Speeding Up Tumor Growth Simulations Using Parallel Programming and Cellular Automata

A. J. Tomeu, A. G. Salguero and M. I. Capel

***Abstract*— The study of tumor growth biology with computer-based models is currently an area of active research. Different simulation techniques can be used to describe the complexity of any real tumor behavior, among these, "cellular automata"-based simulations provide an accurate tumor growth graphical representation while, at the same time, keep simpler the implementation of the automata as computer programs. Several authors have recently published relevant proposals, based on the latter approach, to solve tumor growth representation problem through the development of some strategies for accelerating the simulation model. These strategies achieve computational performance of cellular-models representation by the appropriate selection of data types, and the clever use of supporting data structures. However, as of today, multithreaded processing techniques and multicore processors have not been used to program cellular growth models with generality. This paper presents a new model that incorporates parallel programming for multi and manycore processors, and implements any synchronization requirement necessary to implement the solution. The proposed parallel model has been proved using Java and C++ program implementations on two different platforms: chipset Intel i5-4440 and one node of 16-processors cluster of our university. The improvement resulting from the introduction of parallelism into the model is analyzed in this paper, comparing it with the standard sequential simulation model currently used by researchers in mathematical oncology.**

***Keywords*— Cellular Automaton, Biological Patterns, High Performance Computing, Mathematical Oncology, Multicore Programming, Tumoral Growth Simulation, Parallel Programming, Speedup, Stem Cells.**

I. INTRODUCCIÓN

EN la actualidad es de todo punto necesario alcanzar una comprensión profunda y pormenorizada de la biología tumoral con el objetivo de desarrollar nuevas terapias de lucha contra el cáncer. Esta comprensión puede lograrse mediante técnicas de investigación puramente biológicas, es decir, mediante el diseño y control de experimentos en modelos animales o *in vitro*, o bien mediante el desarrollo de modelos matemáticos e informáticos que permitan simular la biología tumoral, sometiendo dicho modelo a experimentos que permitan observar su respuesta frente a distintas estrategias terapéuticas. Conocer la dinámica de crecimiento tumoral, y cómo puede llegar a modificarse como resultado de la aplicación de terapias citotóxicas, antiangiogénicas, inmunológicas o de radiación, es, sin duda, crucial para alcanzar éxito en la estrategia de lucha contra la enfermedad basada en métodos computacionales.

Aunque el modelado por medios computacionales ni puede ni debe sustituir a la investigación biológica, sí es cierto que durante determinadas fases de la investigación puede suponer una importante reducción de costes y de riesgos, frente a la experimentación directa en el laboratorio, que conlleva siempre costes más altos, protocolos de experimentación complejos, y tiempos de experimentación más largos.

Por ello, la aproximación basada en modelos matemáticos está claramente en auge [3, 4, 6], ya que proporciona información útil y rápida al personal del laboratorio, e incluso permite decidir, con carácter previo a la experimentación, si una determinada línea de investigación debería ser desarrollada –y financiada– en el laboratorio o no.

Existen varias técnicas [2, 5, 10] de simulación biológica en general y de la dinámica tumoral en particular, aunque las que imitan mejor a la realidad biológica se basan en el uso de autómatas celulares de diversos tipos, que se han propuesto en la literatura durante los últimos años. De hecho, existen múltiples propuestas para simular de forma eficiente y determinista la dinámica tumoral [8] mediante estos modelos, que utilizan el mismo modelo algorítmico de base, con ligeros cambios en las técnicas estocásticas que rigen el crecimiento del tumor, o la incorporación de capacidades *stem*, es decir, de introducción de células madre.

Sin embargo, la simulación multiescala (tanto a un nivel de célula como ente individualizado, cuanto al nivel de agregado macroscópico, compuesto por una colección de células.) de la dinámica tumoral mediante autómatas celulares estocásticos, susceptibles de alto rendimiento computacional, es ahora cuando comienza a desarrollarse.

Trabajos recientes [7, 14] proponen modelos de autómatas celulares que contemplan cambios en el entorno local de las células del autómata mediante procesos estocásticos de tipo Monte Carlo. La limitación principal de estos modelos es que son computacionalmente muy mejorables, ya que su implementación suele utilizar aproximaciones de hebra única, lo cual, cuando se han de procesar los cambios de miles de células, da lugar a serias limitaciones en el rendimiento de la aplicación software de simulación.

Se han propuesto algunas alternativas [14] a la línea de investigación anterior, que fundamentalmente tratan de optimizar el rendimiento del referido modelo. Estas alternativas utilizan la selección adecuada de los tipos de datos primitivos que representan la información, el acceso eficiente a memoria, o el uso de estructuras de datos que contemplan la evolución dinámica del espacio de memoria que éstas ocupan en cada momento.

Por otra parte, la posibilidad de utilizar el paralelismo

A. J. Tomeu, University of Cádiz, antonio.tomeu@uca.es
A. G. Salguero, University of Cádiz, alberto.salguero@uca.es
M. I. Capel, University of Granada, mcapel@ugr.es

multicore [11, 16] como técnica de programación para mejorar el rendimiento del tiempo de simulación apenas está comenzando a ser explorada en este contexto.

La aportación de este trabajo fundamentalmente consiste en proponer una técnica general de paralelización aplicable a la simulación de la dinámica de crecimiento tumoral mediante autómatas celulares, utilizando una estrategia de partición del dominio de datos, con base en los principios que desarrollamos en [17], y sobre el modelo secuencial estándar de crecimiento tumoral [14] que, a diferencia de los métodos basados en autómatas celulares estocásticos secuenciales, permite obtener linealmente aceleración de los cálculos respecto del número de procesadores en un rango apreciable de éste.

La organización del trabajo es la siguiente: en la Sección II se presenta el concepto de autómata celular adaptado al modelado biológico que pretendemos realizar. La Sección III ilustra el modelo de simulación de crecimiento tumoral utilizando un autómata celular e introduce el algoritmo básico de procesamiento secuencial estándar. La Sección IV estudia la mejora del modelo secuencial anterior mediante la propuesta de varias estrategias y la implementación con Java y C++ de algunas de ellas. La sección V muestra los resultados del procesamiento paralelo del modelo en varios nodos del *cluster* de procesadores de la Universidad de Cádiz y finalmente, recogemos las conclusiones y el trabajo futuro a desarrollar en la Sección VI.

