Avances en el Post-procesamiento de los Códigos de la Plataforma AZTLAN

Armando Gómez Torres, Vicente Xolocostli Munguía
Instituto Nacional de Investigaciones Nucleares
Carretera México - Toluca s/n, La Marquesa, Ocoyoacac, Estado de México, C.P. 52750, México
armando.gomez@inin.gob.mx; vicente.xolocostli@inin.gob

Julian Arturo Duran Gonzalez, Edmundo del Valle Gallegos
Instituto Politécnico Nacional, Escuela Superior de Física y Matemáticas
Unidad Profesional Adolfo López Mateos, Zacatenco, Delegación Gustavo A. Madero, Ciudad de México, C.P. 07738, México.
redfield1290@gmail.com; edmundo.delvalle@gmail.com

Resumen

A medida que el poder de cómputo avanza, se vuelve más común el uso de potentes solucionadores numéricos, pero también de herramientas de pre y post procesamiento tanto para facilitar la creación de archivos de entrada como para hacer un mejor análisis de resultados. Las herramientas de preprocesamiento están basadas en una interfaz gráfica para el usuario mediante la cual el usuario interactúa de forma simple para crear un archivo de entrada que tradicionalmente se construía en la forma de un archivo de texto. También las herramientas de preprocesamiento permiten revisar y corregir detalles como la malla, la distribución de los materiales y errores comunes de definición de parámetros antes de ejecutar el código y de esta manera evitar perder tiempo obteniendo resultados que no corresponden a lo que se deseaba modelar. Para el caso del post procesamiento, es también de vital importancia facilitar la extracción de información del archivo de salida y, en la medida de lo posible simplificar el análisis de resultados por medio de imágenes y/o videos que permitan visualizar con precisión los puntos bajo estudio. Para la plataforma AZTLAN, se han comenzado las implementaciones para pre y post procesamiento con base en la biblioteca de fuente abierta MED y HDF5. Las primeras implementaciones se han realizado para los códigos AZTRAN y AZKIND y los primeros resultados se presentan en este trabajo. Cabe señalar que una aproximación similar se usará para resolver el acoplamiento neutrónico – termohidráulico más adelante.

1. INTRODUCCIÓN

Los grandes esfuerzos internacionales relacionados con el desarrollo de software para reactores nucleares, además de la optimización de los solucionadores numéricos, se enfoca en ofrecer al usuario final las facilidades para poder realizar análisis más rápido y con mayor precisión. Por estas razones, actualmente se invierte una fuerte cantidad en el desarrollo de herramientas para el pre y post procesamiento tanto para facilitar la creación de archivos de entrada como para hacer

un mejor análisis de resultados. Las herramientas de preprocesamiento están basadas en una interfaz gráfica para el usuario mediante la cual el usuario interactúa de forma simple para crear un archivo de entrada que tradicionalmente se construía en la forma de un archivo de texto. También las herramientas de preprocesamiento permiten revisar y corregir detalles como la malla, la distribución de los materiales y errores comunes de definición de parámetros antes de ejecutar el código y de esta manera evitar perder tiempo obteniendo resultados que no corresponden a lo que se deseaba modelar. Para el caso del post procesamiento, es también de vital importancia facilitar la extracción de información del archivo de salida y, en la medida de lo posible simplificar el análisis de resultados por medio de imágenes y/o videos que permitan visualizar con precisión los puntos bajo estudio. Existen en la actualidad diversos programas auxiliares comerciales y de licencia libre que permiten al usuario final visualizar los modelos y los resultados. La implementación del lado del desarrollador no siempre es simple, pues muchas veces se tiene que invertir un esfuerzo considerable en hacer las implementaciones al código fuente de manera que sean transparentes para el usuario final en la medida de lo posible. Para la plataforma AZTLAN [1], se han comenzado las implementaciones para pre y post procesamiento con base en la biblioteca de fuente abierta MED y HDF5. Estas herramientas permiten crear tanto la malla como la matriz de resultados conforme se calcula, es decir, directamente en el código fuente por lo que no es necesario correr ningún programa o script adicional al inicio o al final de la ejecución para poder tener el archivo con extensión "***.med***" que es el que se visualiza en la plataforma de fuente abierta SALOME. Las primeras implementaciones se han realizado para los códigos AZTRAN y AZKIND y los primeros resultados se presentan en este trabajo. Cabe señalar que una aproximación similar se usará para resolver el acoplamiento neutrónico – termohidráulico más adelante.

