Algoritmos Paralelos para la Resolución de Ecuaciones Diferenciales Ordinarias Mediante OpenMP

E. Arias[1], V.Hernández[2], J. J. Ibáñez[2], J. Peralta[3]

Resumen— **En los últimos años está adquiriendo un gran auge el estudio de los métodos numéricos para la resolución Ecuaciones Diferenciales Ordinarias (ODE's). Muchos de los métodos numéricos existentes se basan en la aproximación de un modelo continuo mediante un modelo discreto y el cálculo de una solución aproximada en un conjunto finito de puntos. En [1] se presenta una nueva aproximación al cálculo de ODE's donde la principal aportación pasa por permitir una solución del problema independientemente de que el Jacobiano sea o no invertible. En el presente trabajo se presenta un nuevo algoritmo basado en [1] que permite la resolución de ODE's. Además, se ha llevado a cabo una implementación paralela sobre arquitecturas de memoria compartida de dicho algoritmo. Tanto el algoritmo secuencial como el algoritmo paralelo desarrollado se han implementado utilizando librerías estándar tanto en el cómputo como en la comunicación en aras de obtener portabilidad, robustez y eficiencia.**

Palabras clave— **Ecuaciones Diferenciales Ordinarias, Ecuaciones Diferenciales Algebraicas, Librerías estándar, Computación de Altas Prestaciones, OpenMP.**

I. Introducción

En los últimos años está adquiriendo un gran auge el estudio de los métodos numéricos para la resolución de Ecuaciones Diferenciales Ordinarias (ODE's). En [1] se presenta una nueva aproximación al cálculo de ODE's donde la principal aportación pasa por permitir una solución del problema independientemente de que el Jacobiano sea o no invertible. En el presente trabajo se presenta una implementación paralela sobre memoria compartida basada en [1] que permite la resolución de ODE's.

Considérese el siguiente problema de valores iniciales descrito por la ecuación diferencial ordinaria

$$\dot{x} = f(x,t), \tag{1}$$
$$x(t_0) = x_0 \tag{2}$$

donde $x \in R^N$, $t \in (t_0, t_f]$, f es una función continua y Lipschitziana $f \in C^2$.

La ecuación (1) es, en general, no lineal y además, sólo puede resolverse analíticamente en casos muy concretos. Sin embargo, para $t \in I = [t_0, t_f]$, se puede obtener una solución analítica aproximada, de la siguiente manera. El intervalo I se divide en n subintervalos, $I = (t_0, t_1] \bigcup \ldots \bigcup (t_{n-1}, t_n]$, y en cada subintervalo, $I_i = (t_i, t_{i+1}], \quad i = 0, 1, \ldots, n-1$, la ecuación (1) se aproxima por

$$y' = g(y,y) \equiv F_i + J_i(y - y_i) + T_i(t - t_i), \;\; t \in (t_i, t_{i+1}], \tag{3}$$

$$y(t_i) = y_i, \tag{4}$$

donde

$$J_i = \frac{\partial f}{\partial x}(y(t_i), t_i), \quad T_i = \frac{\partial f}{\partial t}(y(t_i), t_i), \quad F_i = f(y(t_i), t_i),$$

$y(t_0) = x_0$, y J_i es una matriz Jacobiana y T_i es un vector gradiente. El lado derecho de la ecuación (3) es el polinomio de Taylor de primer grado de $f(x,t)$ alrededor de $(y(t_i), t_i)$ en I_i. Si f es de clase $\mathbf{C}^2$, la aproximación lineal a f tiene un orden de exactitud del tipo $O((t_{i+1} - t_i)^2)$.

Como la ecuación (3) es lineal, la solución analítica en I_i viene dada por

$$y(t) = y_i + \int_{t_i}^{t} e^{(J_i(t-\tau))}[T_i(\tau - t_i) + F_i]d\tau, \quad i = 0, 1, \ldots, n-1 \tag{5}$$

La ecuación (5) proporciona una solución aproximada, analítica a trozos, del problema de valores iniciales planteado en las ecuaciones (1) y (2), donde y_i es la solución de las ecuaciones (3) y (4), en el intervalo I_{i-1}, evaluada en el punto t_i.