II. AUTÓMATAS CELULARES

Existen múltiples definiciones del concepto de autómata celular en la literatura. Nosotros escogemos la propuesta establecida con carácter general en [4], adaptada a modelos biológicos reticulares en [6] y en [7], y aplicada a la simulación tumoral en [14]. Definimos pues un autómata celular (AC) como una 4-upla (ζ, ε, N^I, ρ) donde:

- ζ es una red regular discreta de células (también denominados nodos o celdas) junto con algún conjunto de condiciones de frontera en el caso finito, que definen la vecindad de las células situadas en la periferia de la red,
- ε es un conjunto finito (habitualmente con estructura de anillo abeliano) de estados que las células de la red pueden adoptar,
- un conjunto finito N^I de células que definen la vecindad de células con las que interactúa una célula dada, y
- una función de transición ρ que especifica cómo una célula de la red cambia de estado en función del tiempo y del estado de la vecindad N^I.

Dado lo anterior, un espacio de células puede definirse como una red ζ incluida en el espacio real R^d que cubre de forma homogénea una porción del espacio euclídeo d-dimensional. Cada célula está etiquetada por su posición$r \in \zeta$. La disposición espacial de las células está especificada por las conexiones con sus vecinos más cercanos, las cuáles se obtienen uniendo pares de células en alguna disposición regular. Para cualquier coordenada espacial r, el retículo de vecindad $N_b(r)$ es una lista de células vecinas definidas por

$$N_b(r) = \{r + c_i : c_i \in N_b, i = 1, \cdots, b\} \quad (1)$$

Donde b es el número de coordinación o, dicho de otra forma, el número de vecinos próximos en el retículo que interactúan con la célula ubicada en la coordenada$r$. Con N_b denotamos a la plantilla de vecinos próximos con elementos $c_i \in R^d$, para $i = 1, \cdots, b$.

En el caso que nos ocupa, y para $d = 2$, los únicos polígonos regulares que forman una teselación regular del plano son triángulos $(b = 3)$, rectángulos $(b = 4)$ y hexágonos $(b = 6)$, y nosotros escogeremos para nuestro modelo el segundo caso, de manera que

$$\zeta = \{r : r = (r_1, r_2) \in Z^2\} \quad (2)$$

El número total de células disponibles lo notaremos habitualmente por $|\zeta|$. En simulaciones de computador, los AC utilizan retículos que han de ser necesariamente finitos $(|\zeta| < \infty)$, y deben imponerse condiciones de frontera que establezcan cuáles son los vecinos de aquellas células situadas en las fronteras declaradas. En nuestro caso, utilizaremos la condición de frontera nula, es decir, consideramos que las células situadas en la periferia del retículo únicamente tienen vecinas en el interior del mismo.

El conjunto de células vecinas cuyo estado influye en una dada, que viene definido mediante la vecindad de interacción $N_b^I(r)$ para una célula r dada, de acuerdo a la siguiente expresión:

$$N_b^I(r) = \{r + c_i : c_i \in N_b^I\} \quad (3)$$

Esta vecindad o plantilla de interacción puede escogerse de varias formas, aunque nosotros nos decantaremos en nuestra simulación por la conocida como vecindad de *Moore* (Fig. 1), dada por

$$N_8^I = \begin{Bmatrix} (1,0), (0,1), (-1,0), (0,-1), \\ (-1,1), (1,-1), (-1,-1), (1,1) \end{Bmatrix} \quad (4)$$

$x_{i-1,j-1}$	$x_{i-1,j}$	$x_{i-1,j+1}$
$x_{i,j-1}$	$x_{i,j}$	$x_{i,j+1}$
$x_{i+1,j-1}$	$x_{i+1,j}$	$x_{i+1,j+1}$

Figura 1. Vecindad de *Moore* para una célula situada en $r = (i, j)$.

Por otra parte, cada célula $r \in \zeta$ tiene asignado un estado $s(r) \in \varepsilon$. Los elementos del conjunto ε pueden ser números, letras o símbolos. Nosotros escogeremos $\varepsilon = \{0, 1\}$ para modelar a células tumorales vivas (con capacidad de proliferación o migración) y muertas (cuando una célula muere por agotar su capacidad de proliferación, sucumbe al sistema inmune, etc.).

Una configuración global del autómata $s \in \varepsilon^{|\zeta|}$ queda determinada por el estado de todas las células del retículo y nuestro modelo ofrece una instantánea del estado del tumor en un instante dado del tiempo, y cambia dinámicamente ese estado utilizando una secuencia temporal discreta de acuerdo a la regla que la función de transición impone.

Finalmente, esa dinámica de evolución temporal del modelo viene determinada por la función de transición ρ, que especifica cómo una célula cambia de estado en función de su estado anterior, y de la interacción de la misma con su vecindad de células, de acuerdo a la ecuación 5.

$$\rho: \varepsilon^{\mu} \rightarrow \varepsilon \tag{5}$$

donde $\mu = |N_b^I|$. La regla es espacialmente homogénea, y no depende por tanto en forma explícita de la posición r de una célula dada. Extensiones de la definición dada para incluir homogeneidades espaciales o temporales son factibles. Si el AC es determinista la función de transición lleva a un único cambio de estado factible, mientras que si es estocástico, el nuevo estado de una célula depende de alguna distribución de probabilidad de acuerdo a

$$\rho\left(s_{N(r)}\right) = \begin{cases} z^1 \text{ con probabilidad } W\left(s_{N(r)} \rightarrow z^1\right) \\ \vdots \\ z^{|\varepsilon|} \text{ con probabilidad } W\left(s_{N(r)} \rightarrow z^{|\varepsilon|}\right) \end{cases} \tag{6}$$

donde $z^j \in \varepsilon$ y $W\left(s_{N(r)} \rightarrow z^j\right)$ es la probabilidad de transición al estado z^j, independiente del tiempo, dada la configuración de vecindad $s_{N(r)}$.

III. MODELO DE SIMULACIÓN TUMORAL

Una célula tumoral en el modelo es una entidad individual que ocupa un punto de una red bidimensional finita ζ, y que puede [12] realizar las siguientes acciones: migrar a otro punto de la red, proliferar mediante mitosis, morir, o permanecer estable en la fase G_0 del ciclo celular (Fig. 2), donde las células están quiescentes.

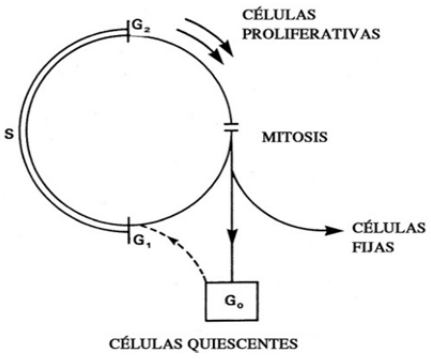

Figura 2. Dinámica de Fases del Ciclo Celular Estándar.