2. BIBLIOTECA DE FUNCIONES MED

En la plataforma de fuente abierta SALOME, se usa el formato MED (DEM: Data Exchange Model) para la manipulación de la malla y de los campos de resultados [2]. El intercambio de datos se lleva a cabo ya sea por medio de archivos tipo MED (file formalism) o directamente en memoria (memory formalism). Por supuesto que el uso eficiente de cada uno de los formalismos dependerá de la aplicación. Así, si lo que se quiere es tener un intercambio eficiente de información entre dos o más códigos, un intercambio de información en memoria puede ser lo más óptimo. Sin embargo, si lo que se desea es tener concentrados todos los resultados en un archivo de salida para un posterior análisis detallado de resultados, entonces la escritura en archivo de salida ***.med*** es lo conveniente [3]. De igual forma, una combinación de las dos metodologías es posible, haciendo intercambios en memoria y escribiendo datos en archivo cuando sea necesario, por ejemplo.

Las bibliotecas MED – están organizadas en múltiples capas:

i. La capa MED–file: API en C y FORTRAN para implementar los objetos mesh (malla) y field (campos de resultados).

ii. La capa MED–memory: API en C++ para crear y manipular la malla y los campos en memoria.

iii. Python API: generada usando SWIG, la cual envuelve por completo a la API en C++ para manipulación de memoria.

iv. CORBA API: usada para simplificar el cálculo distribuido dentro de SALOME.

v. Clases MED–Client: usadas para simplificar y optimizar la interacción de objetos distantes con el solucionador local.

Gracias a la MED–memory, cualquier componente puede accesar una malla o un campo distante, de tal forma que dos programas ejecutando en máquinas distintas pueden intercambiar malla y campos. Esas mallas y campos se pueden leer y escribir fácilmente en un archivo MED habilitando el acceso a la suite completa de herramientas de SALOME (CAD, mallado, visualización, etc.)

Aunque la creación de archivos MED se puede hacer directamente en el código fuente en FORTRAN o C, la implementación de memoria MED sólo se puede hacer con una clase en C++. Así que, si asumimos que un código fuente está desarrollado en FORTRAN, al menos dos estrategias pueden explorarse:

1. Creación directa de los archivos MED en el archivo fuente mediante el uso de las funciones en la API MED FORTRAN, seguida por una lectura y vaciado para intercambiar datos y escribir los resultados en archivo, o

2. La creación de una memoria MED directamente con una interfaz a la clase en C++ para hacer el intercambio en memoria seguido por una escritura final para poder hacer el post procesamiento.

Ambas opciones tienen sus ventajas y desventajas. La primera opción permite el uso de SALOME [4] como un post-procesador sin tener que integrar el código en la plataforma (únicamente abriendo el archivo MED que tiene los resultados para propósitos de visualización), sin embargo implica un esfuerza redundante (escribir archivos MED en FORTRAN y en C++). La segunda opción es más directa, la creación de la memoria MED en la API en C++ para intercambio, visualización en línea y la escritura de archivos MED para propósitos de post procesamiento, pero con la desventaja de que el código se debe de integrar en SALOME o al menos compilar por medio de una API en C++, lo cual implica un esfuerzo considerable en la parte de programación.

Para los fines de este trabajo y con el objetivo de darle portabilidad a los códigos de la plataforma SALOME, la primera estrategia es la que se describe y se usa en esta etapa del proyecto, aunque la segunda estrategia será la que se lleve a cabo en las etapas finales del proyecto.

2.1. MED-memory API

Para comprender un poco más cómo funcionan las bibliotecas MED, se describirá brevemente la filosofía de la memoria MED.

La librería MED-memory usa el espacio de nombre (namespace): MEDMEM el cual es el *namespace* general en donde las clases principales se definen. Para el nivel de uso más elemental, la API consiste en pocas clases localizadas en el namespace MEDMEM (ver Figura 1):

- MED: el contenedor global;
- MESH: la clase que contiene los objetos de la malla en 2D o 3D;
- SUPPORT: la clase que contiene una lista de los elementos de la malla;
- FIELD: la clase que contiene principalmente una lista de valores sobre un soporte particular.

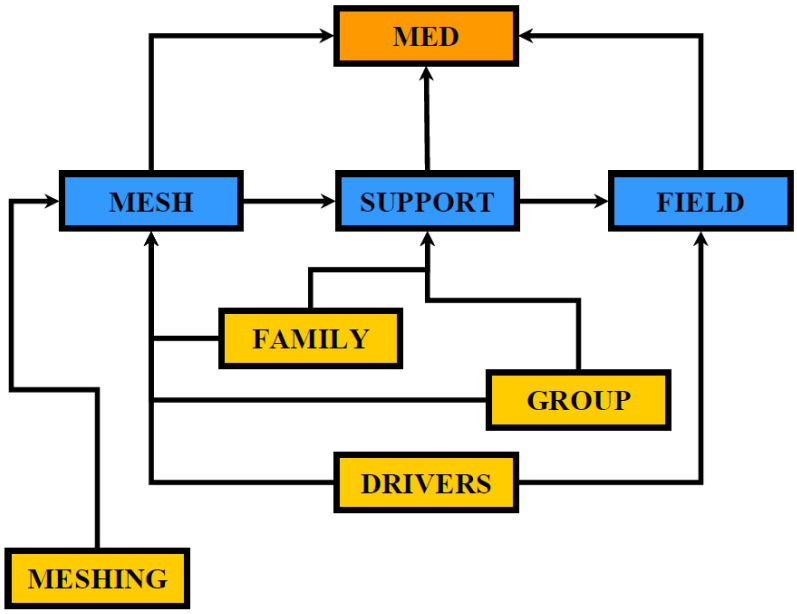

Figura 1. Estructura de las clases en la API de MED memory

Un uso más Avanzado de la memoria MED es posible a través de otras clases que incluyen:

- GROUP: una clase heredada del SUPPORT que se usa para crear soportes enlazados a grupos de la malla. Un grupo almacena una lista restrictiva de elementos usados para establecer condiciones iniciales o de frontera.
- FAMILY: es una clase usada para manipular un cierto tipo de soporte que no se intercepta con ningún otro.
- MESHING: una clase para construir mallas desde lo básico (scratch), se puede usar para transformar mallas de algún formato específico a MED o para integrar un generador de malla en la plataforma SALOME.
- DRIVERS: le permite al usuario tener un control fino de las operaciones de I/O.

La Figura 1 muestra cómo el contenedor MED controla el ciclo de vida de los objetos que contiene: el destructor del contenedor principal destruirá todos los objetos que le sean agregados. Por otro lado, los ciclos de vida de los objetos MESH, SUPPORT y FIELD son independientes, es decir, si se destruye un SUPPORT, por ejemplo, no tendrá ningún efecto sobre la MESH que se

refiere a ese SUPPORT. Pero es importante mantener el enlace: un MESH agrega a un SUPPORT el cual a su vez agrega a un FIELD. Si se necesita eliminar objetos de la memoria MED, el FIELD se debe borrar primero, después el SUPPORT y finalmente el MESH.

2.1.1 Crear una malla

La clase MESHING es la necesaria cuando se necesita crear una malla desde "scratch". Este proceso se debe realizar usualmente después de leer el archivo de entrada. El código debe definir su dominio y la malla para un vaciado de resultados mediante una representación FIELD. La creación de un objeto MESHING implica la definición de coordenadas y conectividades:

Coordenadas: El primer paso es la definición de las coordenadas de la malla, los argumentos típicos de una malla son:

- SpaceDimension: Tipo entero con las dimensiones del dominio considerado.
- NumberOfNodes: Tipo entero con el número total de nodos considerado.
- Coordinates: Tipo array de doble precisión que contiene las coordenadas de los puntos que definen la malla.
- System: Tipo "string" con el Sistema de coordenadas deseado: "CARTESIAN", "CYLINDRICAL", "SPHERICAL".
- Mode: Tipo "string" definiendo la manera en que se introducen las coordenadas: Las coordenadas pueden ser escritas en una forma entrelazada MED_FULL_INTERLACE (x1, y1, z1, x2, y2, z2, …) o sin entrelazar MED_NO_INTERLACE (x1, x2… y1, y2… z1, z2 …).

Connectivities: una vez que las coordenadas se han definido, el siguiente paso es definir las conectividades de las coordenadas. Para cada tipo de elemento, la conectividad se usa para crear celdas (cells), bordes (edges) o caras (faces) de elementos. Primero que todo, es necesario definir la conectividad de los elementos celdas (cells). Después de eso, es posible definir la conectividad de las caras y bordes si es que fuera necesario. Hay varios tipos de conectividades basados en el tipo de celda que se desee. Por ejemplo, una celda en una dimensión (un segmento) puede crearse con dos vértices, un triángulo con tres y un cuadrado con cuatro. El caso con poliedros es más complejo. Las conectividades deben seguir una regla para construir cada poliedro basado en las coordenadas de los vértices. En el caso de un núcleo con geometría rectangular, por ejemplo, los hexaedros son la mejor opción para el mallado. La Figura 2 muestra un hexaedro (cubo) y el orden en el que las conectividades deben darse, con base en los vértices [MED2003]. Para cada tipo de conectividad considerada, se deben usar las siguientes funciones en el orden en que aparecen:

- setNumberOfTypes: establece el número de tipos geométricos diferentes.
- setTypes: establece los diferentes tipos de geometrías (MED_TETRA4, MED_PYRA5,
- MED_HEXA8, etc… [medMEM]). Los tipos se deben de proporcionar en orden ascendente de nodos de ese tipo.
- setNumberOfElements: establece el número de elementos de cada tipo geométrico.
- setConnectivity: establece la conectividad de cada tipo geométrico.