Este tipo de método es conocido como **método de linealización a trozos**.

En [2] se presenta una metodología basada en la integración analítica de (5) que da lugar a la siguiente expresión, siempre y cuando J_i sea invertible

$$y(t) = y_i - J_i^{-1}[F_i + T_i(t - t_i)] - J_i^{-2}T_i + e^{J_i(t-t_i)}[J_i^{-1}F_i + J_i^{-2}T_i], \quad i = \tag{6}$$

II. Resolución de ODE's mediante la aproximación del Jacobiano libre de inversión

En el caso de que el Jacobiano no sea invertible, la ecuación (6) no tiene solución. La nueva aproximación resuelve la ecuación (5) aún cuando el Jacobiano J_i sea singular, esto es, no sea invertible. Esto es posible realizando el cálculo de la exponencial de la matriz J_i mediante el método de los aproximantes de Padé [3].

[1] Departamento de Informática. Universidad De Castilla -La Mancha. Avda. España s/n. 02071-Albacete. España. earias@info-ab.uclm.es.

[2]Departamento de Sistemas Informáticos y Computación. Universidad Politénica de Valencia. Camino de Vera s/n. 46071-Valencia. España. {iblanque,vhernand,jjibanez}@dsic.upv.es

[3] Escuela Politécnica Superior de Albacete. Universidad De Castilla -La Mancha. Avda. España s/n. 02071-Albacete. España.

A continuación se describe la nueva metodología para la resolución de ODE's mediante la aproximación del Jacobiano libre de inversión.

Sea el siguiente cambio de variable

$$s = \tau - t_i,$$

y tomando

$$\Theta = t - t_i,$$

la integral en (5) puede expresarse como

$$\int_{t_i}^{t} e^{J_i(t-\tau)}[T_i(\tau - t_i) + F_i]d\tau = \quad (7)$$

$$= \left[\int_0^{\Theta} e^{J_i(\Theta - s)} s ds\right] T_i + \left[\int_0^{\Theta} e^{J_i(\Theta - s)} ds\right] F_i. \quad (8)$$

Las dos integrales que forman el lado derecho de la expresión pueden ser calculadas de la manera que a continuación se explica. Se considera la matriz triangular superior a bloques C [4]

$$C = \begin{bmatrix} J_i & I_N & 0_N \\ 0_N & 0_N & I_N \\ 0_N & 0_N & 0_N \end{bmatrix}, \quad (9)$$

donde $J_i \in R^{N\times N}$ es la matriz Jacobiana, 0_N y I_N son respectivamente, la matriz nula y la matriz identidad de orden N. Sea también la exponencial de la matriz $C\Theta$

$$e^{C\Theta} = \begin{bmatrix} F_1(\Theta) & G_1(\Theta) & H(\Theta) \\ 0_N & F_2(\Theta) & G_2(\Theta) \\ 0_N & 0_N & F_3(\Theta) \end{bmatrix}, \quad (10)$$

siendo $F_1(\Theta)$, $F_2(\Theta)$, $F_3(\Theta)$, $G_1(\Theta)$, $G_2(\Theta)$ y $H(\Theta)$, matrices cuadradas de orden N.

Puesto que la exponencial anterior cumple

$$\frac{de^{C\Theta}}{d\Theta} = Ce^{C\Theta},\ e^{C\Theta}\Big|_{\Theta=0} = I_{3N},$$

por consiguiente

$$\begin{pmatrix} \frac{dF_1(\Theta)}{d\Theta} & \frac{dG_1(\Theta)}{d\Theta} & \frac{dH(\Theta)}{d\Theta} \\ 0_N & \frac{dF_2(\Theta)}{d\Theta} & \frac{dG_2(\Theta)}{d\Theta} \\ 0_N & 0_N & \frac{dF_3(\Theta)}{d\Theta} \end{pmatrix} =$$

