Paralelización de un algoritmo de calibrado para el modelado de ecosistemas

Albert Agraz[1], Josep Lluís Lérida[1], Francesc Solsona[1], Mari Angels Colomer[2]

Resumen— **Este artículo explica el proceso de paralelización de una aplicación que permite realizar de forma automática el calibrado de modelos de ecosistemas diseñados con `P-Systems`.**

Primero realizamos una introducción a los `P-Systems` y su funcionamiento. Analizaremos la imposibilidad de determinar el valor de ciertos parámetros que no son observables en la naturaleza y la necesidad de determinar unos valores adecuados para estos parámetros.

Veremos como nos encontramos delante de un problema de explosión combinatoria y calcularemos el orden de complejidad del mismo. Diseñaremos una aplicación serie que explore todas los posibles valores asignables a estos parámetros y que encuentre cuál es la que proporciona resultados más próximos a la realidad.

Acabaremos realizando una aplicación paralela que reduce sustancialmente el tiempo de ejecución del proceso de calibrado. Finalmente, indicaremos los resultados obtenidos y el trabajo futuro a realizar.

Palabras clave— **Ecosistemas, paralelización, `P-Systems`, `P-Lingua`, calibrado, explosión combinatoria, colas SGE.**

I. Introducción

La simulación matemática de procesos naturales tiene especial interés para poder conocer sus mecanismos de funcionamiento y poder predecir comportamientos bajo diferentes escenarios plausibles.

La metodología de modelado ha evolucionado a medida que el conocimiento de la realidad ha aumentado así como las prestaciones de los sistemas informáticos existentes en la actualidad. La existencia de potentes centros de cálculo, accesibles desde Internet, ha provocado un cambio en el modo de trabajar, de abordar los problemas y en el tipo de herramientas de modelado que se utilizan.

En la actualidad debemos dejar de asociar el concepto de modelado a la existencia de expresiones matemáticas que, de manera más o menos hábil, relacionan unos *inputs* con unos *outputs*. Durante muchos años se han utilizado modelos basados en las ecuaciones de Lotka Volterra [1] con muy buenos resultados, pero con limitaciones importantes como el número de procesos a modelar. En la actualidad se ha producido una explosión de modelos computacionales entre los que destacamos los modelos de viabilidad y los multiagentes. La última generación de modelos son conceptualmente más sencillos de entender pero más difíciles de describir, ya que no se pueden expresar mediante formulas analíticas.

[1]Dept. d'Informàtica i Enginyeria Industrial, Univ. de Lleida, email: albert.agraz@udl.cat, jlerida,francesc@diei.udl.es

[2]Dept. de Matemàtica, Univ. de Lleida, colomer@matematica.udl.cat

Entre los modelos computacionales de última generación están los `P-Systems` descritos por George Păun [2][3]. Son modelos bioinspirados basados en el funcionamiento de las células. Las células son organismos diminutos con unas zonas diferenciadas en su interior, que contienen una serie de orgánulos que operan simultáneamente de manera aleatoria, comunicándose e intercambiando materiales con el medio.

Los componentes básicos de un `P-System` son: la estructura de membranas, el alfabeto de trabajo y las reglas de evolución. Los componentes utilizados para modelar un problema concreto dependen de la estrategia seguida por el modelador; procurando siempre minimizar el coste computacional, que suele estar relacionado con el número de reglas. Existen distintas variantes de `P-Systems`, en el presente trabajo nos centraremos en la variante que se utiliza para la modelado de dinámicas poblacionales, denominados *Population Dynamic P-Systems models* (PDP).

Estos `P-Systems` han sido utilizados con resultados muy satisfactorios para el estudio del quebrantahuesos en los Pirineos catalanes [4][5][6][7], del mejillón cebra en el embalse de Ribarroja (Aragón), y otros como el rebeco, el acebo o el tritón pirenaico [8], el cual utilizaremos como base en la experimentación del presente trabajo.

