Aproximación Paralela del Código Termohidráulico de Subcanal COBRA-TF

Enrique Ramos[a], Agustín Abarca[b], Jose E. Roman[a],
Rafael Miró[b]

[a]Dpto. de Sistemas Informáticos y Computación,
Universitat Politècnica de València,
Camí de Vera s/n, 46022 Valencia, España
ramos@dsic.upv.es, jroman@dsic.upv.es

[b] Inst. de Seguridad Industrial, Radiofísica y Medioambiental (ISIRYM)
Universitat Politècnica de València,
Camí de Vera s/n, 46022 Valencia, España
aabarca@isirym.upv.es, rmiro@iqn.upv.es

Resumen *– La paralelización permite el uso de numerosos procesadores (o núcleos) que colaboren en la obtención de la solución a un único problema. Esto no sólo reduce el tiempo de computación, sino que además aumenta la cantidad de memoria disponible en un clúster. Por tanto pueden resolverse problemas de gran dimensión.*

Con objeto de validar la paralelización de COBRA-TF se ha modelado a nivel de subcanal un elemento combustible tipo PWR y se ha simulado un pequeño estacionario, obteniéndose una considerable aceleración en la simulación en paralelo comparado con el caso secuencial.

1. INTRODUCCIÓN

En ingeniería nuclear es habitual el uso de códigos de simulación numérica para analizar el comportamiento de los diferentes componentes de las plantas nucleares, tanto en funcionamiento normal del reactor como en situaciones menos frecuentes como pueden ser la recarga de combustible y secuencias accidentales. La elevada exigencia en términos de seguridad hace necesario utilizar modelos de gran nivel de detalle, lo que conlleva un alto coste computacional de las simulaciones. La paralelización de los códigos es muy conveniente en este contexto, para hacer eficiente este tipo de estudios.

COBRA-TF (Coolant Boiling in Rod Arrays Code – Two Fluids) es un código termohidráulico de subcanal (permite resolver problemas a nivel de varilla de combustible nuclear) que utiliza dos campos y tres fases para modelar el flujo bifásico (agua+vapor) [1], [2]. Las tres fases son el vapor, el líquido continuo y el líquido presente a modo de gotas dentro del vapor en ciertos regimenes. Cada fase utiliza un conjunto de tres ecuaciones tridimensionales para la masa, momento y energía, con una excepción: se utiliza una ecuación de la energía para ambas fases líquidas, la continua y el líquido presente en la fase vapor. La formulación de dos fluidos emplea un conjunto separado de ecuaciones de conservación y relaciones constitutivas para cada fase. Los efectos de una fase en otra se tienen en cuenta en los términos de interacción que aparecen en la correspondiente ecuación de gobierno. Las ecuaciones de conservación tienen la misma forma para cada fase; solamente se diferencian en las relaciones constitutivas y las propiedades físicas.

En este trabajo se presenta la primera fase de la paralelización del código COBRA-TF, la cual ha consistido en la integración con una librería paralela para resolución de sistemas de ecuaciones, concretamente PETSc [3], [4], [5]. Además se ha paralelizado la

generación del Jacobiano, de modo que cada procesador se encarga de la generación de los coeficientes asociados a un subconjunto de celdas. En las secciones 2 y 3 se describen algunos detalles del código que son relevantes para la paralelización. Las secciones 4, 5 y 6 describen la librería PETSc y la estrategia utilizada, y los resultados preliminares se muestran en la sección 7.

2. MODELO

Para realizar las simulaciones con el código COBRA-TF se ha modelado un elemento combustible tipo PWR genérico de un reactor nuclear de agua a presión (PWR). Este elemento combustible tiene 3.4 m de longitud activa y consta de 236 varillas de combustible y 20 tubos guía. En la figura 1 podemos ver una esquema de este tipo de combustibles.

Figura 1. Elemento combustible para reactor PWR.

El modelo realizado mediante COBRA-TF consta de 256 canales termohidráulicos, 236 de los cuales están calefactados ya que presentan combustible nuclear y 20 representan los tubos guía de las barras de control. Axialmente el modelo se ha dividido en 100 nodos, con lo que en conjunto se obtienen 25600 celdas computacionales por cada elemento combustible. Se han modelado mediante coeficientes de pérdidas de presión las 6 rejillas separadoras situadas en los niveles axiales 11, 26, 41, 55, 70 y 85 del elemento combustible. Además en el primer y último nodo se modelan, también mediante coeficientes de pérdidas de presión, los cabezales del combustible. La disposición radial de varillas de combustible y tubos guía dentro de la matriz de combustible se presenta en la figura 2.