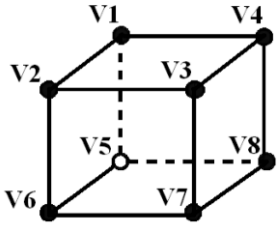

Figura 2. Orden de las conectividades en un hexaedro (cubo)

2.1.2 Creación de un FIELD

Una vez que la malla está creada, la creación de FIELD es inmediata y se requiere únicamente de un método en donde se especifica el tipo de datos que se va a almacenar en cada celda de la malla, el tiempo asociado a ese dato, o la iteración y proporcionar un arreglo (array) con los datos en el orden adecuado, es decir, en el orden en que se definieron las celdas con las conectividades. En la siguiente sección se presentará un caso práctico.

3. IMPLEMENTACIÓN MED EN AZTRAN

En la Figura 3 se muestra una porción del código fuente usado para la creación de las coordenadas. La Figura 4 muestra lo mismo para las conectividades. Adicionalmente, en la Figura 5 se muestra la definición del módulo para hacer uso de las funciones necesarias y en las Figuras 6 y 7 se muestran las funciones para la creación del archivo MED.

```
gomez@ubuntu: ~/Downloads
        deb_ind = 1
        ind_ijk = 1
        write(dbg,*)'coordinates = ',coordsz
      do k = 1,nemz+1

        do j = 1,nemy+1
          do i = 1,nemx+1
                write(dbg,*)'number = ',deb_ind
                coord(ind_ijk) = x_mesh(i)
                write(dbg,*)'coor x = ',coord(ind_ijk)
                coord(ind_ijk+1) = y_mesh(j)
                write(dbg,*)'coor y = ',coord(ind_ijk+1)
                coord(ind_ijk+2) = z_mesh(k)
                write(dbg,*)'coor z = ',coord(ind_ijk+2)
                ind_ijk = ind_ijk + 3
                deb_ind = deb_ind + 1
          enddo
        enddo
      enddo
                                                    45,11         30%
```