Los parámetros que utiliza el modelo para un caso concreto son siempre los mismos independientemente del camino seguido para la obtención del *output*. Los parámetros son las medidas de campo realizadas por los expertos a lo largo del tiempo. El valor de algunos de estos parámetros no es puntual, sino que son intervalos, el experto es capaz de acotarlo pero no de precisar un único valor. Por lo tanto previa a la aplicación de los modelos para predecir futuros comportamientos, es necesario calibrar el modelo. Esto significa, buscar el valor de los parámetros que mejor ajusta los resultados del modelo a la realidad observada.

Considerando que el número de parámetros no precisados puede ser importante y estos además pueden interaccionar entre sí, nos encontramos ante un problema de exploración combinatoria, con un coste computacional importante. No obstante el correcto ajuste de estos modelos es de vital importancia, pues puede significar la diferencia entre un modelo válido y uno erróneo, es decir, entre obtener valores veraces o predicciones equivocadas. Así pues, se hace imprescindible el diseño de una herramienta de calibrado que permita el ajuste automático y eficiente de los parámetros del modelo PDP.

En este proyecto utilizaremos `P-Lingua` como motor de simulación para modelos PDP, desarrollado

por el Grupo de Investigación en Computación Natural [9] de la Universidad de Sevilla y publicado bajo la licencia `GNU GPL`. En trabajos previos [10][11] se ha tratado de paralelizar y optimizar el motor de simulación, pero nunca se ha abordado el problema del calibrado presentado en el presente trabajo.

II. Conceptos Previos

En esta sección describiremos los componentes principales de un `P-System`, los *inputs* necesarios para la modelado de un ecosistema y evaluaremos el orden de magnitud del proceso de calibración.

A. `P-Systems`

Un PDP está formado por un conjunto de entornos que, en el caso de los ecosistemas, suele estar asociado a distintos espacios geográficos. Dentro de cada entorno hay un `P-System` que no es más que un conjunto de **membranas** jerarquizadas con una determinada carga eléctrica (figura II-A). En el interior de estas membranas hay un conjunto de **objetos** que, en el caso de los ecosistemas, suelen estar asociados a los *inputs* del modelo (animales, comida,etc). Estos objetos evolucionan mediante las **reglas de evolución** pudiendo moverse entre membranas, salir del entorno, disolverse o generar nuevos objetos.

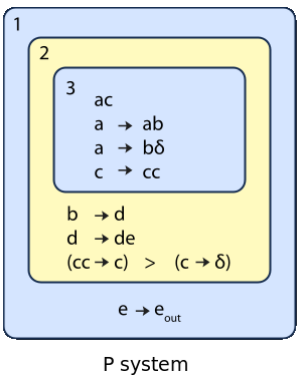

Fig. 1. Esquema de un P-Systema.

En los modelos PDP las reglas de evolución que pueden aplicarse en la región comprendida entre las membranas son del tipo:

$$r \equiv u^a[v^b]_i^\alpha \xrightarrow{f_r} u'^c[v'^d]_i^{\alpha'}$$

Si exterior a la membrana i que posee carga α, se encuentra el objeto u con multiplicidad a y en la membrana etiquetada con i el objeto v con multiplicidad b, con una probabilidad f_r se aplica la regla de manera que cambia la carga de la membrana de α a α' y los objetos u y v evolucionan a objetos u' y v' con multiplicidades c y d respectivamente.

Las reglas que se aplican entre entornos son de la forma:

$$r_e \equiv (x)_{e_j} \xrightarrow{p(x,j,j_1,\cdots,j_k)} (y_1)_{e_{j_1}} \cdots (y_k)_{e_{j_k}}$$

El objeto x pasa del entorno e_j a los entornos $e_{j_1} \cdots e_{j_k}$ en el paso entre entornos, pudiendo evolucionar a objetos $y_1 \cdots y_k$, respectivamente.

El punto de partida es la configuración inicial formada por la **estructura de membranas**, las **reglas de evolución** y el **alfabeto inicial**. En cada paso de computación se aplican todas las reglas posibles obteniéndose una configuración nueva, la secuencia de configuraciones forman una simulación.