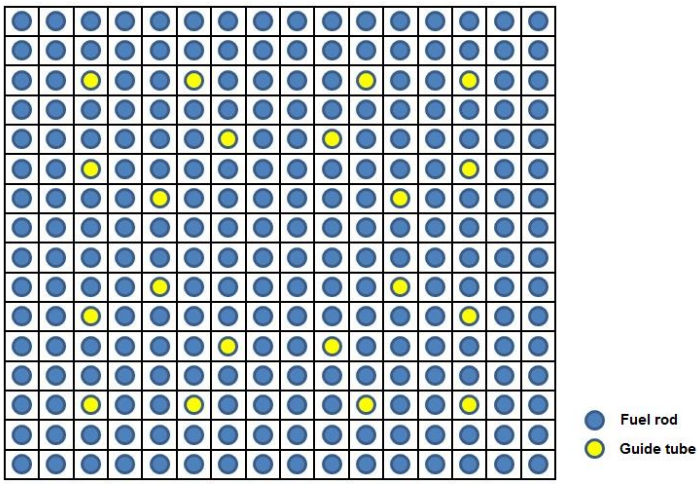

Figura 2. Esquema radial del combustible modelado.

Se realiza una simulación del estado estacionario de un elemento combustible como el descrito anteriormente. Las condiciones de contorno son de caudal másico y temperatura a la entrada y presión a la salida. Además, la potencia se mantiene constante a lo largo de toda la simulación.

3. RESOLUCIÓN NUMÉRICA

El primer paso de la resolución consiste en resolver las ecuaciones de momento de cada celda por eliminación Gaussiana, cuya forma matricial se muestra en la ecuación (1).

$$\begin{bmatrix} c_1 - 1 & d_1 & 0 \\ c_2 & d_2 - 1 & e_2 \\ 0 & d_3 & e_3 - 1 \end{bmatrix} \begin{Bmatrix} f_l \\ f_v \\ f_e \end{Bmatrix} = \begin{Bmatrix} -a_1 - b_1 \Delta P \\ -a_2 - b_2 \Delta P \\ -a_3 - b_3 \Delta P \end{Bmatrix} \tag{1}$$

El segundo paso nos permite calcular las velocidades, necesarias para la linealización de las ecuaciones de masa y energía. Mediante el método de Newton-Raphson [6] por bloques se obtiene la variación de las variables independientes que garantizan un error residual nulo, cuya ecuación podemos ver en la ecuación (2).

$$\begin{aligned} E_{\mathrm{CV}} = & \frac{\left[(\alpha_v \rho_v)_j^n - (\alpha_v \rho_v)_j\right] A_{c_j}}{\Delta t} + \sum_{\mathrm{KA}=1}^{\mathrm{NA}} \frac{\left[(\alpha_v \rho_v)^* \widetilde{U}_{v_j} A_{m_j}\right]_{\mathrm{KA}}}{\Delta x_j} \\ & - \sum_{\mathrm{KB}=1}^{\mathrm{NB}} \frac{\left[(\alpha_v \rho_v)^* \widetilde{U}_{v_{j-1}} A_{m_{j-1}}\right]_{\mathrm{KB}}}{\Delta x_j} - \sum_{L=1}^{\mathrm{NKK}} S_L \left[(\alpha_v \rho_v)^* \widetilde{V}_{v_L}\right]_j \\ & - \frac{\Gamma_j}{\Delta x_j} - \frac{S_{cv_j}}{\Delta x_j} \end{aligned} \tag{2}$$