Las células tumorales habitualmente son de tipo *stem* (con capacidad de proliferación ilimitada) y *no-stem* (pueden efectuar una mitosis en la fase M (mitosis) del ciclo celular un número limitado de veces). Nosotros no supondremos células *stem*, pero en cambio modelaremos si una célula vive o no mediante una distribución de probabilidad

$$\rho_l\left(s_{N(r)}\right) = \begin{cases} 1 \text{ con probabilidad } W\left(s_{N(r)} \rightarrow 1\right) \\ 0 \text{ con probabilidad } W\left(s_{N(r)} \rightarrow 0\right) \end{cases} \tag{7}$$

El ajuste de esta distribución permite tener, si se desea, células *stem* también, y lo que es más importante, permite modelar la supervivencia de las células ocasionada por factores intrínsecos al tumor (angiogénesis, microambiente tumoral, etc.) o a factores extrínsecos de tipo terapéutico (respuesta a citostáticos o citotóxicos, uso de fármacos antiangiogénicos, radiación, interferones, etc.) ajustando la distribución en función de la respuesta tumoral conocida ya sea *in vivo* o *in vitro,* y contrastada en la práctica clínica o de laboratorio.

Una célula que está viva puede proliferar generando una célula hija mediante mitosis siempre que haya espacio disponible para ello en su vecindad, fenómeno que nuestro modelo contempla mediante una segunda distribución de probabilidad, condicionada a la anterior

$$\rho_p\left(s_{N(v)}\right) = \begin{cases} 1 \text{ con probabilidad } W'\left(s_{N(v)} \rightarrow 1\right) \\ 0 \text{ con probabilidad } W'\left(s_{N(v)} \rightarrow 0\right) \end{cases} \tag{8}$$

Ahora, la posición de la red $v \in s_{N(r)}$ albergará a la célula resultante de la mitosis de la célula r, si hay espacio para albergarla. Finalmente, una célula viva puede migrar, cambiando de posición dentro del tumor, siempre que haya espacio disponible para ello. Esto se ha contemplado mediante una tercera distribución de probabilidad, también condicionada a ρ_l

$$\rho_m\left(s_{N(v)}\right) = \begin{cases} 1 \text{ con probabilidad } W''\left(s_{N(v)} \rightarrow 1\right) \\ 0 \text{ con probabilidad } W''\left(s_{N(v)} \rightarrow 0\right) \end{cases} \tag{9}$$

La interpretación es exactamente igual que en la ocasión anterior. A partir del modelo estocástico descrito, y teniendo en cuenta todas las variables que afectan al desarrollo del crecimiento tumoral, y las distribuciones de probabilidad propuestas necesarias para tener en cuenta dichas variables, mostramos el pseudocódigo que proponemos como modelo algorítmico de simulación de crecimiento tumoral utilizando un autómata celular bidimensional. Este pseudocódigo está derivado del modelo algorítmico estándar en la literatura para este tipo de simulaciones [7].

El algoritmo ACT es pues una simulación secuencial del modelo de crecimiento tumoral. Inicialmente todo el segmento de tejido a simular se supone vacío de células tumorales, excepto algunas células concretas, que se plantan en un punto central del dominio a modo de semilla tumoral, y que se encuentran en la fase G_0 del ciclo de vida celular.

El modelo está parametrizado por las distribuciones de probabilidad descritas en el apartado anterior, que permiten mediante simulación estocástica de Monte Carlo decidir si una

célula muere, permanece en la fase G_0 del ciclo, o bien entra en la fase M del mismo y se reproduce por mitosis, siempre que tenga espacio suficiente alrededor para hacerlo. La dirección de propagación de la mitosis se decide mediante la distribución de probabilidad de la línea 12, que escoge una de cuatro direcciones posibles.

Posteriormente la rutina `proliferation` de la línea 13 decide, una vez establecido que la célula debe proliferar, si hay espacio para la nueva célula resultante de la mitosis, sitúa una nueva célula activa en fase G_0 a través de la rutina de actualización de posiciones de las células.

```
Algorihtm ACT
  Input:
    nGen, nCells, cells[x][y],
    W(s_N(r) → z_1), W(s_N(r) → z_1), W''(s_N(v) → 1).
  Output: simulación crecimiento tumoral
  en cells.
  1. Poner semilla tumoral inicial en
  cells[x][y];
  2. For(i=0; i<nGen; i++)
  3.  For(x=0; j<nCells; j++)
  4.    For(y=0; y<nCells; y++)
  5.      rr← random();
  6.      If(rr ≥ W(s_N(r) → z_1))
  7.        cells[x][y]=0;
  8.        goto(line 2);
  9.      If(rr < W'(s_N(v) → 1))
  10.       PH++;
  11.       If(PH ≥ NP)
  12.         Pi← random( ); i=1, 2, 3, 4.
  13.         If(proliferation()) GoTo
  (line 4);
  14.       Else GoTo (line 2);
  15.     Else
  16.       rrm ← random();
  17.       If(rrm < W''(s_N(v) → 1)){
  18.         Mi← random();i=1, 2, 3, 4.
  19.         If(migration()) GoTo (line
  4)
  20.         ElseGoTo (line 2);
  21. ReupdateCell Positions;
  22. GoTo (line 1);
end
```

[Algoritmo ACT]

Ello se hace determinando los espacios presentes en la vecindad, y migrando a estos, con probabilidades dependientes de la ausencia o presencia de espacios. Si la mitosis no es posible al no haber espacio disponible para ello, entonces se pasa al análisis de la célula siguiente.

La simulación del fenómeno de migración celular se trata de forma similar en las líneas 16 a 20 del algoritmo, estimándose probabilidades que deciden la dirección de la migración de una célula de la misma forma, y haciendo ésta efectiva si hay lugar al que migrar, no teniendo efecto alguno en caso contrario.

A partir del modelo anteriormente descrito, desarrollamos una primera simulación mediante un programa escrito en lenguaje Java de tipo secuencial, con una única hebra de ejecución. La simulación se configuró con una semilla inicial de cuatro células tumorales y cien ciclos de tiempo para una red de tejido de 32×32 células, con las características de control del modelo que se muestran en la Tabla 1.

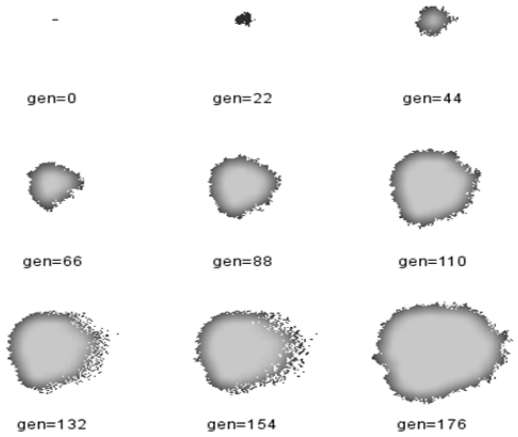

Figura 3. Evolución del modelo de simulación tumoral con la carga paramétrica descrita en la Tabla 1 para un dominio tisular de 128×128 células y 200 generaciones. Se muestran instantáneas del estado del tumor a partir de una semilla inicial de cuatro células.