El resultado de la simulación son los valores asignados al conjunto de objetos que se encuentran en cada membrana en el último paso de simulación.

B. Modelado de ecosistemas

Definiremos un modelo como el conjunto de reglas, membranas y alfabeto de trabajo (objetos) que definen el comportamiento de un sistema. Un modelo con valores iniciales en el alfabeto de trabajo, será denominado escenario. Un modelo dispone de tantos escenarios como combinaciones de valores posibles pueden tomar sus objetos.

El proceso de modelado de ecosistemas se sustenta en 4 fases principales, representadas en la figura II-B.

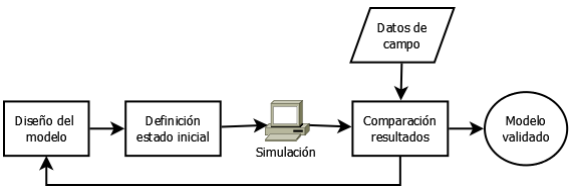

Fig. 2. Modelado de ecosistemas.

El primer paso, el **diseño del modelo**, consiste en identificar aquellas reglas de evolución que definen el comportamiento del sistema en estudio, así como la estructura de membranas que permita representar distintos estados durante la evolución, grupos de elementos, etc. Además se deben definir los objetos dentro de cada membrana. En nuestro caso, el conjunto de reglas, la estructura de las membranas y los objetos se definen mediante el lenguaje de programación `P-Lingua` [9] en un archivo fuente con extensión `.pli`.

Una vez definido el modelo, deberemos establecer el **escenario inicial**. Esto es definir un valor inicial para los objetos que hemos creado dentro de las distintas membranas para simular la evolución del sistema. La asignación de valores adecuados a estos objetos es esencial para la simulación veraz de la realidad.

Una vez definido el escenario inicial con el conjunto de reglas, estructura de membranas y valores iniciales de los objetos procedemos a la **simulación** de la evolución del sistema. Para ello utilizaremos el motor de simulación `P-LinguaCore` [12]. Esta aplicación recibe como parámetros de entrada el escenario inicial y el numero de pasos de simulación que se desean ejecutar sobre el ecosistema. El número de pasos es un parámetro definido por el ecólogo experto y está directamente relacionado con el número de años o meses que se desea evolucionar el ecosistema. Una vez finalizada la simulación obtenemos un fichero donde se registra el estado final del ecosistema y el proceso de evolución sufrido.

Dado que en el modelo definido se introduce la aleatoriedad propia de los sistemas naturales, los usuarios ecólogos tratan de simular el mismo escenario múltiples veces construyendo el resultado final como el promedio de los resultados obtenidos en las distintas simulaciones realizadas.

El modelo PDP analizado en este trabajo, modela el ecosistema del tritón pirenaico [8] mediante un único entorno con unas 20 membranas, 300 variables y 70 reglas de evolución. El coste temporal en la simulación de este modelo en una máquina convencional es aproximadamente del orden de 1 minuto y medio. Si tenemos en cuenta que se suelen hacer del orden de 30 a 100 simulaciones para normalizar los resultados nos encontramos con ejecuciones que pueden tardar de 45 minutos a las 2 horas y media.

En la última etapa del proceso de modelado debemos realizar la **validación**. Esto consiste en comprobar que los resultados del modelo se aproximan al comportamiento real del ecosistema. Para ello comparamos los resultados obtenidos a lo largo de los distintos pasos de simulación con observaciones reales del medio y calculamos el error producido. Daremos por buenos aquellos escenarios que más se aproximan a los valores observados reales.

C. Proceso de calibrado

Para obtener un buen ajuste del modelo es muy importante definir correctamente los valores de los parámetros de entrada. El valor de algunos de estos parámetros no es puntual, el experto es capaz de acotarlo pero no de precisar un valor exacto. Por lo tanto, es necesario poder ajustar con la mayor precisión posible el valor de estos parámetros. Para ello el usuario ecólogo debe generar múltiples escenarios con distintos valores para los parámetros de entrada, simular cada escenario varias veces para normalizar los resultados y comparar los resultados de cada escenario con los valores reales observados con el fin de obtener el escenario que mejor se acerca al comportamiento real.