$$\begin{bmatrix}
\frac{\partial E_{\mathrm{CG}}}{\partial \alpha_g} & \frac{\partial E_{\mathrm{CG}}}{\partial \alpha_v} & \frac{\partial E_{\mathrm{CG}}}{\partial \alpha_v h_v} & \frac{\partial E_{\mathrm{CG}}}{\partial (1-\alpha_v) h_l} & \frac{\partial E_{\mathrm{CG}}}{\partial \alpha_e} & \frac{\partial E_{\mathrm{CG}}}{\partial P_j} & \frac{\partial E_{\mathrm{CG}}}{\partial P_{i=1}} & \cdots & \frac{\partial E_{\mathrm{CG}}}{\partial P_{\mathrm{NCON}}} \\
\frac{\partial E_{\mathrm{CL}}}{\partial \alpha_g} & \frac{\partial E_{\mathrm{CL}}}{\partial \alpha_v} & \frac{\partial E_{\mathrm{CL}}}{\partial \alpha_v h_v} & \frac{\partial E_{\mathrm{CL}}}{\partial (1-\alpha_v) h_l} & \frac{\partial E_{\mathrm{CL}}}{\partial \alpha_e} & \frac{\partial E_{\mathrm{CL}}}{\partial P_j} & \frac{\partial E_{\mathrm{CL}}}{\partial P_{i=1}} & \cdots & \frac{\partial E_{\mathrm{CL}}}{\partial P_{\mathrm{NCON}}} \\
\frac{\partial E_{\mathrm{CV}}}{\partial \alpha_g} & \frac{\partial E_{\mathrm{CV}}}{\partial \alpha_v} & \frac{\partial E_{\mathrm{CV}}}{\partial \alpha_v h_v} & \frac{\partial E_{\mathrm{CV}}}{\partial (1-\alpha_v) h_l} & \frac{\partial E_{\mathrm{CV}}}{\partial \alpha_e} & \frac{\partial E_{\mathrm{CV}}}{\partial P_j} & \frac{\partial E_{\mathrm{CV}}}{\partial P_{i=1}} & \cdots & \frac{\partial E_{\mathrm{CV}}}{\partial P_{\mathrm{NCON}}} \\
\frac{\partial E_{\mathrm{CE}}}{\partial \alpha_g} & \frac{\partial E_{\mathrm{CE}}}{\partial \alpha_v} & \frac{\partial E_{\mathrm{CE}}}{\partial \alpha_v h_v} & \frac{\partial E_{\mathrm{CE}}}{\partial (1-\alpha_v) h_l} & \frac{\partial E_{\mathrm{CE}}}{\partial \alpha_e} & \frac{\partial E_{\mathrm{CE}}}{\partial P_j} & \frac{\partial E_{\mathrm{CE}}}{\partial P_{i=1}} & \cdots & \frac{\partial E_{\mathrm{CE}}}{\partial P_{\mathrm{NCON}}} \\
\frac{\partial E_{\mathrm{EL}}}{\partial \alpha_g} & \frac{\partial E_{\mathrm{EL}}}{\partial \alpha_v} & \frac{\partial E_{\mathrm{EL}}}{\partial \alpha_v h_v} & \frac{\partial E_{\mathrm{EL}}}{\partial (1-\alpha_v) h_l} & \frac{\partial E_{\mathrm{EL}}}{\partial \alpha_e} & \frac{\partial E_{\mathrm{EL}}}{\partial P_j} & \frac{\partial E_{\mathrm{EL}}}{\partial P_{i=1}} & \cdots & \frac{\partial E_{\mathrm{EL}}}{\partial P_{\mathrm{NCON}}} \\
\frac{\partial E_{\mathrm{EV}}}{\partial \alpha_g} & \frac{\partial E_{\mathrm{CG}}}{\partial \alpha_v} & \frac{\partial E_{\mathrm{EV}}}{\partial \alpha_v h_v} & \frac{\partial E_{\mathrm{EV}}}{\partial (1-\alpha_v) h_l} & \frac{\partial E_{\mathrm{EV}}}{\partial \alpha_e} & \frac{\partial E_{\mathrm{EV}}}{\partial P_j} & \frac{\partial E_{\mathrm{EV}}}{\partial P_{i=1}} & \cdots & \frac{\partial E_{\mathrm{EV}}}{\partial P_{\mathrm{NCON}}}
\end{bmatrix}$$

$$\cdot \begin{Bmatrix}
\delta\alpha_g \\ \delta\alpha_v \\ \delta\alpha_v h_v \\ \delta(1-\alpha_v) h_l \\ \delta\alpha_e \\ \delta P_j \\ \delta P_{i=1} \\ \vdots \\ \delta P_{i=\mathrm{NCON}}
\end{Bmatrix} = - \begin{Bmatrix}
E_{\mathrm{CG}} \\ E_{\mathrm{CL}} \\ E_{\mathrm{CV}} \\ E_{\mathrm{CE}} \\ E_{\mathrm{EL}} \\ E_{\mathrm{EV}}
\end{Bmatrix} \qquad (3)$$

Aplicando Newton-Raphson podemos obtener las ecuaciones matriciales para cada celda que se muestran en la ecuación (3) en la que podemos ver el Jacobiano del sistema de ecuaciones evaluado para las variables independientes, y compuesto por las derivadas analíticas de cada ecuación con respecto a la variación lineal de las variables independientes. También podemos observar el vector solución que contiene las variaciones lineales y el vector de error.

Una vez resuelta la ecuación se reduce para obtener la solución para las variables independientes, derivando una ecuación para cada celda, lo que obliga a resolver un sistema de ecuaciones formado por todas las celdas. Si se alcanza la convergencia se pasa al paso de tiempo siguiente, de lo contrario se disminuye el paso de tiempo de la simulación y se repite el proceso.

En resumen, cada iteración externa se compone principalmente de dos partes: el cálculo de los coeficientes y la resolución de un sistema de ecuaciones lineales. La matriz coeficiente de este sistema de ecuaciones es dispersa (el porcentaje de elementos no nulos es muy bajo). Esto se debe al método de discretización utilizado, concretamente se trata del método de diferencias finitas. Es importante aprovechar esta característica para resolver el sistema eficientemente, utilizando métodos iterativos. En este trabajo, se ha abordado la paralelización de la resolución del sistema y del cálculo de los coeficientes mediante el reparto de las celdas del modelo entre varios procesos MPI.