En dicha tabla, el conjunto de probabilidades descrito parametriza nuestro modelo estocástico, mientras que la variable NP indica cuántas señales necesita una célula para poder proliferar (en esta aproximación basta con una) y salir de la fase G_0 del ciclo celular. PH define el número de señales de proliferación.

TABLA I
PARÁMETROS DEL MODELO DE SIMULACIÓN TUMORAL

Parámetros de la Simulación	Valores
$W(s_{N(r)} \to 1)$	0.8
$W(s_{N(r)} \to 0)$	0.2
$W'(s_{N(v)} \to 1)$	0.2
$W''(s_{N(r)} \to 1)$	0.25
PH (núm. señales proliferación)	1
NP (señales para proliferar)	1

La Figura 3 muestra una secuencia gráfica con la evolución del tumor simulada a partir del modelo descrito en el algoritmo `ACT` de acuerdo a los parámetros citados en la tabla. Estos parámetros garantizan un crecimiento rápido del tumor invadiendo el tejido circundante de forma densa y compacta, según se describe en la referencia [13].

La curva de crecimiento del número de células vivas presentes en la muestra simulada se ilustra en la Fig. 4. En ella se aprecia el clásico comportamiento de crecimiento *gompertziano*, que regula la dinámica tumoral *in vivo*. El tiempo en la simulación tiene carácter discreto y cada intervalo de tiempo discreto $\Delta t = 1/24$ supone una hora; 24

pasos de simulación representan un día. Nosotros procesamos 100 pasos de tiempo discreto, equivalente a algo más de cuatro días de evolución del tumor, para generar la curva.

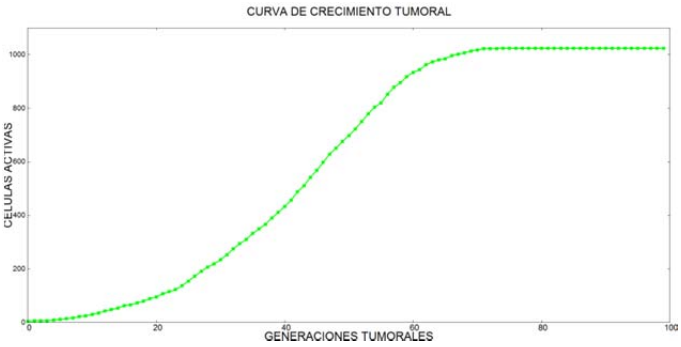

Figura 4. Simulación de crecimiento tumoral con el modelo de hebra única, donde se aprecia el comportamiento *gompertziano* clásico. La curva muestra la función discreta $celulas = F(generación\ tumoral)$.

Naturalmente, simulaciones de este tamaño no resultan de especial utilidad en la práctica. En el trabajo [17] se establece que en tumores de dinámica de crecimiento bien conocida, como son los de mama, un tumor clínicamente detectable (bien sea mediante exploración clínica o utilizando técnicas de imagen.) de un gramo de peso tiene al menos 10^9celulas. El llegar a este punto puede requerir entre dos y ocho años de tiempo de evolución, dependiendo de la agresividad de las células tumorales, representada por el tipo histológico.

En consecuencia, para procesar simulaciones más coherentes con la dinámica tumoral real, hemos de manejar *arrays* de tejidos de al menos $10^5 \times 10^5$ células. La implementación de las simulaciones ha de incluir entre 17520 y 70080 pasos de tiempo discreto. A modo de ejemplo ilustrativo, para una simulación de $10^3 \times 10^3$ células que se ejecuta con 5000 pasos de tiempo discreto (unos doscientos días de evolución de la patología) se requiere de un tiempo de procesamiento de 133,13 segundos, sobre un procesador *Intel core© i5-4440* a 3.10 GHZ, en la simulación de hebra única.

IV. MEJORA DE RENDIMIENTO DEL MODELO

La limitación principal del modelo presentado radica pues en el tamaño del segmento de tejido modelado, que únicamente permite simular tumores con tiempos de ejecución razonables de unos cuantos cientos de células. Tumores de este tamaño no son clínicamente detectables en la realidad, y tampoco pueden considerarse una diana terapéutica válida si queremos simular modelos terapéuticos computacionalmente. Es por tanto necesario desarrollar simulaciones de mayor tamaño.

Podría quedar ahora reducido a poco más de dos minutos de tiempo de ejecución y, por tanto, es una buena mejora respecto de la resolución inicial del problema. Sin embargo, esta mejora se ha obtenido mediante una aproximación de hebra única, que utiliza únicamente una fracción $1/n$ de la potencia que el procesador ofrece si suponemos que este dispone de *n* cores físicos. El objetivo principal de este trabajo ha sido el desarrollo de un modelo paralelo de crecimiento tumoral de mejor rendimiento, que utilice todos los *cores* disponibles en la máquina para optimizar el tiempo de duración de la simulación. A partir de él, mediremos cuál es la ganancia de rendimiento (*speed up*) del modelo paralelo frente al secuencial.

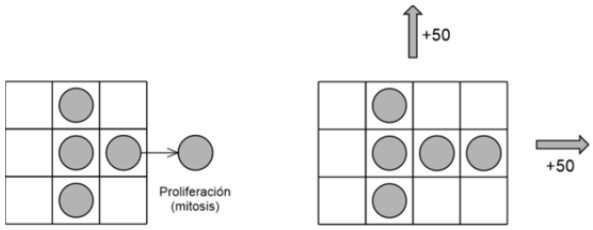

Figura 5. Crecimiento Dinámico del Dominio.

Para lograr mejoras en el tiempo de cálculo existen varias estrategias para elevar el rendimiento de la simulación:

- Selección de tipos de datos: se han utilizado tipos de datos que ocupan únicamente un byte de memoria. En concreto, la simulación modela el segmento de tejido mediante un *array* bidimensional de tipo byte, que requiere únicamente un byte frente a los dos necesarios si se utiliza el tipo *short* o los ocho del tipo *long*. Esto permite situar en memoria cache segmentos mayores de la matriz de tejido y reducir los tiempos de acceso a la información, aumentando el segmento de datos que pueden ubicarse en cache.
- Dominios de crecimiento dinámico: en lugar de utilizar desde el comienzo de la simulación un *array* de 400×400 células, si necesitamos una simulación de tamaño 80.000, es posible utilizar un *array* inicial de pequeñas dimensiones, que únicamente contiene la semilla tumoral. Conforme el tumor va creciendo, la simulación ampliará el *array* que la contiene (ver Fig. 5) de forma dinámica. Esto puede conseguirse mediante el uso de estructuras de datos dinámicas.
- Procesamiento paralelo del *array*: la simulación se ha escrito de forma que el dominio tisular se divida en tantos subdominios como *cores* tiene el procesador. Posteriormente, se crea una tarea por cada *core*, encargada de procesar en paralelo con las demás tareas un único subdominio de datos (ver Fig. 6).
- Análisis utilizando dos lenguajes de programación de propósito general con presencia de máquina virtual en un caso y sin ella en el otro, que permiten la programación multihebra: Java y C++, utilizando la revisión mayor de C++ de 2011 y la menor de 2014. Los códigos que hemos utilizado para desarrollar las simulaciones y el análisis que se expone en este trabajo se encuentran disponibles en http://antoniotomeu.wix.com/atomeu#!tumoralgrotwthsimulation/ca1p
- Uso de ejecutores [16] cuando ello sea posible, ya que crear y destruir tareas que procesan el *array* de tejido cada vez que este deba ser recalculado presenta altas latencias de ejecución que no son deseables. Para resolver este último problema, es mejor utilizar un ejecutor, que crea una reserva (pool) de hebras. Las hebras del pool se crean de una vez, pero mediante el envío de tareas a las hebras para

que las procesen, éstas últimas se pueden reutilizar tantas veces como sea necesario. Actualmente Java incorpora *ejecutores*, pero C++ aún no lo hace de forma nativa, estando prevista su incorporación en una futura versión.

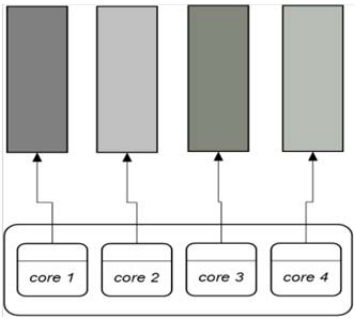

Figura 6. Procesamiento paralelo del *array* de tejidos en un procesador con cuatro *cores*. Cada *core* procesa una hebra diferente sobre un subdominio de datos distinto.

El uso del procesamiento paralelo aporta innegables ventajas cuando la simulación se desarrolla sobre uno de los procesadores *multicore* que están disponibles. Actualmente, las hebras disponibles procesan en paralelo y libremente sus respectivos subdominos de datos, siendo no obstante necesario imponerle a la solución paralela dos condiciones de sincronización:

- Condición de barrera: es necesaria para resincronizar las tareas que procesa el ejecutor cuando una generación del tumor ha sido procesada y se debe comenzar a procesar la siguiente.
- Condición de control de exclusión mutua. Cuando una célula prolifera o migra a un segmento de datos diferente de aquél en que se encuentra, la hebra que controla su crecimiento invade un subsegmento de datos propio de otra hebra; es necesario, por tanto, hacerlo respetando la propiedad de exclusión mutua en el acceso a los datos, de forma que la hebra propietaria del segundo segmento de datos pueda percibir el cambio en los datos y utilizarlos en su propia tarea de forma segura. Hemos utilizado cerrojos reentrantes para solventar esta cuestión en ambos lenguajes.

Ambas condiciones de sincronización reducirán, sin duda alguna, el máximo paralelismo del que potencialmente podría disponerse, pero son absolutamente esenciales para lograr una versión paralela de la simulación que cumpla con la propiedad de seguridad.

V. ANÁLISIS Y RESULTADOS

A partir del modelo paralelo, desarrollamos diferentes experimentos de simulación, con C++ y Java, utilizando dos plataformas:

- El *cluster* de apoyo a la investigación de la Universidad de Cádiz, procesando nuestras simulaciones sobre un nodo del mismo. El cluster ejecuta el sistema operativo *Linux Red Hat Enterprise 6.4*. Los nodos de entrada utilizan la versión *Server High Availability*, y los nodos de cálculo la versión *Compute Node*. La gestión de los nodos es tarea del *Cluster Managment Utility* de HP y cada nodo incorpora dos procesadores *Intel Xeon E5 2670* a 2.6 GHZ de 8 *cores*, totalizando 16 *cores* por nodo. El *hyperthreading* no está activo. Cada nodo dispone de 128 GBytes de RAM y 123 MBytes de *smart cache* (L3). La versión del *Java Development Kit* disponible en el cluster fue 1.7.0_40 y la del compilador g++ fue Red Hat 4.4.7-11 (revisión menor de 2014).
- Una plataforma estándar basada en un *chip Intel Core i5-4440*, con 4 *cores* físicos a 3.10 GHZ sin *hyperthreading*, con 8 GBytes de RAM, 6 MBytes de cache L3, y sistemas operativos Windows 10 y Linux Fedora 22. Se realizaron pruebas sobre ambos sistemas operativos con el *Java Development Kit* versión 1.8.0-51 y con el compilador g++ Read Hat 5.3.1-6 (revisión menor de 2014).

TABLA II
TIEMPOS DE EJECUCIÓN (SEGUNDOS) SEGÚN TIPOS DE DATOS

Retícula	Tiempo (long)	Tiempo (Byte)
6000	59.49	57.87
7000	77.58	71.39
8000	96.69	86.55
9000	114.33	122.45
10000	165.95	149.80
11000	211.41	189.61
12000	251.04	225.87
13000	293.87	264.66
14000	341.48	307.36

La primera estrategia de mejora, consistente en emplear una versión de la simulación de hebra única con tipos de datos primitivos de diferente tamaño (para procesar *arrays* de esta longitud, el tamaño del *heap* que por defecto utiliza la máquina virtual de Java es insuficiente y debe aumentarse. En nuestro caso, aumentamos ese tamaño a 512MB, mediante la parametrización del comando `java` con el *flag* `–Xmx512m`.), ofreció los resultados que se muestran en la Tabla 2 para el lenguaje Java. Para obtenerla, se realizaron simulaciones con un dominio tisular de tamaño creciente, para un total de 800 generaciones de tiempo discreto en todos los casos, equivalentes a aproximadamente 33 días de evolución del tumor. Se empleó un único nodo del *cluster*, utilizando un único *core* de ese nodo. En la Tabla 2 es posible apreciar como la simple elección del tipo de datos (*long* frente a *byte*) del *array* bidimensional con el que se simula el dominio tisular produce un impacto razonablemente significativo en el rendimiento temporal de la simulación, a pesar de que la misma dispone de una única hebra para procesar el dominio tisular.

Concluimos que, para modelar las células, la simple elección del tipo de dato adecuado ofrece mejoras en el tiempo de ejecución de hasta un 30%. Por tanto, en las versiones paralelas del modelo a desarrollar en ambos lenguajes, se escogerá tipos de datos que únicamente requieren un *byte* de memoria.