En ocasiones el número de parámetros que el usuario ecólogo no puede precisar puede ser muy grande y además estos suelen interaccionar entre sí, por lo que nos encontramos ante un problema combinacional con un coste computacional importante que en ocasiones es inabordable. Por ejemplo, suponiendo que disponemos de 3 parámetros que pueden tomar 5 posibles valores, podemos generar hasta 5^3 posibles escenarios. En el caso de utilizar solamente 30 simulaciones para normalizar los resultados y suponiendo 1 minuto y medio para cada simulación, requeriríamos ($5^3 * 30 * 1,5 \frac{min}{sim} = 5625 min.$) aproximadamente 4 días para ajustar estos parámetros a los valores que mejores resultados proporcionan.

Dado el coste computacional tan elevado que puede supone el calibrado, se hace necesario implementar soluciones paralelas que permitan calibrar los modelos PDP para muchas variables en tiempos mucho más razonables para los usuario ecólogos. En el presente trabajo hemos implementado una solución serie con el fin de analizar el coste temporal del calibrado de un modelo PDP real usando un sistema de cómputo convencional. A continuación proponemos una solución paralela que, aún explorando todos los posibles escenarios optimiza al máximo el uso de los recursos del sistema minimizando enormemente el tiempo de obtención de la soluciones y permitiendo escalar el problema aumentando el número de parámetros a calibrar.

III. Calibración Serie

El objetivo de la aplicación serie es calibrar el modelo simulando todos los posibles escenarios un número de veces (N) determinado por el usuario ecólogo, promediando los resultados de cada uno de ellos y determinando cuál de ellos se ajusta mejor a la realidad.

Para llevar a cabo esta operación la aplicación ejecuta el procedimiento descrito en el Algoritmo 1. Como se puede ver, el algoritmo recibe como parámetros: el archivo con extensión `.pli` donde se definen la estructura de membranas y el conjunto de reglas de evolución, el archivo con extensión `.var` donde se define un valor inicial para aquellos parámetros que se van a mantener fijos en todos los escenarios y, finalmente, un archivo con extensión `.cal` donde se especifican los parámetros que deseamos calibrar y el rango de valores que pueden tomar estos parámetros, tal como se muestra a continuación:

```
q{1}=0.50:0.70:0.05
```

Donde `q{1}` es la variable a calibrar, seguida de el valor mínimo que puede tomar, el máximo y, por último, el incremento en que se desea recorrer el rango.

Algorithm 1 Calibración Serie

Require: ficheros: pli, var y cal

1: *lst_escenarios* := generar_lst_escenarios(*cal*);
2: **for** *Id_escenario* in *lst_escenarios* **do**
3: *escenario* := generar_escenario(*Id_escenario*,*pli*,*var*);
4: **for** i in N **do**
5: *resultados*[i] := simular(*escenario*);
6: **end for**
7: *resultado* := promediar(*resultados*);
8: *error* := distancia_euclidea(*resultado*,*real*);
9: **if** $error < error_{min}$ **then**
10: $error_{min} := error$;
11: $escenario_{min} := escenario$;
12: **end if**
13: **end for**
14: **return** *escenario*

Una vez se dispone de los *inputs*, el algoritmo genera mediante la función `generar_lst_escenarios(cal)`, una lista ordenada con los identificadores del conjunto de escenarios posibles a simular (linea 1). Para identificar cada escenario se utiliza una lista de longitud p, donde

p representa el número de parámetros a calibrar definidos por el archivo `.cal`.