$$= \begin{pmatrix} J_iF_1(\Theta) & J_iG_1(\Theta) + F_2(\Theta) & J_iH(\Theta) + G_2(\Theta) \\ 0_N & 0_N & F_3(\Theta) \\ 0_N & 0_N & 0_N \end{pmatrix},$$

y, como resultado, se obtienen los siguientes problemas de valores iniciales

$$\frac{dF_1(\Theta)}{d\Theta} = J_iF_1(\Theta);\quad F_1(0) = I_N,$$

$$\frac{dF_2(\Theta)}{d\Theta} = 0_N;\quad F_2(0) = I_N,$$

$$\frac{dF_3(\Theta)}{d\Theta} = 0_N;\quad F_3(0) = I_N,$$

$$\frac{dG_1(\Theta)}{d\Theta} = J_iG_1(\Theta) + F_2(\Theta);\quad G_1(0) = 0_N,$$

$$\frac{dG_2(\Theta)}{d\Theta} = F_3(\Theta);\quad G_2(0) = 0_N,$$

$$\frac{dH(\Theta)}{d\Theta} = J_iH(\Theta) + G_2(\Theta);\quad H(0) = 0_N.$$

Las soluciones a estos problemas vienen dadas por

$$F_1(\Theta) = e^{J_i\Theta},\ F_2(\Theta) = F_3(\Theta) = I_N,$$
$$G_2(\Theta) = \Theta I_n,$$

$$G_1(\Theta) = \int_0^{\Theta} e^{J_i(\Theta - s)} ds,\ H(\Theta) = \int_0^{\Theta} e^{J_i(\Theta - s)} s ds. \quad (11)$$

Notar que las integrales involucradas en (8) vienen dadas por la expresión (11).

Recapitulando, una vez que la matriz Jacobiana J_i y los vectores T_i y F_i han sido computados la expresión de la ecuación (5) puede ser obtenida mediante

$$y(t) = y_i + H(\Theta)T_i + G_1(\Theta)F_i, \quad (12)$$

donde $H(\Theta)$ y $G_1(\Theta)$ son los bloques $(1,3)$ y $(1,2)$ de la matriz exponencial dada por (10). Notar nuevamente que en este caso la condición de invertibilidad de la matriz Jacobiano desaparece.

III. Algoritmos Implementados

Tanto Los algoritmos secuenciales como paralelos del presente trabajo han sido implementados teniendo en cuenta las siguientes caraterísticas:

- **Portabilidad**: Para conseguir algoritmos portables se han utilizado librerías estándar tanto de algebra lineal numérica en la parte computacional (BLAS [5], LAPACK [6], PBLAS [7]) como en la parte de comunicación y sincronización (OpenMP [8], [9]).
- **Eficiencia**: En aras de obtener buenas prestaciones se han utilizado algoritmos orientados a bloques, explotando de esta manera la jerarquía de memoria de los computadores actuales.

En la implementación secuencial del algoritmo propuesto anteriormente se han utilizado las siguientes rutinas:

- Rutinas de BLAS
  - _COPY (BLAS I): Copia de un vector.
  - _SCAL (BLAS I): Escalado de un vector.
  - _AXPY (BLAS I): Escalado de un vector x mas otro vector y.
  - _GEMV (BLAS II): Producto matriz-vector.
  - _GEMM (BLAS III): Producto de matrices.
- Rutinas de LAPACK:
  - _GESVD: Descomposición en valores singulares.