Figura 3. Creación de coordenadas en AZTRAN

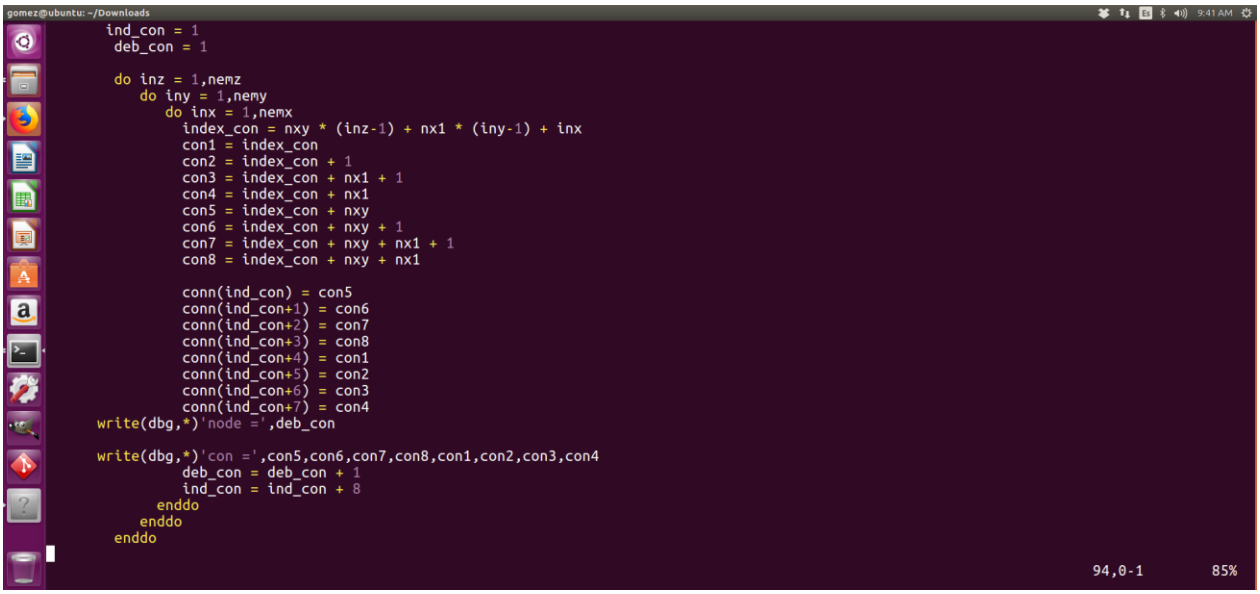

Figura 4. Creación de conectividades en AZTRAN

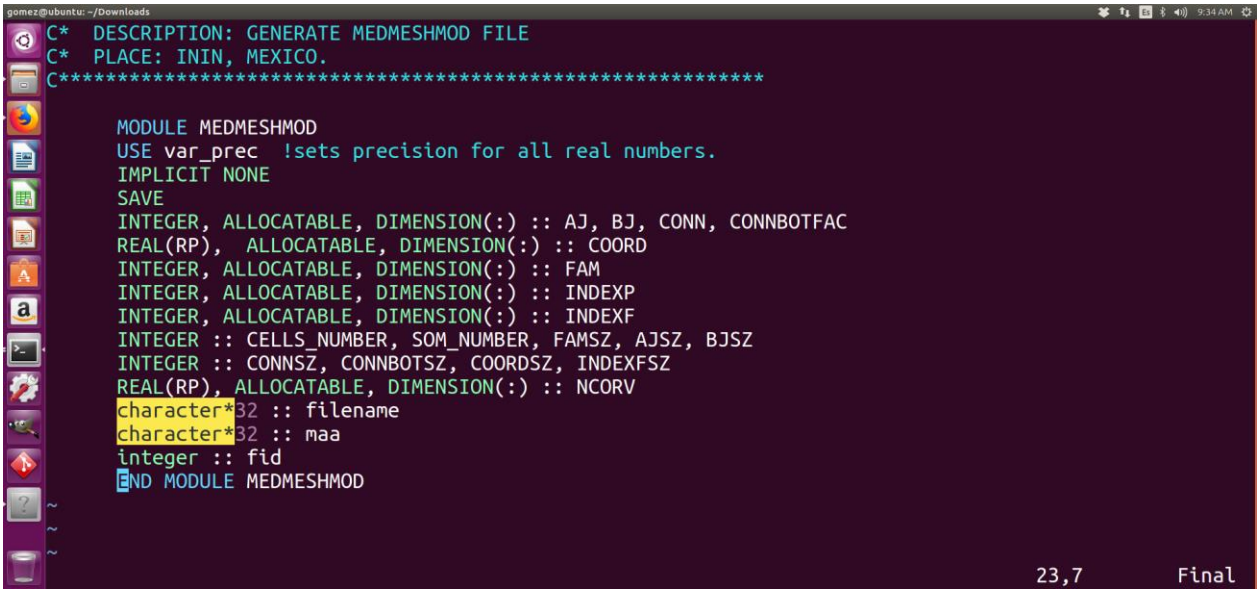

Figura 5. Módulo con variables necesarias para la implementación MED

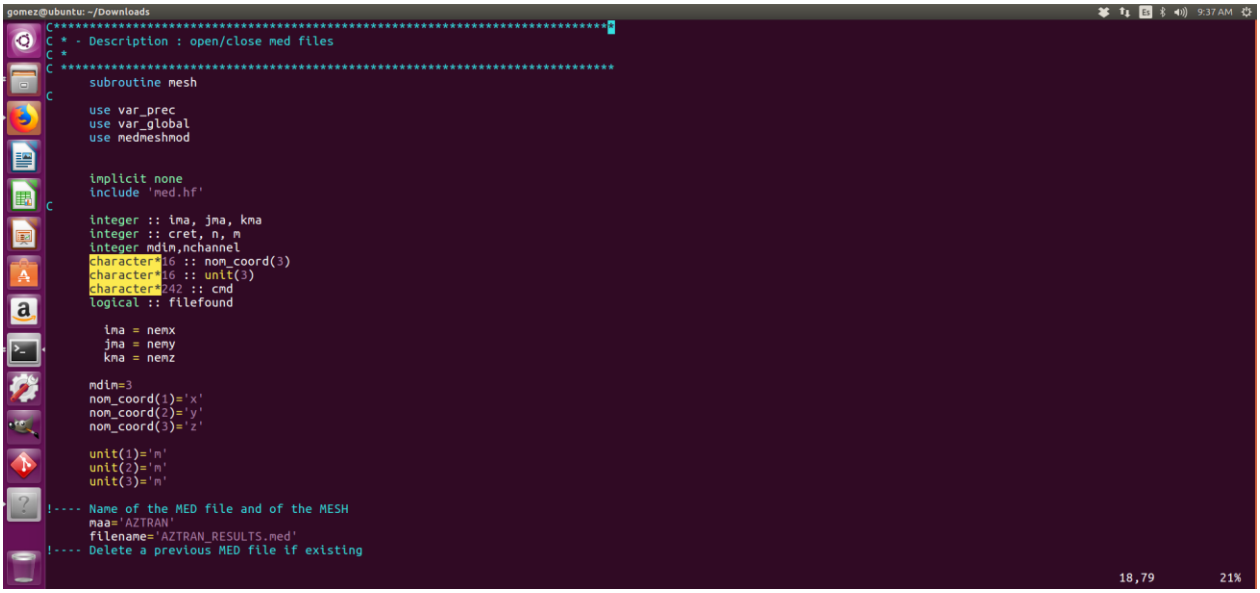

Figura 6. Creación de un archivo MED