Supongamos que la tabla I representa los rangos de valores definidos por el archivo `.cal`. Cada escenario se forma como una combinación distinta de los valores de cada parámetro. Para identificar los escenarios utilizaremos una lista de longitud 5, donde cada valor i de la lista representa el índice (columna) en el rango de posibles valores del valor escogido para ese escenario. Por ejemplo, el escenario `m{3,1}=0.05, m{4,1}=0.05, m{5,1}=0.05, p{1}=0.55, p{2}=0.55` tendrá como identificador el vector `(0,0,0,0,0)`, mientras que el escenario `m{3,1}=0.08, m{4,1}=0.07, m{5,1}=0.07, p{1}=0.55, p{2}=0.10` tendrá por identificador el vector `(3,1,2,0,5)`.

La ventaja principal de esta convención es la posibilidad de referirnos a un escenario sin tener que indicar los objetos y sus valores.

TABLA I
TABLA DE CALIBRADO

Nombre	Valores					
m{3,1}	0.05	0.06	0.07	0.08	0.09	0.10
m{4,1}	0.05	0.07	0.09	0.11	0.13	
m{5,1}	0.05	0.06	0.07	0.08	0.09	0.10
p{1}	0.55	0.60	0.65			
p{2}	0.05	0.06	0.07	0.08	0.09	0.10

Una vez generada la lista de escenarios el algoritmo explora la lista para procesar cada uno de ellos (linea 2). Con el identificador del escenario que se desea procesar, el algoritmo genera, mediante la función `generar_escenario(Id_escenario,pli,var)`, un archivo compuesto por el conjunto de reglas y estructura de las membranas (`.pli`), el conjunto de valores de los parámetros fijos (`.var`) y por último el conjunto de valores del escenario a procesar (`Id_escenario`) (linea 3).

A continuación, el escenario a procesar se simula un número N de veces determinado por el usuario ecólogo con el fin de promediar los resultados (lineas 4-7). Una vez promediados los resultados se evalúa la calidad de los parámetros del escenario mediante la distancia euclídea entre los datos obtenidos fruto de la simulación y los obtenidos de las observaciones reales del medio. Llamaremos a esta distancia el "error" del escenario, y nos quedaremos con aquél cuya distancia a la observaciones reales sea mínima (lineas 8-12).

La complejidad de este algoritmo viene dada por el número de parámetros p a calibrar. Cada nuevo parámetro que añadimos hace que el número de combinaciones se multiplique por el rango de posibles valores r. En el caso en que todos los parámetros tuviesen la misma longitud en el rango de valores el total de escenarios sería r^p, cada uno de ellos simulado N veces. Dado que el total de simulaciones es constante y muy inferior al total de escenarios posibles podemos determinar que el orden de complejidad del algoritmo serie es exponencial $O(r^p)$.

IV. Calibración Paralela

Dado el grado de complejidad del algoritmo serie y la importancia para el usuario ecólogo de la calibración de estos modelos. Se hace imprescindible buscar una solución al problema de la calibración aprovechando al máximo la cantidad de recursos computacionales que nos pueden ofrecer en la actualidad entornos de Supercomputación, o entornos distribuidos *Cluster* y *Grid*. Para ello, presentamos una solución paralela a la calibración que trata de minimizar la dependencia entre las tareas paralelas, minimizar las comunicaciones entre ellas y maximizar el aprovechamiento de los recursos disponibles.

Primero debemos identificar aquellos puntos susceptibles de ser paralelizados:

- Distribución de los escenarios: consiste en simular distintos escenarios simultáneamente. Dado que los escenarios no tienen dependencia entre ellos, se pueden ejecutar tantos a la vez como nodos tengamos disponibles.
- Ejecución de las repeticiones: para mitigar los efectos de la aleatoriedad propia de los `P-Systems` simulamos cada escenario un numero determinado de veces y al final promediamos los resultados. Podríamos simular cada una de estas veces en un nodo distinto de forma simultánea.

Ambas opciones de paralelización se podrían aplicar de forma conjunta. En el presente trabajo nos hemos centrado en el paralelismo a nivel de escenario, aunque hemos preparado la aplicación para futuras incorporaciones del paralelismo a nivel de repetición.