La segunda estrategia, basada en el crecimiento dinámico del dominio tisular, está actualmente en fase de síntesis,

utilizando para ello el modelo paralelo propuesto, hibridado con el crecimiento dinámico del dominio tisular descrito. No disponemos de datos de rendimiento contrastados con rigor que ofrecer por ahora.

La tercera estrategia utiliza programación paralela multihebra para acelerar la simulación mediante un enfoque basado en el paralelismo de datos, aplicada al algoritmo básico `ACT` descrito anteriormente. La Figura 6 ilustra esta estrategia si suponemos un procesador idealizado de cuatro *cores*. Puede verse como el dominio tisular se procesa en paralelo mediante cuatro tareas que tienen un *core* dedicado, cada una de las cuales es responsable del procesamiento del segmento de un subdominio tisular distinto.

En esta simulación y para ambos lenguajes (C++ y Java), se ha optado por unas condiciones de simulación más exigentes y más próximas a la realidad biológica, tanto en el tamaño del dominio tisular, que ahora es de $10^5 \times 10^5$, como en el tiempo de simulación, que ahora ha sido de 1500 pasos de tiempo discreto. Se ha mantenido la carga paramétrica del modelo explicitada en la Tabla 1.

Es necesario aclarar que el objetivo del experimento que hemos desarrollado no es comparar los lenguajes Java y C++ entre sí, puesto que actualmente ofrecen capacidades de programación paralela diferentes; C++ aún no dispone de ejecutores de forma nativa, y por tanto estaríamos comparado cosas distintas. El objetivo del experimento es mostrar qué capacidad de aceleración (*speedup*) ofrecen Java y C++ con respecto al algoritmo secuencial estándar [14], con las capacidades de programación paralela con que cada lenguaje cuente en la actualidad.

Con ambos lenguajes, la simulación fue desarrollada sobre uno de los nodos del *clúster* cuyas características ya hemos citado, y para un número de tareas paralelas crecientes. Comenzamos con simulaciones de hebra única, que responden al modelo secuencial estándar [7], y llegamos a simulaciones que utilizan hasta 32 tareas. Los tiempos de base (tiempos requeridos para efectuar la simulación con una única hebra de ejecución. No hay presentes por tanto ni paralelismo, ni concurrencia) para el cálculo del *speedup* en las simulaciones de hebra única fueron de 208.38 segundos para Java, y de 87.54 segundos para C++ (en la compilación del código del modelo el lenguaje C++ se aplicaron las optimizaciones derivadas del uso del *flag* `-O3`). Los resultados de tiempos y *speedup* del experimento se muestran en la Figura 7. El s*peedup* se ha obtenido contrastando nuestro modelo paralelo con el modelo de simulación secuencial estándar que propone la literatura [9]. Comprobamos cómo ha existido paralelismo real (una tarea por *core*) durante la ejecución de la simulación, hasta llegar a ocupar el total de dieciséis cores del nodo. Posteriormente, se ha ejecutado la simulación utilizando paralelismo simulado (concurrencia), al existir más tareas que cores, llegándose a totalizar un máximo de dos tareas por *core*.

En ambos lenguajes se aprecia cómo la introducción del paralelismo de datos mejora los tiempos de ejecución (que se reducen al aumentar el número de hebras) y también el *speedup* (que aumenta) del programa de simulación, frente al modelo algorítmico secuencial estándar. Las mejoras alcanzan el punto óptimo cuando se tiene aproximadamente una tarea por *core*, para entrar posteriormente en una fase de *plateau*, si se continúa aumentando el número de tareas.

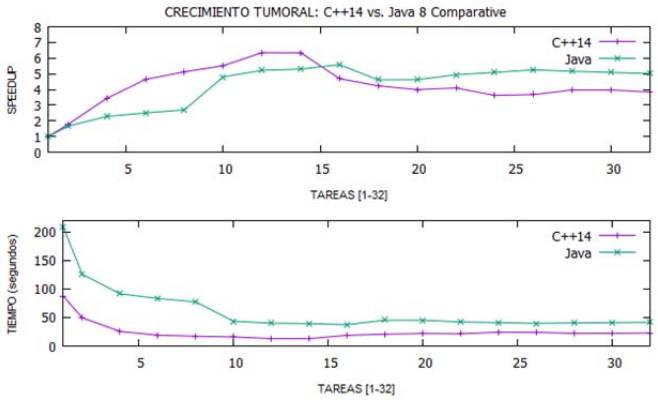

Figura 7. Resultados de las simulaciones de hebra múltiple en un nodo del cluster con 16 *cores* para los lenguajes Java y C++.

Esto responde a comportamientos ya observados en [17] y también por otros autores [10], que analizaron problemas de naturaleza similar, cuya solución se obtiene mediante un cálculo fundamentalmente numérico.

La observación que se manifiesta al aplicar nuestra estrategia de paralelización es que, una vez ocupado un *core* por una tarea, es mejor dejarlo trabajar en ella, puesto que el aumento del paralelismo de datos no va a mejorar el rendimiento de la aplicación. También hemos de mencionar que el *speedup* óptimo obtenido con ambos lenguajes (5.57 para Java y 6.34 para C++) está lejos del máximo teórico posible en un nodo de esta máquina, que es de dieciséis.

La segunda simulación desarrollada sobre la plataforma *Intel i5* siguió unas líneas similares, comenzando con una simulación de hebra única y llegando a simulaciones que utilizan hasta 8 tareas, dado un máximo de 4 *cores* físicos. En este caso, los tiempos base para la simulación de hebra única se muestran en la tabla III, para un tamaño de la simulación de $6 \cdot 10^3 \times 6 \cdot 10^3$ y 1500 pasos de tiempo discreto.

TABLA III
TIEMPOS BASE (SEGUNDOS) SOBRE INTEL i5-4440

	Java	C++
Linux	76.05	57.87
Windows	100.56	71.39

El resultado completo del experimento en esta plataforma se ilustra en las Figuras 8 y 9. La Figura 8 muestra los resultados del experimento con el sistema operativo Windows 10, y nos muestra cómo Java acelera mejor que C++ en todo el dominio de tareas analizado, obteniéndose *speedups* discretos (de 2.0 para Java y de 1.3 para C++) sobre el máximo teórico posible, que es de 4.0

En tiempo y para este sistema operativo, Java se comporta peor que C++ para un número de tareas inferior al número de *cores*, observándose una inversión de la tendencia cuando el número de tareas es mayor al número de *cores*, donde Java se comporta mejor que C++, dado que ofrece una mejor gestión del paralelismo. Con ambos lenguajes se mejora a la versión secuencial estándar propuesta en la literatura [13].