El objetivo de este trabajo ha consistido en la implementación de algoritmos paralelos sobre arquitecturas de memoria compartida utilizando OpenMP. OpenMP es un entorno para la programación de arquitecturas de memoria compartida que se han convertido en tan sólo unos años en un estándar para paralelizar aplicaciones. Debido a los buenos resultados obtenidos con esta herramienta, ésta es utilizada por empresas tales como Sun, Hewlett-Packard, Intel, IBM, Compaq y SGI, entre otros. OpenMP trabaja, por tanto, con hilos. Los hilos han apare-

cido como una buena herramienta en la que apoyarse para aprovechar y acelerar aún más los sistemas multiprocesador (memoria compartida). Las características de los hilos favorecen el aumento del rendimiento de los procesadores, disminuyendo la sobrecarga debida al cambio de contexto entre las tareas que ejecuta un procesador. Además son aplicables a una gran diversidad de problemas, y se integran perfectamente con el hardware paralelo, aumentando su rendimiento.

El problema básico en la escritura de un programa concurrente es identificar qué actividades pueden realizarse concurrentemente. Además la programación concurrente es mucho más difícil que la programación secuencial clásica por la dificultad de asegurar que el programa concurrente es correcto.

Los programas concurrentes a diferencia de los programas secuenciales tienen una serie de problemas muy particulares derivados de las características de la concurrencia, de entre ellos destacan los siguientes:

- **Violación de la exclusión mutua**: En ocasiones ciertas acciones que se realizan en un programa concurrente no proporcionan los resultados deseados. Esto se debe a que existe una parte del programa donde se realizan dichas acciones que constituye una **región crítica**, es decir, es una parte del programa en la que se debe garantizar que si un proceso accede a la misma, ningún otro podrá acceder. Se necesita pues garantizar la **exclusión mutua**.
- **Bloqueo mutuo o Deadlock**: Un proceso se encuentra en estado de **deadlock** si está esperando por un suceso que no ocurrirá nunca. Se puede producir en la comunicación de procesos y más frecuentemente en la gestión de recursos.
- **Retraso indefinido o starvation**: Un proceso se encuentra en **starvation** si es retrasado indefinidamente esperando un suceso que puede no ocurrir nunca.

Por lo anteriormente expuesto, el *handicap* de trabajar en un entorno multiprocesador, es precisamente la programación de dicho entorno. Es aquí donde estas librerías son muy útiles ya que las modificaciones que hay que llevar a cabo en el código secuencial son mínimas, para hacer que éste funcione en paralelo.

En OpenMP, como ya hemos comentado, partiendo del código secuencial, y añadiendo unas pocas directivas y una pequeña modificación del código, conseguimos paralelizar dicho código. Toda directiva debe comenzar con la palabra clave **¡$omp** para que el compilador las reconozca como directivas de paralelizacion. Las directivas más utilizadas son:

- **parallel do:** Se usa justo antes de un bucle y se acaba el bucle paralelizado con **end parallel do**. Con esta directiva se indica que el bucle que viene a continuación, va a ser dividido entre tantos hilos o procesadores como previamente se haya indicado en el código mediante la rutina *set_num_threads(x)*, donde x es el número de procesadores que se quieren utilizar.
- **shared:** Indica que cualquier hilo podrá acceder y modificar dicha variable en cualquier momento, es decir, es una variable compartida.
- **private:** Indica que cada hilo tendrá una copia de dicha variable para su propio uso. Un ejemplo claro son los contadores de las iteraciones.
- **reduction:** Esta directiva se utiliza para operaciones de reducción tales como sumas cumulativas, mayor, menor, etc. En el trozo de código siguiente se muestra un ejemplo de uso de la directiva **reduction**

```
<$omp parallel do reduction(+:sum)
do I=1, N
   sum=sum+a(i)
endo
<$omp end parallel do
```

- **parallel:** Esta directiva, al contrario que **parallel do**, no divide el trabajo sino que lo multiplica. Es decir, si se dispone de n hilos, al utilizar la directiva **parallel** cada hilo ejecutará lo que se encuentre en su entorno. En el siguiente ejemplo se imprimiría n veces *"Hola Mundo"*.

```
<$omp parallel
      print *,"Hola Mundo"
<$omp end parallel
```

- **single:** Dentro de una zona paralelizada por la directiva **parallel**, puede existir una parte de código que no es aconsejable que la ejecuten varios hilos. Por tanto, con la directiva **single** se indica que esa parte del código sea llevada a cabo por un sólo hilo de los creados.
- **critical:** Indica, también dentro de un **parallel**, que esa zona de código será ejecutada por todos los hilos, pero de uno en uno, nunca todos a la vez.