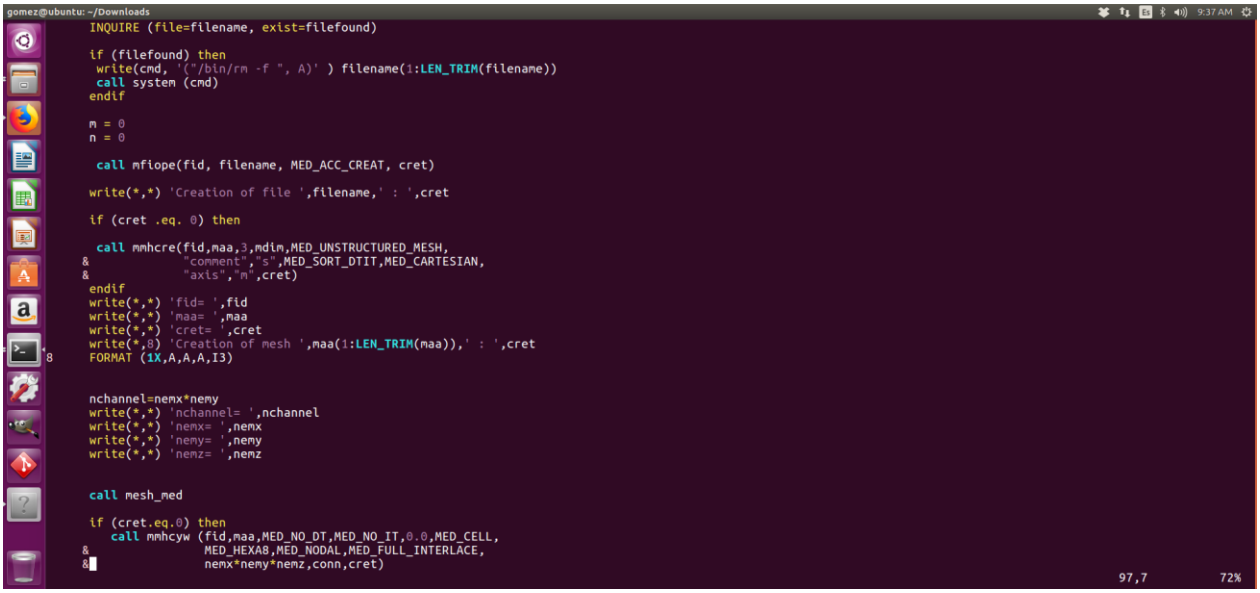

Figura 7. Creación y apertura de archivo MED

En la Figura 8 se presenta el cálculo de la potencia y la escritura del FIELD “power” en el archivo MED para cada celda definida con anterioridad en la malla.

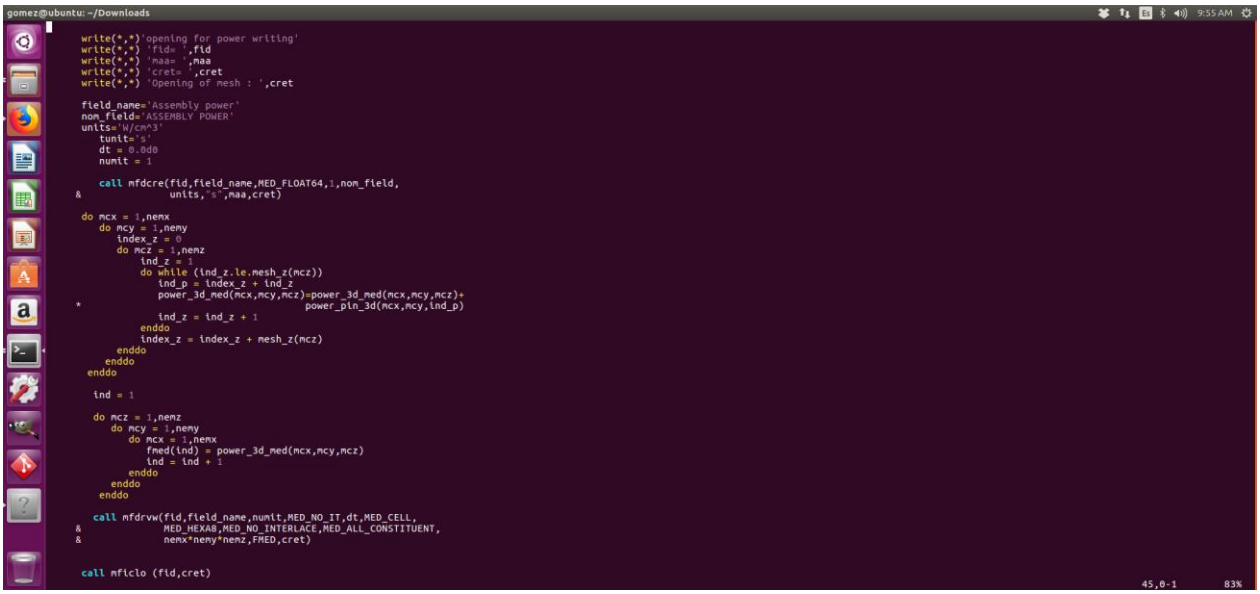

Figura 8. Cálculo y escritura de la potencia en un archivo MED.