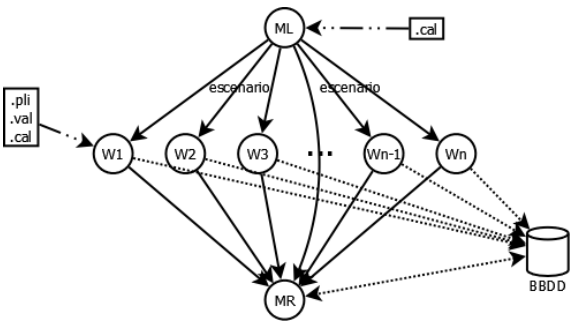

Fig. 3. Esquema de calibración paralelo.

La aplicación seguirá el paradigma Master-Worker (figura IV). Uno de los protocolos de comunicación y sincronización más utilizados para la implementación de aplicaciones paralelas en entornos distribuidos es MPI. No obstante, el procesamiento paralelo de los distintos escenarios no requiere de comunicación entre los Workers. Por otro lado, el Master solo necesita recibir el error obtenido por cada escenario con el fin de obtener el escenario que mejor se ajusta a las observaciones reales. En este caso, almacenando los resultados de los Workers en una base de datos solo se necesita notificar la finalización de cada Worker, de modo que cuando todos han finalizado el Master puede procesar los resultados. Este mecanismo de

sincronización se puede implementar también mediante las funcionalidades del sistema de colas. Por este motivo hemos decidido optar por una implementación basada en el sistema de colas del SGE. De este modo, el Master lanzará al sistema de colas el procesamiento de todos los posibles escenarios y esperara a que estos finalicen. El sistema de colas gestiona donde y cuando se debe procesar cada escenario, y cuando todas acaban se relanza el Master que procesa los resultados almacenados en la base de datos para decidir el mejor ajuste de los parámetros.

El proceso paralelo implementado se divide en tres fases principales: el **despliegue** de todos los escenarios, la **ejecución paralela** de las simulaciones por parte de los Workers y, finalmente, el **procesamiento de los resultados** para obtener el mejor ajuste de los parámetros.

En el Algoritmo 2 se detallan los pasos que realiza el Master para desplegar la ejecución de todos los escenarios. En primer lugar crea la base de datos en una localización accesible tanto desde el master como los workers (linea 1). A continuación, se generan la lista ordenada con los identificadores de los escenarios que se deben simular (linea 2). Por cada escenario de la lista, mediante la función `lanzar_simulación(Id_escenario,N)`, se encola un Worker en la cola del SGE (lineas de 3 a 5). Finalmente, se relanza a sí mismo para la última fase.

Algorithm 2 Fase de despliegue - Master

Require: fichero: cal
1: crear_base_datos(*localizacion*);
2: *lst_escenarios* := generar_lst_escenarios(*cal*);
3: **for** *escenario* in *lst_escenarios* **do**
4: lanzar_simulacion(*Id_escenario*, *N*);
5: **end for**
6: relanzar_master();

La **fase de ejecución** la realiza el Worker. Como podemos observar en el Algoritmo 3, el Worker se conecta a la base de datos (linea 1). A continuación, a partir de los archivos `.pli`, `.var`, `.cal` y del identificador del escenario `Id_escenario`, se genera el archivo con todos los datos del escenario a simular (linea 2).

Algorithm 3 Fase de ejecución - Worker

Require: código, ficheros: pli, var y cal
1: conectar_base_datos();
2: *escenario* := generar_escenario(*Id_escenario*, *pli*, *var*, *cal*);
3: **for** *i* in *N* **do**
4: *resultados*[*i*] := simular(*escenario*);
5: **end for**
6: *resultado* := promediar(*resultados*);
7: *error* := distancia_euclidea(*resultado*,*real*);
8: guardar_base_datos(*Id_escenario*, *error*);

Una vez generado el archivo se simula cada escenario un numero N de veces determinado por el usuario ecólogo (lineas 3-5), se promedia el conjunto de resultados obtenidos (linea 6) y se calcula el error producido por el escenario (linea 7). Finalmente, el error se almacena en la base de datos junto con el identificador del escenario (linea 8).

