-02 6.310E+01
1.304E-02 3.851E+00 5.435E-02 1.582E+02
1.500E-02 3.308E+00 5.920E-02 1.082E+02
1.520E-02 3.258E+00 5.964E-02 1.045E+02
1.520E-02 3.258E+00 5.964E-02 1.452E+02
1.553E-02 3.180E+00 6.037E-02 1.380E+02
1.586E-02 3.104E+00 6.112E-02 1.312E+02
1.586E-02 3.104E+00 6.112E-02 1.517E+02
2.000E-02 2.338E+00 6.897E-02 8.397E+01
3.000E-02 1.377E+00 8.228E-02 2.886E+01
4.000E-02 9.202E-01 9.019E-02 1.335E+01
5.000E-02 6.545E-01 9.478E-02 7.292E+00
6.000E-02 4.900E-01 9.734E-02 4.432E+00
8.000E-02 3.078E-01 9.923E-02 2.012E+00
8.800E-02 2.632E-01 9.928E-02 1.547E+00
8.800E-02 2.632E-01 9.928E-02 7.321E+00
1.000E-01 2.128E-01 9.894E-02 5.237E+00
1.500E-01 1.049E-01 9.484E-02 1.815E+00
2.000E-01 6.260E-02 8.966E-02 8.464E-01
3.000E-01 2.988E-02 8.036E-02 2.930E-01
4.000E-01 1.746E-02 7.310E-02 1.417E-01
5.000E-01 1.143E-02 6.734E-02 8.257E-02
6.000E-01 8.060E-03 6.263E-02 5.406E-02
6.500E-01 6.907E-03 6.060E-02 4.513E-02
6.620E-01 6.667E-03 6.013E-02 4.333E-02
7.000E-01 5.986E-03 5.873E-02 3.830E-02
8.000E-01 4.621E-03 5.537E-02 2.871E-02
9.000E-01 3.675E-03 5.243E-02 2.244E-02
1.000E+00 2.991E-03 4.993E-02 1.810E-02