4. ALGUNOS RESULTADOS

En la Figura 9 se muestra la malla para un caso sencillo de un cubo dividido en una malla de 10 x 10 x 10. Es posible en esta etapa hacer una verificación de la malla usando la plataforma SALOME que es en donde se visualiza.

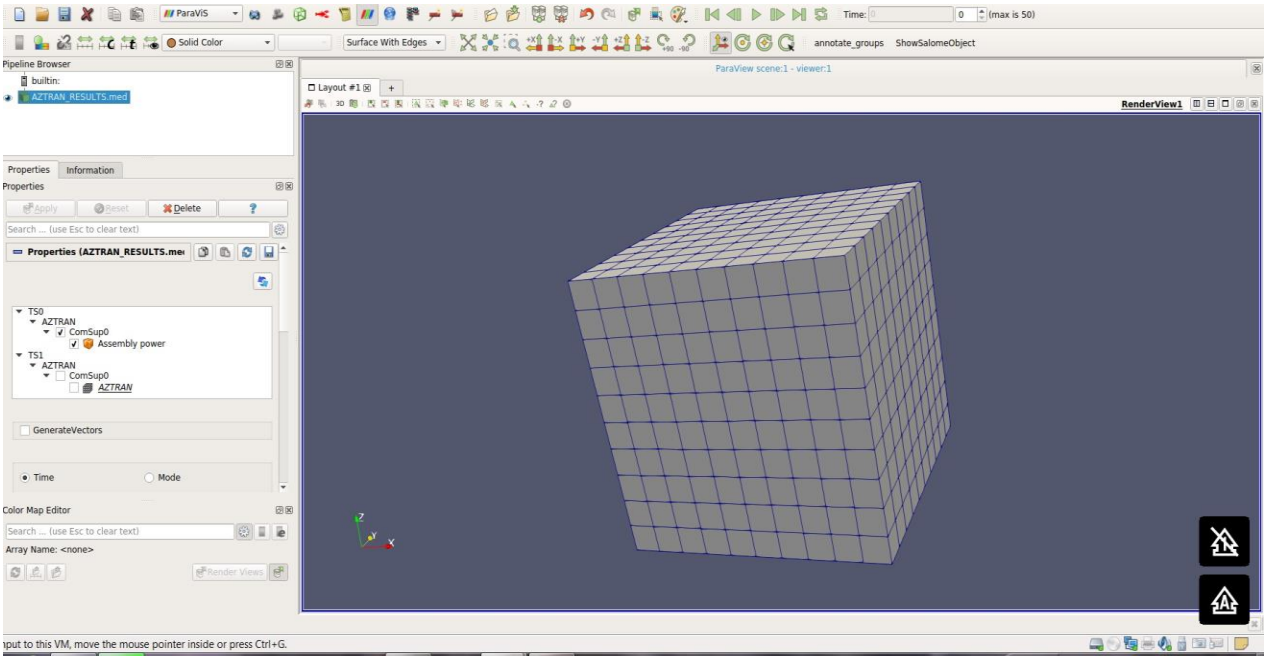

Figura 9. Visualización de la malla en SALOME

Finalmente, en las Figuras 10 y 11 se muestra la potencia para un ensamble calculada con el código AZTRAN y para un núcleo completo calculada con el código AZKIND.