La manera más común y eficiente que hay de paralelizar un código secuencial con las directivas OpenMP, es atacando directamente a los bucles. Los bucles sencillos y anidados son los que más tiempo de ejecución consumen. A continuación se muestra la parelelización de un ejemplo simple. Considérese el siguiente trozo de código

```
do 30 i = 1,m
   dy(i) = dy(i) + da*dx(i)
30 continue
```

Utilizando OpenMP el trozo de código anterior queda de la siguiente manera

```
<$omp parallel do private(i) shared(da,dx,dy)
do 30 i = 1,m
   dy(i) = dy(i) + da*dx(i)
30 continue
```

Declarando la variable i como privada (*private*) si $m = 60$ y el número de hilos fuese 6, el hilo 1 trabajaría con los índices del 1 al 10 de i, el segundo con los índices del 11 al 20, y así sucesivamente. Las

variables *da*, *dx* y *dy* se declaran como compartidas. Este tipo de bucles aparece con profusión en las rutinas *daxpy*, *dcopy*, *dlaset*, etc.

Un ejemplo más complicado en el que se tienen dos bucles anidados es el siguiente

```
DO 60, J = 1, N
   IF( X( JX )!=ZERO )
     THEN
       TEMP = ALPHA*X( JX )
       DO 50, I = 1, M
          Y( I ) = Y( I ) + TEMP*A( I, J )
       50 CONTINUE
   END IF
   JX = JX + INCX
60 CONTINUE
```

A la hora de paralelizar este tipo de bucles, es posible adoptar dos soluciones. Una de ellas es paralelizar sólamente el bucle interno, como se muestra a continuación

```
do 60, j=1, n
  if (x(jx+(j-1)*incx).ne.zero)
    then
     temp = alpha*x(jx+(j-1)*incx)
     <$omp parallel do private(I)
     do 50, i=1,M
        y(i) = y(i) + temp*A(i,j)
50   continue
     <$omp end parallel do
  endif
60 continue
```

La otra solución pasa por una inversión de los bucles. Dicha solución se presenta a continuación y es la adoptada en la paralelización de la rutina *dgemv*.

```
<$omp parallel do private(J,I,TEMP)
      shared(y, alpha,x,ix,incx,n,m)
do 50, i=1,m
   do 60, j=1, n
      y(i)=y(i)+alpha*x(jx+(j-1)*incx)*A(i,j)
   60 continue
50 continue
<$omp end parallel do
```

Por último, se muestra un ejemplo de paralelización de un producto de matrices donde se tienen 3 bucles anidados que aparecen en la rutina *dgemm*:

```
<$omp parallel do private(J,I,L,Temp)
                shared(A,B,C,ALPHA,BETA)
DO 120, J = 1, N
    DO 110, I = 1, M
       TEMP = ZERO
       DO 100, L = 1, K
          TEMP = TEMP + A( L, I )*B( L, J )
100    CONTINUE
       IF( BETA==ZERO )
        THEN
         C( I, J ) = ALPHA*TEMP
        ELSE
         C( I, J ) = ALPHA*TEMP + BETA*C(I,J)
       END IF
110 CONTINUE
120 CONTINUE
```

En definitiva, se ha creado una librería alternativa basada en BLAS que implementa las rutinas de BLAS anteriormente citadas. El nombre de dichas rutinas es el mismo que el de BLAS pero con el prefijo OP.