4. PETSC

PETSc (Portable Extensible Toolkit for Scientific Computation, es un software numérico orientado a objetos para la resolución de ecuaciones en derivadas parciales, paralelizado mediante paso de mensajes. Está siendo utilizado en muchas aplicaciones científicas en todo el mundo. En PETSc todo el código se construye alrededor de una serie de estructuras de datos y algoritmos que han sido encapsuladas mediante técnicas de orientación a objetos. Su ventaja consiste en trabajar directamente con estos objetos abstrayéndose de la estructura de datos interna, lo que le da una gran potencia al permitir

programar los métodos numéricos y aplicaciones habituales sin tener que estar pendiente de multitud de detalles de implementación relativos a las estructuras de datos o la paralelización. En la figura 3 podemos ver sus componentes principales.

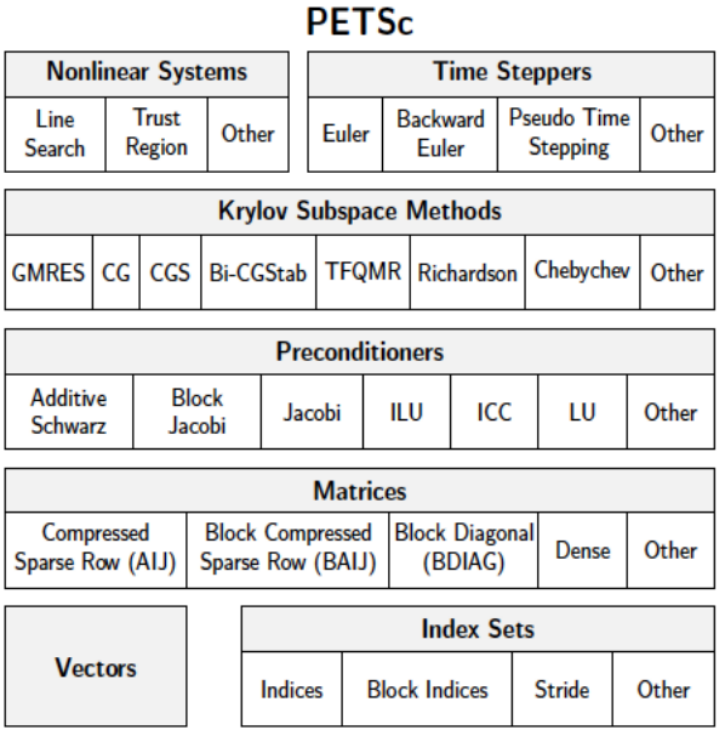

Figura 3. Componentes de PETSc.

Aparte de algunos objetos auxiliares relativos a la gestión de las mallas, los componentes básicos a nivel de datos que incluye PETSc son los vectores y las matrices. Estos objetos son análogos a los vectores y matrices del álgebra lineal, y PETSc ofrece un gran número de operaciones que permiten realizar los cálculos más habituales, facilitando enormemente la labor del programador. Por encima de ellos tenemos la posibilidad de usar varias clases de solvers, tanto lineales como no lineales, diferentes precondicionadores y métodos de resolución de sistemas de ecuaciones entre otras posibilidades, todo ello en el entorno paralelo que el propio PETSc gestiona en función de las características del sistema en el cual se instala.

Las matrices en PETSc tienen una interfaz independiente de la implementación, lo que implica poder utilizar diferentes formatos de almacenamiento de matrices sin necesidad de modificar el código fuente. Esto permite por ejemplo especificar el formato de almacenamiento al ejecutar el programa, pudiendo así comparar las prestaciones de las distintas alternativas. Por defecto las matrices se almacenan en el formato disperso comprido por filas. Otras alternativas disponibles son el formato simétrico (almacenando unicamente la parte triangular superior) o el formato por bloques. Todos ellos son estructuras de datos paralelas, con distribución orientada a bloques de filas contiguas, y la comunicación necesaria para realizar las operaciones en paralelo es gestionada internamente por PETSc. En su uso básico, el programador solo ha de preocuparse de que cada proceso MPI se encargue de rellenar la parte que le ha sido asignada.

PETSc permite instalaciones en modo ‘debugger’, para permitir la depuración de errores, y en modo optimizado, con el cual podemos conseguir tiempos de cálculo sensiblemente inferiores, adecuado ya para la fase de producción de resultados.

4.1. Tareas computacionales

Las principales tareas computacionales requeridas en cada paso son las siguientes:

1. Actualizar las variables de estado respecto al paso de tiempo actual.
2. Computar la matriz Jacobiana y el vector de residuos de la ecuación no lineal para la iteración de Newton-Raphson.
3. Resolver el sistema lineal de ecuaciones.

Debido al esquema de discretización por diferencias finitas, los dos primeros paso requieren del uso de valores de celdas vecinas, con la necesidad de comunicación que ello conlleva.

Respecto al tercer paso, hay que señalar que la matriz de coeficientes del sistema lineal es dispersa, por lo que los métodos iterativos son los recomedables para la resolución del sistema.

