DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajos Prácticos I y II

Sistemas Complejos en Máquinas Paralelas
Segundo Cuatrimestre de 2015

Integrante	LU	Correo electrónico
Panarello, Bernabé	194/01	bpanarello@gmail.com
Oller, Luca	667/13	ollerrr@live.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Introducción

En este informe explicaremos el desarrollo del trabajo pràtico I, detallaremos el modelo matemático utilizado para encarar el problema a través de la ecuación diferencial presentada por la cátedra. Propondremos una discretizaciòn, a nuestro parecer, adecuada, su implementaciòn secuencial y las implementaciones en paralelo. Finalmente, mostraremos los resultados conseguidos en el crecimiento del tumor y las comparaciones entre las implementaciones realizadas.

2. Discretización e implementación

2.1. Método de discretización elegido

Nuestra primera intención fue utilizar la discretización de Crank Nicolson para lograr la estabilidad incondicional, pero la complejidad de dicha discretización hacía muy difícil la detección de errores en el código. Antes de continuar con esta discretización Decidimos intentar una implementación mucho más sencilla utilizando una discretización explícita. Encontramos que con un delta t pequeño de 0.01 días el algoritmo no diverge por lo que finalmente adoptamos dicha discretización. Discretización de cada término: Inicialmente partimos de la ecuación propuesta en el enunciado:

$$\frac{\partial C}{\partial t} = \frac{\partial}{\partial x}\left(D\,\frac{\partial C}{\partial x}\right) + \frac{\partial}{\partial y}\left(D\,\frac{\partial C}{\partial y}\right) + \frac{\partial}{\partial z}\left(D\,\frac{\partial C}{\partial z}\right) + \rho\, C\, ln\,\left(\frac{C_m}{C}\right) - R(s,C)$$

Figura 1: Ecuación propuesta en el enunciado.

Aplicando la regla de la cadena, obtenemos:

$$\frac{\partial C}{\partial t} = \frac{\partial D}{\partial x}\cdot\frac{\partial C}{\partial x} + D\cdot\frac{\partial^2 C}{\partial x^2} + \frac{\partial D}{\partial y}\frac{\partial C}{\partial y} + D\frac{\partial^2 C}{\partial^2 y} + \frac{\partial D}{\partial z}\frac{\partial C}{\partial z} + D\frac{\partial^2 C}{\partial^2 z} + \rho\cdot C\cdot \ln Cm/c - R(s,C)$$

donde D debe interpretarse como D(x,y,z) y C como C(x,y,z). Si llamamos F a la función de discretización explícita, obtenemos:

$$F(\frac{\partial C}{\partial t}) = F(\frac{\partial D}{\partial x}\frac{\partial C}{\partial x} + D\frac{\partial^2 C}{\partial x^2} + \frac{\partial D}{\partial y}\frac{\partial C}{\partial y} + D\frac{\partial^2 C}{\partial^2 y} + \frac{\partial D}{\partial z}\frac{\partial C}{\partial z} + D\frac{\partial^2 C}{\partial^2 z} + \rho * C * \ln Cm/c - R(s,C))$$

$$F(\frac{\partial C}{\partial t}) = F(\frac{\partial D}{\partial x})\cdot F(\frac{\partial C}{\partial x}) + F(D)\cdot F(\frac{\partial^2 C}{\partial x^2}) + F(\frac{\partial D}{\partial y})\cdot F(\frac{\partial C}{\partial y}) + F(D)\cdot F(\frac{\partial^2 C}{\partial^2 y}) + F(\frac{\partial D}{\partial z})\cdot F(\frac{\partial C}{\partial z})$$

$$+F(D)\cdot F(\frac{\partial^2 C}{\partial^2 z}) + F(\rho)\cdot F(C)\cdot F(\ln Cm/c) - F(R(s,C))$$

En nuestro caso, F sólo transforma los términos que representan derivadas de acuerdo al esquema explícito de la siguiente manera:

- $$F(\frac{\partial G}{\partial u})(u_0) = \frac{G(u_0+\Delta u) - G(u_0-\Delta u)}{2\Delta u}$$
- $$F(\frac{\partial^2 G}{\partial^2 u})(u_0) = \frac{G(u_0+\Delta u) - 2G(u_0) + G(u_0-\Delta u)}{\Delta u^2}$$
- $$F(\frac{\partial C}{\partial t})(t_0) = \frac{C(t_0+\Delta t) - C(t_0)}{\Delta t}$$