IV. Resultados Experimentales

En esta sección se describen los resultados experimentales obtenidos. Las pruebas se han llevado a cabo en un sistema Compaq AlphaServer GS80 instalado en el servicio de supercomputación de la Universidad de Castilla-La Mancha (SSC- UCLM). Consta de 6 procesadores Alpha EV7. Cada procesador consta de caché en chip y fuera de él. Tanto una como otra proporcionan una muy baja latencia lo que redunda en un gran ancho de banda en el acceso a los datos. La memoria caché fuera de chip es de 4MB trabajando a 366MHz. La caché interna se divide en caché de instrucciones (I-Caché) y de datos (D-Caché), cada una de 64Kbytes respectivamente. La velocidad por procesador es de 731Mhz. En cuanto a la memoria principal, el sistema GS80 soporta hasta 64Gbytes. En la instalación existente en el SSC-UCLM se disponen de 4Gbytes de memoria RAM. El sistema operativo empleado en la plataforma es el Sistema operativo Compaq TRU64 Unix v5.1.

Las pruebas a realizar se han obtenido del banco de pruebas IVPtestset [10] y se pueden clasificar en dos tipos:

- **Pruebas secuenciales**: Se han realizado 4 pruebas de tests de reducido tamaño y los resultados obtenidos se han comparado con los resultados obtenidos por otros paquetes estánda (ODEPACK [11], RADAU5 [12], PSIDE [13], etc).
- **Pruebas paralelas**: Se ha llevado a cabo 1 prueba de test de tamaño más grande que permite mostrar las ventajas del algoritmo paralelo.

Los test secuenciales que se han llevado a cabo son:

- **CHEMAKZO** (Chemical Akzo Nobel Problem): Este problema es un ODE de 6 ecuaciones.
- **LSODE** (Chemical kinetics): Este problema aparece en el solver *lsode* de ODEPACK. Consta de 3 ecuaciones.
- **HIRES** (High Irradiance Response Problem): Este caso de estudio ha sido obtenido a partir de la colección IVPtestset [10]. El problema representa un sistema rígido de 8 ODE's no lineales.

Los resultados obtenidos se resumen en las tablas I, II y III para las pruebas **CHEMAKZO**, **LSODE** y **HIRES** respectivamente. En dichas tablas se presenta una comparativa de la solución proporcionada con la nueva aproximación y con respecto a otros solvers. En todos los casos se observa una precisión mínima de 10^{-5}

En cuanto a las pruebas del algoritmo paralelo

TABLA I

Tabla comparativa entre la solución obtenida mediante la nueva aproximación y el solver RADAU5 para el test **CHEMAKZO**

	RADAU5	Nueva Aproximación
y(1)	0.11616E+00	0.11616E+00
y(2)	0.11194E-02	0.11194E-02
y(3)	0.16213E+00	0.16213E+00
y(4)	0.33970E-02	0.33970E-02
y(5)	0.16462E+00	0.16462E+00
y(6)	0.19895E+00	0.19895E+00

TABLA II

Tabla comparativa entre la solución obtenida mediante la nueva aproximación y el solver lsode de ODEPACK para el test **LSODE**

	ODEPACK	Nueva Aproximación
y(1)	0.98517e+00	0.98517E+00
y(2)	0.33864E-04	0.33864E-04
y(3)	0.14794E-01	0.14794E-01

se ha tenido en cuenta el número de hilos como parámetro de entrada, pudiendo variar entre 1 y 6.

Las prestaciones del algoritmo paralelo desarrollado se van a evaluar en función a dos criterios:

- **Speed-up**: Se define como el ratio entre el tiempo utilizado para resolver el problema en una sola máquina y el tiempo requerido para resolver el mismo problema en un computador paralelo de p procesadores idénticos.
- **Eficiencia**: Se define como el ratio entre el *Speed-up* y el número de procesadores.

El test llevado a cabo para evaluar el algoritmo paralelo ha sido **MEDAKZO** (Medical Akzo problem). Este caso de prueba también ha sido obtenido de la colección IVPtestset [10], que estudia la penetración de los anticuerpos en un tejido infectado por un tumor. El problema representa un ODE de 400 ecuaciones.

MEDAKZO necesita un gran número de iteraciones para poder obtener la solución final. Puesto que las pruebas secuenciales ya ponen de manifiesto la viabilidad del nuevo método, se ha considerado un número menor de iteraciones para obtener los resultados. Con este número de iteraciones ya se percibe la mejora en el uso del paralelismo.