4.2. Estrategia de paralelización

La paralelización permite el uso de varios procesadores (o cores) cooperando en la solución de un problema simple. Esto no sólo reduce el tiempo de computación sino que también incrementa la cantidad de memoria disponible permitiendo resolver problemas más grandes. La paralelización efectiva de COBRA-TF debe contemplar dos puntos importantes:

- Los tres pasos mencionados anteriormente deben ser paralelizados, no sólo la solución del sistema de ecuaciones.
- La memoria se debe manejar de forma apropiada, repartiendo las variables de estado entre los diferentes procesos (en el caso de paralelización con MPI).

La paralelización de COBRA-TF precisa de una estrategia híbrida, dónde la partición del cómputo se desarrolla en dos niveles:

- En el nivel superior se ha desarrollado una paralelización basada en la estrategía de descomposición de dominios con el paradigma de paso de mensajes (MPI).
- En el segundo nivel se prevé realizar en el futuro una paralelización de grano fino asociada a cada subdominio.

Este esquema permite mejor escalabilidad en algunas plataformas de computación, tales como clusters de multicores, o clusters con aceleradores en cada nodo. En este trabajo nos hemos centrado en el nivel superior únicamente.

4.3. Análisis del punto caliente

Para abordar la estrategia correcta a la hora de paralelizar el código COBRA-TF, en primer lugar se realiza un estudio del tiempo, y porcentaje respecto al total, que consume cada uno de los módulos y/o subrutinas que se ejecutaban en el proceso. Para ello se utiliza la herramienta Valgrind [7].

Analizando los resultados obtenidos con Valgrind se aprecia como la subrutina *outer* acumula el 87.75% del coste, que a su vez llama a *xschem*, quien ejecuta *gssolv*, lugar dónde se resuelve el sistema de ecuaciones propiamente dicho. Viendo que el resto de

llamadas se realizan un número muy alto de veces y con coste poco significativo, se toma este punto del código como el objetivo de la paralelización. Una vez deducido el punto adecuado para realizar la paralelización, se establece que es necesario eliminar todas las llamadas a las subrutinas que se ejecutan por debajo de *krylovsolv*, de forma que estos ya no serán necesarios en el cálculo ya que implican el uso de la librería SPARSKIT para la resolución de los sistemas de ecuaciones.

Con esta estrategía a la hora de elegir el punto caliente, se consigue realizar la paralelización de la parte del cálculo que absorbe la mayor cantidad de tiempo. Esto implica que tendremos se tendrán mayores beneficios a medida que aumente el número de veces que hay que resolver el sistema de ecuaciones y sobre todo si aumenta su dimensión, ya que la parte paralela saldrá muy beneficiada frente a realizar el mismo cálculo en secuencial.

4.4. Integración de PETSc

La parte fundamental de la paralelización de la resolución del sistema de ecuaciones es la sustitución de la librería SPARSKIT por PETSc como medio para resolver los sistemas de ecuaciones. Para ello se sustituyeron las llamadas a SPARSKIT por las correspondientes a PETSc analizando el lugar correcto para su inicialización, así como la creación y destrucción de todos los objetos (matrices y vectores) necesarios en el cálculo. Todo ello se tuvo que coordinar con las estructuras de datos ya existentes en el código FORTRAN.

Todo ello sin olvidar la instalación y optimización de PETSc para amoldarse al sistema en el cual se desarrolla la ejecución.

4.5. Adaptación de PETSc

Una vez resuelto el problema de sustituir SPARSKIT por PETSc, se procede a analizar las diferentes posibilidades que PETSc ofrece a la hora de gestionar las matrices, vectores y solucionadores. Para ello se realiza un estudio del patrón de elementos no nulos de la matriz, para poder realizar la reserva de memoria de forma adecuada y en el lugar adecuado. El hecho de reservar correctamente la memoria en los objetos que crea PETSc ofrece importantes ventajas en cuanto a eficiencia a la hora de resolver el sistema de ecuaciones. Además, debido a que el sistema se tiene que resolver muchas veces hasta alcanzar la convergencia, se estudió la posibilidad de reusar los objetos matriz, vector y solver para ganar velocidad, de forma que al mantener su estructura invariable, con todo esto se consigue un considerable aumento de velocidad de cálculo.

4.5. Optimización de PETSc

En la fase de optimización hay que averiguar que combinación de precondicionadores y métodos de resolución es la mejor opción para que PETSc realice el cálculo en la menor cantidad de tiempo posible, permitiendo obtener una significativa ganancia de velocidad. No todos los precondicionadores y métodos funcionan igual en el mismo caso, por lo que se realiza una prueba completa probando las diferentes combinaciones existentes en PETSc. Para ello se probaron los siguientes métodos y precondicionadores [8]:

- Precondicionadores: Block Jacobi (con ILU(0) como precondicionador local), Jacobi, SOR, Additive Schwarz (con ILU(0) como precondicionador local).
- Métodos iterativos de resolución de sistemas: GMRES, DGMRES, BiCGstab, BiCGstab(l).