Fichero de secciones eficaces del óxido de uranio

1.000E-03 1.357E+01 4.526E-03 6.613E+03
1.022E-03 1.354E+01 4.659E-03 6.360E+03
1.045E-03 1.352E+01 4.794E-03 6.115E+03
1.045E-03 1.352E+01 4.794E-03 6.507E+03
1.153E-03 1.339E+01 5.428E-03 5.418E+03
1.273E-03 1.325E+01 6.120E-03 4.514E+03
1.273E-03 1.325E+01 6.120E-03 4.577E+03
1.354E-03 1.314E+01 6.597E-03 4.050E+03
1.441E-03 1.303E+01 7.107E-03 3.585E+03
1.441E-03 1.303E+01 7.107E-03 3.656E+03
1.500E-03 1.295E+01 7.453E-03 3.367E+03
2.000E-03 1.230E+01 1.034E-02 1.853E+03
3.000E-03 1.102E+01 1.595E-02 7.582E+02
3.552E-03 1.036E+01 1.889E-02 5.154E+02
3.552E-03 1.036E+01 1.889E-02 1.255E+03
3.639E-03 1.026E+01 1.933E-02 1.176E+03
3.728E-03 1.016E+01 1.979E-02 1.102E+03
3.728E-03 1.016E+01 1.979E-02 1.572E+03
4.000E-03 9.854E+00 2.116E-02 1.319E+03
4.303E-03 9.531E+00 2.264E-02 1.101E+03
4.303E-03 9.531E+00 2.264E-02 1.282E+03
5.000E-03 8.840E+00 2.588E-02 8.802E+02
5.182E-03 8.670E+00 2.667E-02 8.030E+02
5.182E-03 8.670E+00 2.667E-02 8.524E+02
5.362E-03 8.509E+00 2.744E-02 7.833E+02
5.548E-03 8.346E+00 2.823E-02 7.198E+02
5.548E-03 8.346E+00 2.823E-02 7.507E+02
6.000E-03 7.970E+00 3.008E-02 6.204E+02
8.000E-03 6.568E+00 3.747E-02 3.041E+02
1.000E-02 5.498E+00 4.397E-02 1.736E+02
1.500E-02 3.734E+00 5.705E-02 6.148E+01
1.717E-02 3.218E+00 6.143E-02 4.334E+01
1.717E-02 3.218E+00 6.143E-02 1.037E+02
2.000E-02 2.689E+00 6.626E-02 6.831E+01
2.095E-02 2.540E+00 6.770E-02 6.039E+01
2.095E-02 2.540E+00 6.770E-02 8.577E+01
2.135E-02 2.482E+00 6.830E-02 8.166E+01
2.176E-02 2.425E+00 6.889E-02 7.775E+01
2.176E-02 2.425E+00 6.889E-02 8.971E+01
3.000E-02 1.585E+00 7.833E-02 3.962E+01
4.000E-02 1.065E+00 8.577E-02 1.868E+01
5.000E-02 7.681E-01 9.035E-02 1.035E+01
6.000E-02 5.781E-01 9.305E-02 6.363E+00
8.000E-02 3.631E-01 9.518E-02 2.937E+00
1.000E-01 2.519E-01 9.510E-02 1.608E+00
1.156E-01 1.974E-01 9.434E-02 1.086E+00
1.156E-01 1.974E-01 9.434E-02 4.602E+00
1.500E-01 1.260E-01 9.159E-02 2.373E+00
2.000E-01 7.544E-02 8.683E-02 1.136E+00
3.000E-01 3.620E-02 7.805E-02 4.051E-01
4.000E-01 2.131E-02 7.107E-02 1.998E-01
5.000E-01 1.402E-02 6.553E-02 1.181E-01
6.000E-01 9.920E-03 6.100E-02 7.808E-02
6.500E-01 8.513E-03 5.902E-02 6.544E-02
6.620E-01 8.220E-03 5.857E-02 6.287E-02
7.000E-01 7.386E-03 5.720E-02 5.571E-02
8.000E-01 5.713E-03 5.396E-02 4.192E-02
9.000E-01 4.551E-03 5.117E-02 3.281E-02
1.000E+00 3.709E-03 4.870E-02 2.654E-02

Bibliografía

[1] Simulación de Montecarlo - Risk Industrial. http://www.palisade-lta.com/risk/simulacion_monte_carlo.asp. Último acceso: 6 de Julio de 2012.

[2] Interfaz de Paso de Mensajes – Wikipedia. http://es.wikipedia.org/wiki/Interfaz_de_Paso_de_Mensajes. Último acceso: 6 de Julio de 2012.

[3] Faiz M. Khan. The Physics of Radiation Therapy. Williams & Wilkins.

[4] G. Nelson and D. Reilly. Gamma-Ray Interactions with Matter.

[5] X-ray Interactions. http://whs.wsd.wednet.edu/faculty/busse/mathhomepage/busseclasses/radiationphysics/lecturenotes/chapter12/chapter12.html. Último acceso: 6 de julio de 2012.

[6] XCOM: Photon Cross Sections Database.http://www.nist.gov/pml/data/xcom/index.cfm. Último acceso: 6 de Julio de 2012.

[7] M. Ljungberg, S-E Strand y M. A. King. Monte Carlo calculations in nuclear medicine. Applications in diagnostic imaging.

[8] Aplicaciones de Montecarlo. Simulación. http://nosvamosamontecarlo.blogspot.com.es/2011/11/aplicaciones-de-montecarlo.html. Último acceso: 6 de Julio de 2012.

[9] Iván Cabria. Radiology Simulations. Apuntes de la asignatura de Protección nuclear.

[10] Periodo de semidesintegración – Wikipedia. http://es.wikipedia.org/wiki/Periodo_de_semidesintegración. Último acceso: 6 de Julio de 2012.

[11] Amdahl's law – Wikipedia. http://en.wikipedia.org/wiki/Amdahl%27s_law. Último acceso: 6 de Julio de 2012.