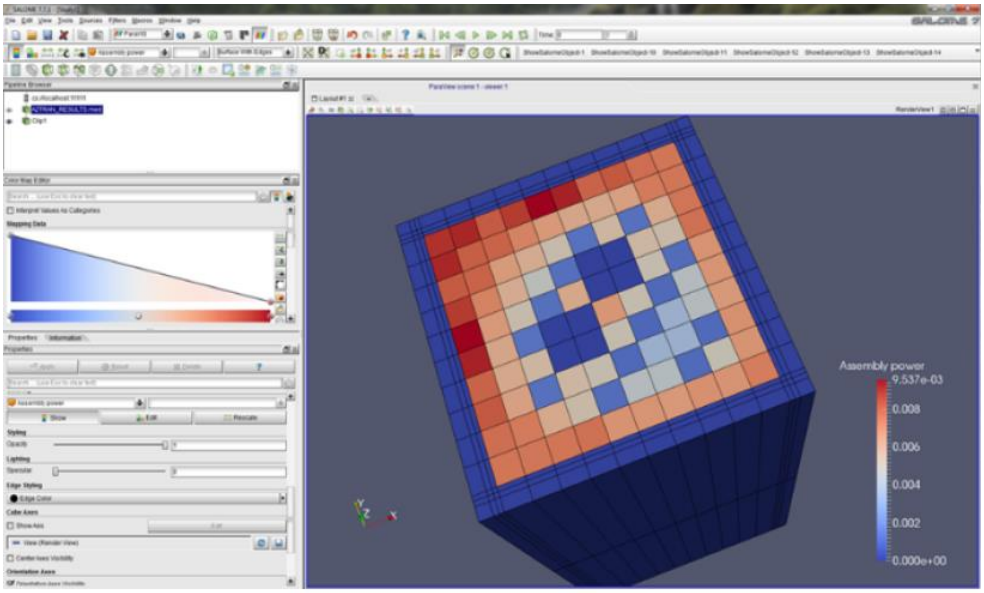

Figura 10. Visualización de la potencia para un ensamble calculada con el código AZTRAN

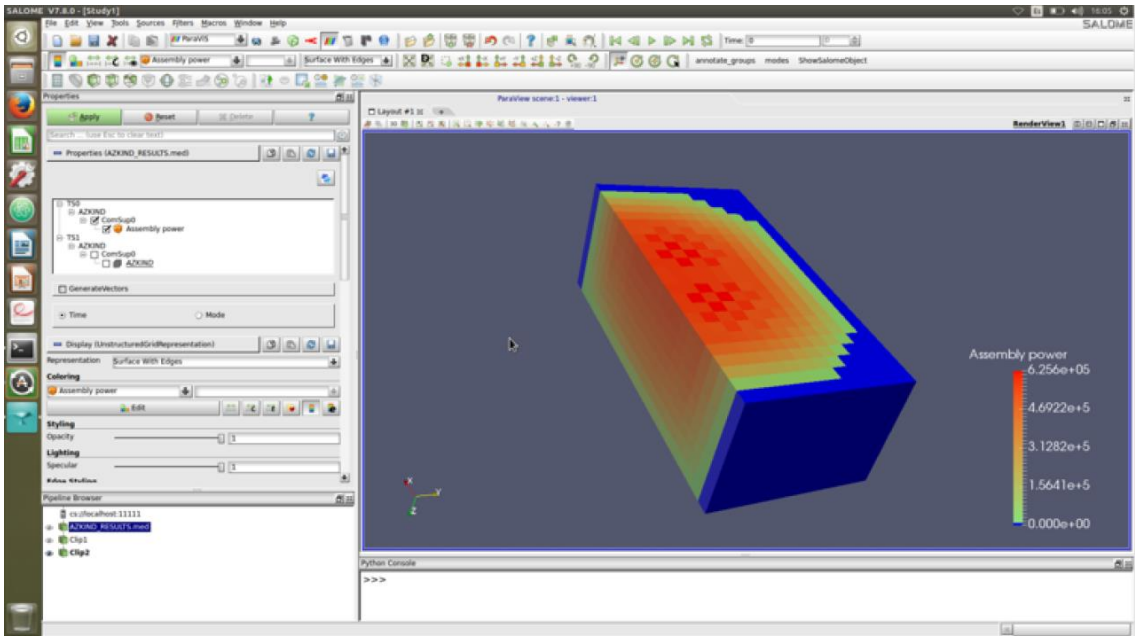

Figura 11. Visualización de la potencia para un núcleo calculada con el código AZKIND

4. CONCLUSIONES

En este trabajo se ha demostrado la implementación de las capacidades de pre y post procesamiento para los códigos en coordenadas cartesianas de la plataforma AZTLAN. Este es sólo un primer paso con miras a dotar a los códigos de muchas más capacidades para que el usuario pueda hacer mejores análisis de resultados y pueda hacer también un consistente pre procesamiento. Más adelante se agregarán las subrutinas termo-hidráulicas y se resolverá la comunicación neutrónica termo-hidráulica por medio del uso de bibliotecas MED, lo cual eficientará la comunicación y facilitará de igual forma el post procesamiento.

AGRADECIMIENTOS

Los autores agradecen el apoyo financiero recibido del proyecto estratégico No. 212602 (AZTLAN Platform) del Fondo Sectorial de Sustentabilidad Energética CONACYT – SENER.

REFERENCIAS

1. Armando M. Gómez Torres, et. al., “AZTLAN Platform: Plataforma Mexicana para el Análisis y Diseño de Reactores Nucleares”, XXV Congreso Anual de la Sociedad Nuclear Mexicana, Boca del Río, Veracruz, 31 de Agosto a 4 de Septiembre (2014).
2. EDF R&D, MED/DEM data exchange model definition, version 2.2. 2003.
3. P.Godbronn, E. Fayolle, N. Bouhamou, J. Roy and N. Crouzet, MEDMEM user’s guide, CEA, Rapport DM2S, July 2006.
4. The open source Integration Platform for Numerical Simulation. http://www.salome-platform.org