4.5. Paralelización en subdominios axiales

Una vez paralelizado el sistema de ecuaciones, se procede a realizar el reparto paralelo de las celdas entre los procesos en su dimensión axial, esto es, cada proceso MPI tiene asignado un espacio continuo de niveles axiales. Habitualmente el número de niveles axiales es limitado, por lo que la escalabilidad del código paralelo está limitada en esta dimensión. La paralelización en el dominio radial será abordada en posteriores trabajos. Como ya se ha comentado, en muchas ocasiones se necesita información de niveles vecinos, por lo que cada proceso deberá tener espacio para estos niveles "fantasma" y además realizar de forma adecuada la comunicación de los datos implicados. Además siempre se cuenta con dos niveles extras, el inferior y el superior, que tienen que ser actualizados de forma correcta en cada iteración.

Esta etapa conlleva tres partes claramente diferenciadas:

- Reparto entre los procesos de las variables que simulan el comportamiento del combustible.
- Comunicación de los datos de dichas variables que están en la frontera en el reparto entre procesos y que por lo tanto son necesarios en los procesos vecinos.
- Reducción de los cálculos que implican máximos, mínimos y sumatorios al estar la carga repartida entre los procesos, de forma que todos posean la información adecuada a cada iteración.

5. RESULTADOS DURANTE LA OPTIMIZACIÓN

Para el estudio se resuelve de forma completa con los métodos bcgs, bcgsl, gmres y dgmres variando con los precondicionadores asm, bjacobi, jacobi y sor. Señalar que los métodos gmres y dgmres divergen. En el caso del programa original, el método de resolución utilizado el BiCGstab y el precondicionador ILUT (Incomplete LU factorization with dual truncation strategy). El precondicionador ILUT no está disponible en PETSc, dónde sí que está ILU(0), que suele ofrecer peores resultados.

En la tablas 4, 5 y 6 se puede observar respectivamente el tiempo total de ejecución y el speedup obtenidos con las diferentes combinaciones. Todos los tiempos se obtuvieron en un HP Proliant con 2 procesadores AMD Opteron 6272 con 16 cores a 2.2 GHz con 2M de caché y 96 Gb de memoria RAM. El Sistema Operativo utilizado es CentOS 6.3 y la versión del PETSc es la 3.3-p6. Se utilizó el OpenMPI realizando un estudio de la afinidad entre procesos y unidades de proceso para obtener la mejor opción en cuanto a la ganancia de prestaciones, siendo la opción que mejores resultados proporcionaba la reserva de cores con la opción --cpus-per-proc 4. Los tiempos obtenidos en este ejemplo se corresponden con una simulación de 1 segundo.

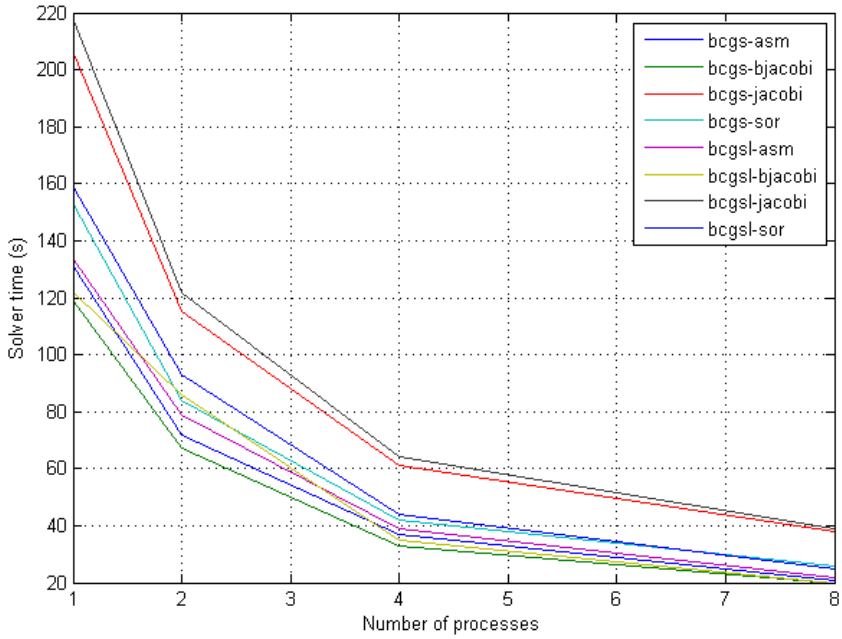

Figura 4. Tiempo empleado por el solver.