Algorithm 4 Fase de recolección - Master

1: conectar_base_datos();
2: *best* = consultar_mejor_escenario();
3: **return** *best*

La última fase es la **fase de recolección**, realizada por el master y descrita en el Algoritmo 4. Una vez finalizan todos los Workers, els SGE lanza de nuevo el proceso Master, el qual se conecta a la base de datos (linea 1) y lanza una consulta SQL para encontrar el escenario que ha generado el error mínimo (linea 2). Finalmente, devuelve los valores que mejor ajustan los parámetros no fijados por el usuario ecólogo.

V. Experimentación

El objetivo principal de esta experimentación es evaluar el coste computacional del problema de la calibración así como analizar la eficiencia que se obtiene con nuestra propuesta de paralelización del algoritmo de calibración. Para realizar las pruebas de rendimiento hemos usado un sistema PDP que modela el ecosistema del tritón Pirenaico [8]. Simularemos un período de 10 años y valoraremos la calidad de las soluciones comparandolas con las observaciones reales disponibles. Nos interesa evaluar y comprar los tiempos de ejecución de la aplicación serie y paralela.

La aplicación serie se ha ejecutado sobre un equipo con Intel Core 2 Quad a 2.4GHz y 4GB de RAM. La aplicación paralela se ha ejecutado sobre un *Cluster* homogéneo de 24 nodos con las mismas características que el equipo utilizado para la aplicación serie.

TABLA II
Tabla de resultados

Proc.	Tiempo (s)	Aceleración	Eficiencia
Serie	123240	-	-
1	126975	1	1
3	42420	2,993	0,9977
6	21300	5,961	0,9935
12	10620	11,956	0,9963
24	5460	23,255	0,9689

La tabla II nos muestra los tiempos de ejecución obtenidos. Se observa cómo la ejecución paralela con un único procesador es únicamente un 3% más lenta que la aplicación serie. Los valores de la eficiencia (Figura V) siempre superiores al 96% nos indican que el *Speedup* se mantiene cercano al ideal.

En la figura V podemos observar como el *Speedup* aumenta con el número de nodos de modo que la

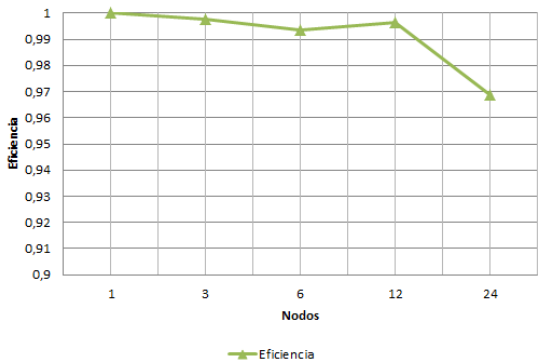

Fig. 4. Gráfico de eficiencia.

eficiencia se mantiene a medida que aumentamos el número de recursos computacionales a utilizar.

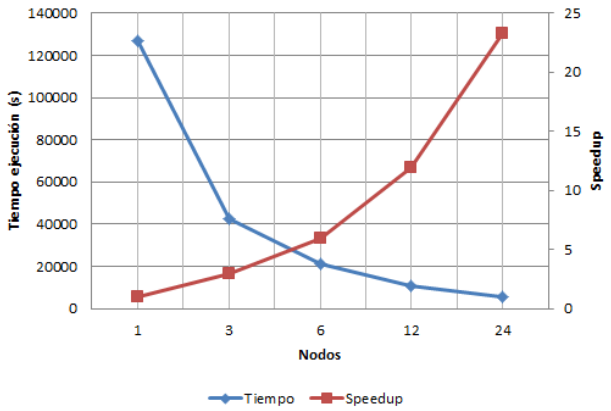

Fig. 5. Gráfico de rendimiento.