TABLA III

Tabla comparativa entre la solución obtenida mediante la nueva aproximación y el solver RADAU5 para el test **HIRES**

	ODEPACK	Nueva Aproximación
y(1)	0.73713E-3	0.73683E-03
y(2)	0.14424E-3	0.14419E-03
y(3)	0.58887E-4	0.58830E-04
y(4)	0.11757E-2	0.11751E-02
y(5)	0.23864E-2	0.23772E-02
y(6)	0.62390E-2	0.62099E-02
y(7)	0.28500E-2	0.28437E-02
y(8)	0.28500-2	0.28563E-02

Los tiempos de ejecución para el test **MEDAKZO** se observan en la tabla IV. En dicha tabla se puede observar una drástica reducción en e el tiempo de ejecución conforme se aumenta el número de procesadores. Esa diferencia es mucho más destacable al pasar de 1 a 2 procesadores.

TABLA IV

Tiempos de Ejecución para el test **MEDAKZO**

N. Procesadores	T. de Ejecución (sg)
1	55.70
2	32.51
3	24.40
4	21.86
5	17.96
6	17.03

En la tabla V se reflejan la ganancia de velocidad o *Speed-up* obtenida. En esta tabla, se aprecia con mayor claridad la ventaja de utilizar el paralelismo. Así pues, utilizando 2 procesadores se obtiene un *speed-up* cercano a 1.75 sobre el máximo teórico de 2, lo que indica un muy buen resultado. Ese buen resultado se empeora conforme aumenta el número de procesadores, aunque incluso hasta 4 procesadores el valor de *speed-up* es bastante bueno.

TABLA V

Speed-up para el test **MEDAKZO**

Número de Procesadores	Speed-up
2	1.712
3	2.28
4	2.55
5	3.10
6	3.27

Por último, en la tabla VI se muestra la eficiencia del algoritmo paralelo frente al secuencial. La cota máxima en el caso de la eficiencia es de 1 (100%). Por consiguiente, quiere decir que utilizando 2 procesadores se está muy cerca de dicha cota, lo que demuestra el buen comportamiento del algoritmo paralelo considerando 2 procesadores. Se observa también como para 3 procesadores se alcanza una eficiencia del 75% y para 4 procesadores del 64%, lo que reafirma lo anteriormente expuesto.

V. Conclusiones y Trabajo Futuro

En el presente trabajo se presenta un nuevo algoritmo para la resolución de ecuaciones diferenciales ordinarias presentado en [1] utilizando la librería OpenMP.

TABLA VI
EFICIENCIA PARA EL TEST **MEDAKZO**

Número de Procesadores	Eficiencia
2	0.86
3	0.76
4	0.64
5	0.62
6	0.55

Tanto la versión secuencial del algoritmo propuesto, como la implementación paralela del mismo, se han llevado a cabo utilizando *software* estándar en aras de obtener tres características deseables en toda implementación: Portabilidad, Robustez y Eficiencia.

Los resultados experimentales demuestran la viabilidad de la nueva metodología con respecto a los resultados obtenidos por otros paquetes estándar existentes y ampliamente aceptados.

La implementación paralela nos permite abordar problemas de mayor dimensión reduciendo el tiempo de ejecución. Los resultados experimentales nos muestran cómo para dos procesadores el tiempo de CPU empleado es casi la mitad. Esta circunstancia se refleja en los valores del *Speed-up* y Eficiencia.

A raiz del trabajo aquí desarrollado, se plantea el siguiente trabajo futuro:

- Extender la metodología propuesta a la resolución de ecuaciones diferenciales implícitas donde tanto las matrices de pesos como Jacobianas sean densas.
- Extender la metodología propuesta a implementaciones específicas en los casos en que las matrices de pesos y Jacobiana no sean densas.
- Incorporar la metodología propuesta al estándar definido en la librería SLICOT del proyecto NICONET [14], [15].