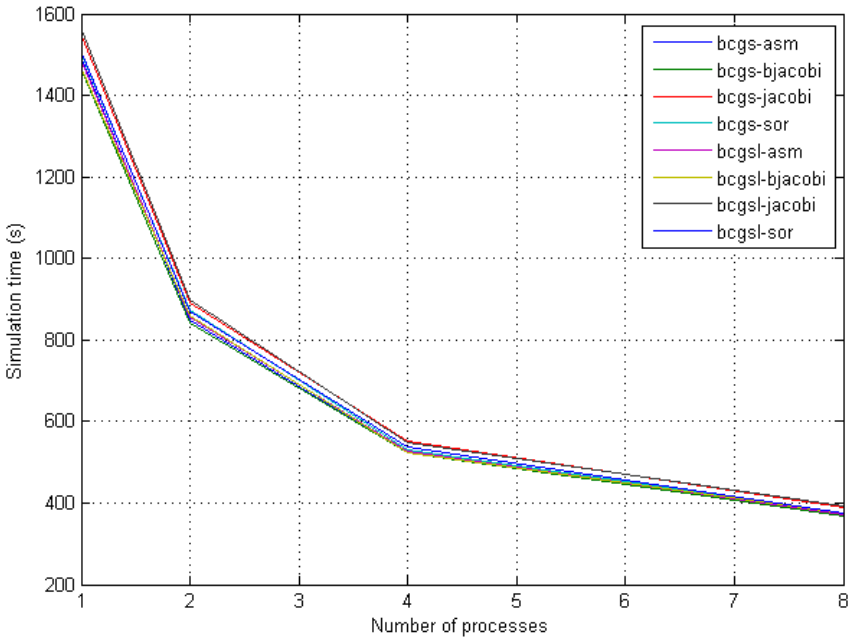

Figura 5. Tiempo total de simulación para cada solver.

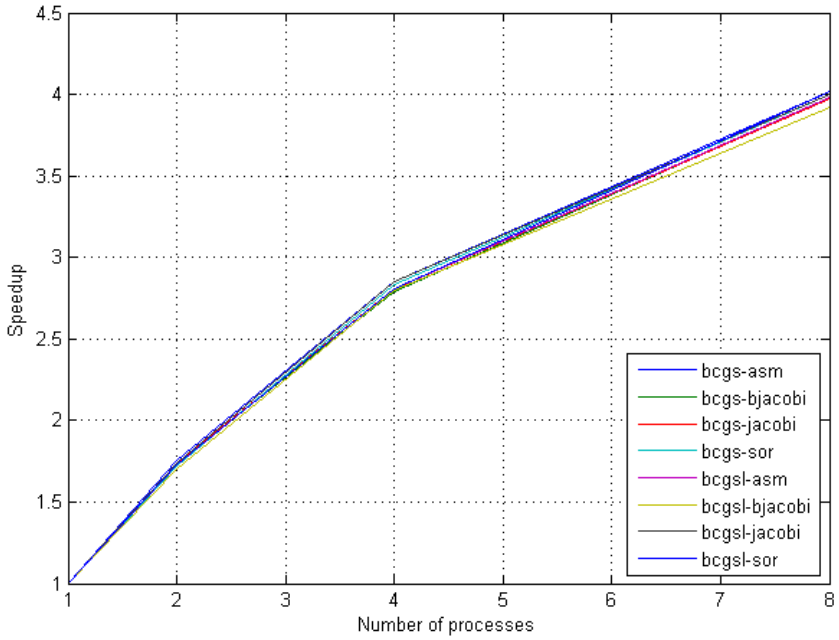

Figura 6. Speedup obtenido con cada solver.

En los resultados se puede apreciar todas las combinaciones ofrecen un comportamiento similar no apreciándose ganancia significativa en ninguna de ellas.

6. RESULTADOS FINALES

Por último, el estudio final consiste en la resolución de un estacionario pequeño estacionario de 4 segundos de duración con la versión paralela del código y el modelo de subcanal del elemento combustible previamente presentado. Para esta simulación utiliza el método bcgs con el bjacobi como precondicionador.

En las figuras 7 y 8 se muestra el tiempo total de simulación y el speedup alcanzado en función del número de procesos MPI utilizados.

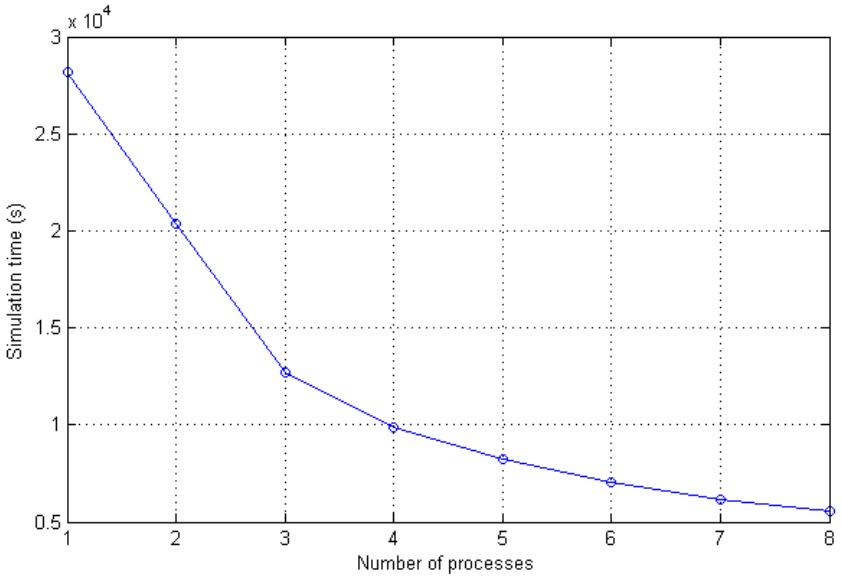

Figura 7. Tiempo total de simulación en función del número de procesos.