VI. Conclusiones y Trabajo Futuro

El tiempo obtenido con la ejecución serie nos muestra el alto coste computacional de la calibración. Hay que tener en cuenta que el modelo utilizado en esta experimentación es un modelo de tamaño medio-pequeño. Esto significa que para modelos superiores se aumenta la complejidad computacional del problema. Los resultados obtenidos para la ejecución paralela nos muestra como el *Speedup* mantiene un crecimiento casi lineal y la eficiencia del sistema implementado se mantiene por encima del 99% en los casos de 1 a 12 nodos, y sólo baja un $2,2\%$ en el caso de los 24 nodos. Con esta evolución podemos afirmar que se trata de una aplicación altamente escalable.

En la actualidad estamos trabajando en la incorporación de técnicas Heurísticas para la exploración del espacio de escenarios posibles. De este modo pretendemos convertir el problema de la explosión combinatoria debido a la exploración exhaustiva del espacio de soluciones, en un problema de búsqueda del mínimo global (o una aproximación lo suficientemente buena) y en un tiempo mucho más que razonable para el usuario ecólogo. Esto nos permite no solo escalar el problema, sino atacar modelos mucho más complejos y proporcionar resultados importantes para la toma de decisiones por parte de los gestores del medio. En este sentido hemos identificado y empezado a experimentar distintos algoritmos (diseño de experimentos, búsqueda del mínimo gradiente, clasificación por árboles binarios, etc.) que se están mostrando eficaces para este tipo de problemas.

Agradecimientos

El presente trabajo ha sido financiado por el Ministerio de Ciencia e Innovación mediante el contrato TIN2011-28689-C02-02. Los autores son miembros del grupo de investigación consolidado de la Generalitat de Cataluña 2009 SGR145.

Referencias

[1] Leigh, E.R., *The ecological role of Volterra's equations.*, Some Mathematical Problems in Biology, 1968.
[2] Păun, G., *Computing with membranes.*, Journal of Computer Systems Science 61, 108-143, 1998.
[3] Păun, G., Rozenberg, G. & Salomaa, A., *The Oxford Handbook of Membrane Computing.*, Oxford University Press, 2010.
[4] Cardona M., Colomer M.A., Pérez-Jiménez M.J., Sanuy D., Margalida A., *Modelling ecosystems using P Systems: The Bearded Vulture, a case study.*, Lecture Notes in Computer Science 5391, 137-156, 2009.
[5] Colomer, M.A., Margalida, A. & Sanuy, D. & Pérez-Jiménez, M.J., *A bio-inspired computing model as a new tool for modeling ecosystems: the avian scavengers as a case study.*, Ecological Modelling 222, 33-47, 2011.
[6] Cardona, M., Colomer M.A., Margalida A., Pérez-Hurtado I., Pérez-Jiménez M.J., Sanuy D., *A P-System based model of an ecosystem of some scavenger birds.*, Lecture Notes in Computer Science 5957, 182-195, 2010.
[7] Margalida, A., Colomer, M.A. & Sanuy, D., *Can wild ungulate carcasses provide enough biomass to maintain avian scavenger populations? An empirical assessment using a bio-inspired computational model.*, PLoS One 6, e20248, 2011.
[8] Colomer M.A., Montoti A., García E. & Fondevilla C., *A computational model to explain annual fluctuations and extinction risk due to climate change related water-flow in a Calotriton asper population.*, EPIC - Environment & Pyrenees International Conference, Universidad de Navarra, 2011.
[9] http://www.p-lingua.org/wiki/index.php/Main_Page
[10] I. Perez-Hurtado, L. Valencia, M.J. Perez-Jimenez, M.A. Colomer, A. Riscos-Núñez., *MeCoSim: A general purpose software tool for simulating biological phenomena by means of P Systems.*, Proceedings 2010 IEEE Fifth International Conference on Bio-inpired Computing: Theories and Applications (BIC-TA 2010), IEEE Press, 2010.
[11] Jose M. Cecilia, José M. García, Ginés D. Guerrero, Miguel A. Martínez-del-Amor, Ignacio Pérez-Hurtado, Mario J. Pérez-Jiménez., *Simulation of P systems with active membranes on CUDA.*, Briefings in Bioinformatics, 11, 3 (2010), 313-322.
[12] http://www.p-lingua.org/wiki/index.php/PLinguaCore