REFERENCIAS

[1] E. Arias, I. Blanquer, V. Hernández, J. J. Ibá nez, and P. Ruiz, "Non-singular jacobian piecewise linearization of ordinary differential equations (ode's)," in *2nd Niconet Workshop*, Ronquecourt–France, 1999, vol. 1.

[2] García López Carmen María,, *Métodos de Linealización para la Resolución Numérica de Ecuaciones Diferenciales*, Ph.D. thesis, Univ. de Málaga, Dept. de Lenguajes y Ciencias de la Computación, 1998.

[3] G.B. Moler and C.F. Van Loan, "19 dubious ways to compute the exponential of a matrix," *SIAM Rev.*, vol. 29, pp. 801–837, 1978.

[4] C.F. Van Loan, "Computing integrals involving the matrix exponential," *IEEE Trans. Automat. Control*, vol. AC-23, pp. 395–404, 1978.

[5] S. Hammarling J. Dongarra, J. Du Croz and Richard J. Hanson, "An extended set of fortran basic linear algebra subroutines," *ACM Trans. Math. Software*, 1988.

[6] E. Anderson *et al.*, *LAPACK Users' Guide*, SIAM, Philadelphia, PA, second edition, 1994.

[7] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, and D. Walker, "A proposal for a set of parallel basic linear algebra subprograms," Technical report ut-cs-95-292, Department of Computer Science, University of Tennessee, May 1995.

[8] L. Dagum, *OpenMP: A Proposed Industry Standard API for Shared Memory Programming*, OpenMP.org., 1997.

[9] Rohit Chandra et al., *Parallel Programming in OpenMP*, Morgan Kaufmann Publishers, 2001.

[10] Parallel Software for Implicit Differential Equations, "Test set for ivp solvers," Report Release 2.1, http://www.cwi.nl/cwi/projects/IVPtestset/., September 1999.

[11] A. C. Hindmarsh, "Brief description of odepack-a systematized collection of ode solvers," Report, www.netlib.org/odepack/doc.

[12] G. Wanner E. Hairer, "Solving ordinary differential equations ii. stiff and differential-algebraic problems.," Springer-Verlag, 1996.

[13] J.J.B. de Swart, W.M. Lioen, and W.A. van der Veen, "Pside," Report, http://www.cwi.nl/cwi/projects/PSIDE/, November 1998.

[14] Numerical Algorithms Group, "Implementation and documentation standards for the subroutine library in control and systems theory SLICOT," Publication NP2032, Numerical Algorithm Group, Eindhoven/Oxford, 1990.

[15] V. Hernández, I. Blanquer, E. Arias, and P. Ruiz, "Definition and implementation of a slicot standard interface and the associated matlab gateway for the solution of nonlinear control systems by using ode and dae packages factorizations and linear system solvers for matrices with toeplitz structure.," Slicot working note, NICONET Project, 2000, SLW2000-3.

[16] E. Arias, I. Blanquer, V. Hernández, and J. J. Ibá nez, "Nonsingular jacobian free piecewise linearization of state equation," in *4th Portuguesse Conference of Automatic Control, CONTROLO'2000*, Guimaraes –Portugal, 2000.

[17] V. Hernández, I. Blanquer, A. Vidal, and E. Arias, "Slicot: Una librería de software numérico, eficiente y fiable para problemas de control con interfaces para matlab," in *III Congreso de Usuarios de MATLAB, MATLAB'99*, Madrid – España, 1999.

[18] L. R. Petzold, "Dassl library documentation," Report, www.netlib.org/ode.

[19] P. N. Brown, A. C. Hindmarsh, and L. R. Petzold., "Daspk package," Tech. Rep., 1995.

[20] P. Kunkel, V. Mehrmann, W. Rath, and J. Weickert, "Gelda: A software package for the solution of general lineal differential algebraic equations," *SIAM Journal Scientific Computing*, vol. 8, pp. 115–138, 1997.

[21] G.H. Golub and C.F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, 1983.