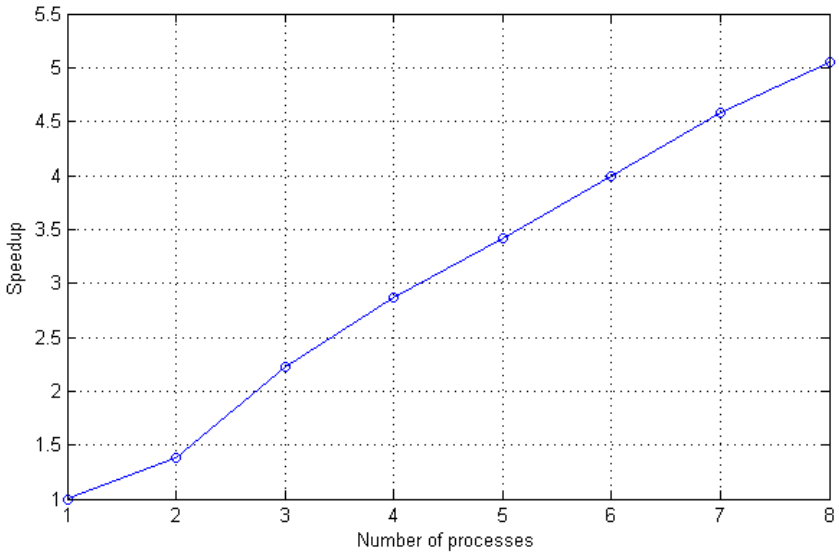

Figura 8. Speedup obtenido en función del número de procesos.

Como puede observarse se obtiene un speedup ligeramente superior con 8 procesos, es decir, utilizando 8 cores de un cluster podemos ejecutar un caso más de 5 veces más rápido con la versión paralela de COBRA-TF que con la secuencial original.

7. CONCLUSIONES

Este proyecto ha sido desarrollado con el objetivo de aplicar las más modernas técnicas de ingeniería de la informática, como son los paradigmas de la computación en paralelo, a los códigos de simulación termohidráulica existentes, como el código de subcanal COBRA-TF. En este trabajo se presentan los resultados conseguidos con el cumpliento del primer objetivo marcado.

En esta primera etapa se ha desarrollado un código en entorno MPI (message-passing code), capaz de ejecutarse en un cluster de computadores, consiguiendo una reducción significativa del tiempo de simulación. Futuros desarrollos incorporarán un segundo nivel de paralelización dentro de los diferentes niveles axiales con el uso de técnicas de multi-threading.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente esponsorizado por la Universitat Politècnica de València bajo los proyectos COBRA_PAR PAID-05-11-2810 y OpenNUC PAID-05-12.

REFERENCIAS

1) Diana Cuervo, Improving the Computation Efficiency of COBRA-TF for LWR Safety Analysis of Large Problems, PHYSOR, Chicago, USA, 2004.

2) Diana Cuervo, *Implementation and performance of Krylov Methods for the solution of Two-Fluid Hydrodinamics Equations in the COBRA-TF Code*, M&C, Avignon, France, 2005.

3) Satish Balay, Jed Brown, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, Hong Zhang, PETSc Web page, "http://www.mcs.anl.gov/petsc", 2011.

4) Satish Balay, Jed Brown, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, Hong Zhang, PETSc Users Manual, ANL-95/11 - Revision 3.3, Argonne National Laboratory, 2012.

5) Satish Balay, William D. Gropp, Dinesh Kaushik, Lois Curfman McInnes, Barry F. Smith, Efficient Management of Parallelism in Object Oriented Numerical Software Libraries, E. Arge and A. M. Bruaset and H. P. Langtangen, 1997.

6) Tjalling J. Ypma, *Historical development of the Newton-Raphson method*, SIAM Review 37 (4), 531-551, 1995.

7) Cerion Armour-Brown, Jeremy Fitzhardinge, Tom Hughes, Nicholas Nethercote, Paul Mackerras, Dirk Mueller, Julian Seward, Bart Van Assche, Robert Walsh, Josef Weidendorfer, valgrind.org, 2012.

8) Y. Saad, *Iterative Methods for Sparse Linear Systems, 2nd edition*, Society for Industrial and Applied Mathematics, 2003.