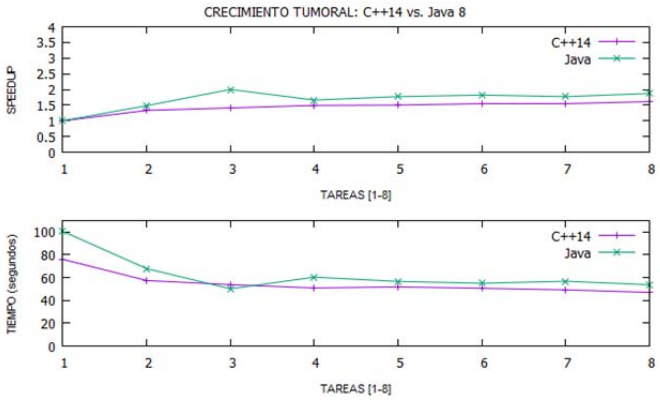

Figura 8. Resultados de las simulaciones de hebra múltiple en *Intel i5* de 4 *cores* para los lenguajes Java y C++ bajo sistema operativo *Windows 10*.

La Figura 9 muestra los resultados del experimento sobre la plataforma *Intel i5*, utilizando como sistema operativo *Linux Fedora 22*. Los resultados son similares a los de la Figura 8 con *Windows* 10, excepto que la diferencia en *speedup* entre Java y C++ aumenta, y los tiempos obtenidos son algo mejores, resultado de mejor rendimiento que *Linux* ofrece sobre *Windows* en general.

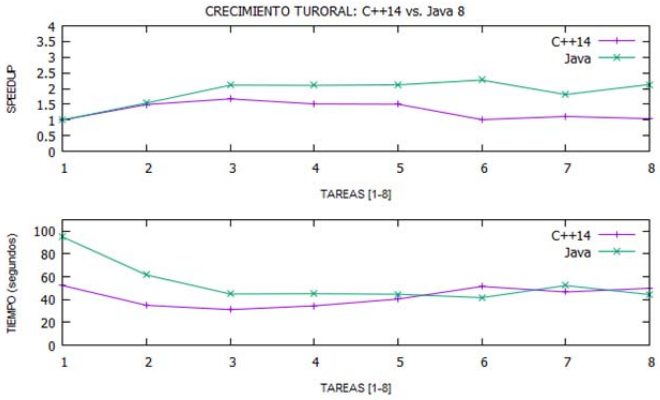

Figura 9. Resultados de las simulaciones de hebra múltiple en *Intel i5* de 4 *cores* para los lenguajes Java y C++ bajo sistema operativo *Linux Fedora 22*.

El hecho clave derivado del análisis anterior es inmediato: dado que en general, utilizar menos tareas que *cores* físicos disponibles no es aconsejable, puesto que el tiempo total de cálculo es peor, podemos afirmar que teniendo un mínimo de una tarea por *core*, Java obtiene mejores *speedups* que C++, hecho que había observado tanto por otros autores [10], como por nosotros [17] en experimentos de naturaleza análoga. Este comportamiento es atribuible en exclusiva al uso de ejecutores en el código Java, que reutiliza las hebras [16], mientras que en la versión actual de C++, al no estar incorporados de forma nativa, estas deben ser creadas y destruidas en cada iteración, con el consiguiente coste en rendimiento temporal, dada la desventaja que C++ presenta de partida. Puede apreciarse también, comparando las Figuras 8 y 9, que el sistema operativo empleado no altera este hecho. A pesar de todo, un aspecto ya constatado en el cluster se mantiene en la plataforma *Intel i5*: el *speedup* máximo obtenido es razonable frente a la simulación de hebra única, pero está lejos del máximo teóricamente posible, que en este caso es de 4.0.

Como habían constatado tanto otros autores [10] como nosotros [17], C++ "acelera" peor que Java en condiciones de paralelismo de datos; sin embargo, considerando como parámetro de análisis el tiempo total de ejecución, C++ se comporta mucho mejor que Java cuando el número de tareas es inferior al número de cores, aunque ambos lenguajes tienden a igualarse conforme el número de tareas crece, con diferencias poco significativas para simulaciones de los tamaños analizados. De nuevo, y tal y como se concluye en [10, 17], Java se constituye en una alterativa de primera magnitud a C++ para el desarrollo de soluciones basadas en paralelismo de datos.

Nuestra previsión es que aún queda margen para mejorar el speedup obtenido notablemente, lo cual se puede conseguir tanto mediante la incorporación de la segunda estrategia (crecimiento dinámico del dominio tisular) como mediante una reducción del número de bloqueos de control de exclusión mutua. Esta excesiva sincronización se puede reducir bien asignando un cerrojo diferente a cada subsegmento del dominio tisular frente al cerrojo único utilizado, o bien utilizando un enfoque libre de bloqueos basado en el uso de memoria transaccional software.

VI. CONCLUSIONES Y TRABAJO FUTURO

Del desarrollo expuesto podemos concluir que, aunque la simulación computacional del crecimiento de tumores de distintos tipos es actualmente un área de investigación muy activa dentro de la Biología Computacional, sin embargo, las estrategias de simulación que se están utilizando en la actualidad están lejos de ser eficientes desde el punto de visto del procesamiento paralelo para *multicores*.

Los trabajos recientes de otros autores que hemos analizado [1, 7], [8, 9] y [12-14], utilizan en todos los casos un enfoque de hebra única, donde las únicas técnicas empleadas en la mejora del rendimiento se basan en la elección del tipo de datos más adecuado (condición ésta considerada como necesaria, pero no suficiente para obtener mejor rendimiento) y crecimientos dinámicos del dominio tisular, que son implementados con estructuras de datos más o menos sofisticadas, proporcionadas por la biblioteca de plantillas de C++ o de contenedores de Java.

Ninguno emplea un enfoque de cálculo de hebras paralelas, aunque sí otras propuestas de mejora de rendimiento [9] basadas en el uso de estructuras de datos dinámicas, que no mejoran los resultados expuestos aquí, al carecer de procesamiento paralelo.

En este trabajo hemos mostrado como la combinación de un tipo de dato básico adecuado, junto con un dominio tisular implementado mediante un array estándar, permite alcanzar valores de *speedup* razonables, frente al modelo secuencial estándar en función, siempre, de un paralelismo de datos con un número de tareas igual al de *cores* físicos disponibles en la

máquina. Más concretamente y para ambas plataformas, hemos observado que para un número de tareas igual o mayor al de *cores*, Java logra mejores aceleraciones. Observamos también que crecimientos ulteriores del número de tareas no mejoran el rendimiento. Analizando la variable temporal concluimos que Java se constituye como una alternativa a C++ en el ámbito del paralelismo de datos. Nuestro trabajo futuro va a seguir varias líneas:

- Por una parte y desde un punto de vista de estudio de la biología tumoral computacional, pretendemos aplicar el modelo y la técnica general de paralelización presentados aquí a tumores sólidos de filiación concreta como son los de próstata, mama o colon, para los cuales existen modelos matemáticos de crecimiento bien contrastados, que admiten ser paralelizados.
- Por otra parte, incorporando el crecimiento dinámico del dominio tisular (segunda estrategia), pretendemos mejorar el modelo general presentado ya que prevemos reducir el número de bloqueos de exclusión mutua mediante el uso de cerrojos múltiples asignados a cada subdominio de datos, y eliminar los bloqueos mediante un modelo alternativo de paralelismo basado en el uso de memoria transaccional software.
- También nos interesa (de forma colateral) conocer cómo se comporta, en términos de rendimiento, nuestro modelo si se desarrolla con otros lenguajes, tanto los que suponen presencia de máquina virtual, como C#, como los que no la incluyen, que se basan en APIs de alta orientación al paralelismo, tales como son MPI y OpenMP.
- Además, y puesto que *in vivo* los tumores desarrollan su progresión en tres dimensiones, pretendemos desarrollar las simulaciones incorporando la tercera dimensión.
- Finalmente, y para posibilitar un análisis de la estrategias terapéuticas mediante modelado computacional basado en múltiples hebras de ejecución, es nuestra intención incorporara modelo matemático presentado características biológicas, como son la angiogénesis o la inhibición del crecimiento inducida por la quimiotaxis, así como el efecto de terapias basadas en la presencia de fármacos citotóxicos, citostáticos, etc.

REFERENCIAS

[1] M.J. Aguilar, L. Baena, A.M. Sánchez, R. Guisado, E. Hermoso, N. Mur and M.I. Capel, "Triglyceride levels as a risk factor during pregnancy", *Nutrición Hospitalaria*, Vol. 32, No. 2, pp. 517-527, 2015.

[2] O. Bandman, "Mapping Physical Phenomena onto CA-Models", *Proceedings of AUTOMATA-2008, Luniver Press*, pp. 381-395, 2008.

[3] M.I. Capel-Tuñón, L. K. Dillon, T. M. Casey, B. H. Cheng and R. M. Seymour, "Towards modal modelling of biological systems", *Technical Report: Michigan State University*, pp. 1-12, 2008.

[4] B. Chopard and M. Droz, "Cellular Automata in Modeling of Physical Systems", *Cambridge University Press*, 1998.

[5] R. Denzer, P. Fitch, I.N. Athanasiadis and D.P. Ames, "Parallel Simulation of Enviromental Phenomena". *Proceedings of 7h Int. Congress on Env. Modeling and Software*. pp. 347-353. San Diego, USA, 2014.

[6] A. Deutsch and S. Dorman, "Cellular Automata Model of Biological Patterns, Characterization, Applications and Analysis". *Birkhäuser*, 2005.

[7] H. Enderling, A. Anderson, M. Chaplain, A. Beheshti, L. Hlatky and P. Hahnfeldt, "Paradoxical Dependencies of Tumor Dormancy and Progression on Basic Cell Kinetics". *Cancer Research*, No. 69, pp. 8814-8821, 2009.

[8] J. Galle, M. Hoffman and T. Aust, "From single cells to tissue architecture. A bottom-up approach to modelling the spatio-temporal organization of complex multicellular systems". *Journal of Mathematical Biology*, No. 58, pp. 261-283, 2009.

[9] P. Gerlee and A. Anderson, "Evolution of Cell Motility in and Individual-Based Model of Tumor Growth". *Journal of Theoretical Biology*, Vol. 259, pp.. 67-83, 2009.

[10] A.K. Goram and A. From, "A Comparative analysis between parallel models in C/C++ and C#/Java", (http://kth.diva-portal.org/smash/get/diva2:648395/FULLTEXT01.pdf), 2013.

[11]H. Gould, J. Tobochnick and W. Christina, "An Introduction to Computer Simulation Methods", *Applications to Physical Systems*, Pearson-Addison Wesley, 2006.

[12] J.L. Henessy, D.A. Patterson, "Computer Architecture", *Elsevier*, Amsterdam, 2012.

[13] C.I. Morton, M. Hlatky, P. Hahnfeldt and H. HENDERLING, "Non-Stem Cancer Cell Kinetics Modulates Solid Tumor Progression", *Theoretical Biology and Medical Modelling*, Vol. 8, pp. 48-62, 2011.

[14] J. Polesczuk and H. Enderling, "A High-Performance Cellular Automaton Model of Tumor Growth with Dynamically Growing Domains", *Applied Mathematics*, Vol. 5, pp. 144-152, 2014.

[15] G. Stegmayer, O. Chiotti, "Volterra NN-based behavioral model for new wireless communications devices", *Neural Computing and Applications*, Vol. 18, pp. 283-291, 2009.

[16] V. Subramanian, "Programming Concurrency On The JVM. Mastering, Synchronization and Actors", *The Pragmatic Bookshelf*, 2012.

[17] A.J. Tomeu, A.G. Salguero, M.I. Capel, "A Parallelisation Tale Of Two Languages", *Annals of Multicore and GPU Programming*, pp. 81-94, 2015.

[18] J. Visvader and G. Lindeman, "Cancer Stem Cells in Solid Tumours: accumulating evidence and unresolved questions", *Nat. Rev. Cancer*, Vol. 8, pp. 755-768, 2008.

Antonio J. Tomeu received a BsC and MsC in Computer Sciences from de University of Granada, Granada, Spain in 1992, and a PhD in Mathematics from University of Cádiz, Cádiz, Spain, in 2002. He is Associate Professor of Computer Science in University of Cádiz, coordinator of TECDIS (Red Iberoamericana de Investigación en Tecnologías Concurrentes, Distribuidas y Paralelas), and editor in chief of *Annals of Multicore and GPU Programming*. His current research interests are applications of parallel programming techniques to simulate natural phenomena.

Alberto G. Salguero received a BsC and MsC in Computer Science from de University of Granada, Granada, Spain in 2004, and a PhD in Computer Sciences from University of Cádiz, Cádiz, Spain, in 2013. He is Associate Professor of Computer Sciences in University of Cádiz, member of TECDIS (Red Iberoamericana de Investigación en Tecnologías Concurrentes, Distribuidas y Paralelas. His current research interests are applications of parallel programming techniques to simulate natural phenomena and ontologies applied to data integration.

Manuel I. Capel received a BsC and MsC in Physics from de University of Granada, Granada, Spain in 1982, and a PhD in Computer Science from University of Granada, Granada, Spain, in 1992. He is Full Professor of Computer Sciences in University of Granada, coordinator of TECDIS (Red Iberoamericana de Investigación en Tecnologías Concurrentes, Distribuidas y Paralelas), and editor in chief of *Annals of Multicore and GPU Programming*. His current research interests are applications of parallel programming techniques to simulate natural phenomena.