Donde u es una variable espacial y t una variable temporal.
Si llamamos D's(x,y,z) a la función de la derivada discretizada de D en x,y,z respecto a s, obtenemos:

$C^{n+1}(x,y,z) = C^n(x,y,z)$ + (D'x(x,y,z) $\cdot \frac{C^n(x-\Delta x,y,z) - C^n(\Delta x,y,z)}{2\cdot\Delta x} + D(x,y,z)\cdot\frac{C^n(x+\Delta x,y,z) - 2\cdot C^n(x,y,z) + C^n(x-\Delta x,y,z)}{\Delta x^2} +$

$D'y(x,y,z)\cdot\frac{C^n(x,y-\Delta y,z) - C^n(x,y+\Delta y,z)}{2\cdot\Delta y} + D(x,y,z)\cdot\frac{C^n(x,y+\Delta y,z) - 2\cdot C^n(x,y,z) + C^n(x,y-\Delta y,z)}{\Delta y^{n2}} +$
$D'z(x,y,z)\cdot\frac{C^n(x,y,z-\Delta z) - C^n(x,y,z+\Delta z)}{2\cdot\Delta z} + D(x,y,z)\cdot\frac{C^n(x,y,z+\Delta z) - 2\cdot C^n(x,y,z) + C^n(x,y,z-\Delta z)}{\Delta z^2} +$
$\rho(D(x,y,z))\cdot C^n(x,y,z)\cdot \ln\frac{CM}{\sqrt[2]{C^n(x,y,z)^2+\epsilon}}$

- $R(C^n(x,y,z)))\cdot\Delta t$

De esta discretización, se desprende que estamos utilizando un stencil en forma de cruz (en 3d) Establecimos experimentalmente un valor dt de 0.01 días el cual parece garantizar que el algorítmo no tenga problemas de estabilidad.

El resto de las constantes utilizadas son las planteadas en el enunciado.

2.2. Condiciones de borde e iniciales

Si bien no prevemos correr la simulación hasta que toque los bordes del dominio de la concentración, establecimos las condiciones de borde de Neuman:

$$\nabla C, n = 0$$

es decir

$$\frac{\partial C}{\partial x} = 0$$

$$\frac{\partial C}{\partial y} = 0$$

$$\frac{\partial C}{\partial z} = 0$$

Como condición inicial, la concentración en el dominio se encuentra en 0 a excepción de una esfera de radio 15mm con concentración CM / 2

2.3. Otras consideraciones

- Consideramos que los valores negativos de concentración no tienen sentido biológico. En nuestros modelo, de producirse dichos valores serán saturados a 0.
- Los valores de concentración que resulten mayores a la concentración máxima (Cm) serán saturados a la misma.

2.4. Software utilizado

El desarrollo se realizó enteramente utilizando C++ con diversas IDEs.

Las dos versiones paralelizadas se desarrollaron utilizando OpenMP y MPI. Para graficar la matriz 3D de concentraciones utilizamos Paraview sobre archivos VTK generados por nosotros a intervalos regulares de iteraciones temporales.

El video se realizó con la herramienta mencoder sobre archivos PNG generados con Paraview.

Las mediciones de tiempo se realizaron utilizando la librería estándar Chrono de C++.

2.5. Estructuras de datos utilizadas

El problema propuesto por el trabajo práctico requiere la representación de una matriz tridimensional para alojar los resultados de las concentraciones de células tumorales calculadas así como los coeficientes de difusión y sus derivadas.

Previendo que el programa iba a tener que paralelizarse utilizando MPI, decidimos utilizar una estructura lo más sencilla posible: Representar la matriz como un vector de TAM_X x TAM_Y x TAM_Z, donde estas tres últimas son las dimensiones del paralelepípedo que representa el dominio del problema. Las posiciones x, y, z de la matriz se guardan en el siguiente orden:

```
    [(0,0,0 - TAMX-1,0,0),(0,1,0 - TAMX-1,1,0),...((0,(TAMY-1),0 -
(TAMX-1),(TAMY-1),0))],

    [(0,0,1 - TAMX-1,0,1),(0,1,1 - TAMX-1,1,1),...((0,(TAMY-1),1 -
(TAMX-1),(TAMY-1),1))],

    .

    .

    .

    [(0,0,(TAMZ-1) - TAMX-1,0,(TAMZ-1)),(0,1,(TAMZ-1) -
TAMX-1,1,(TAMZ-1)),...((0,(TAMY-1),1 - (TAMX-1),(TAMY-1),(TAMZ-1)))]
```

Figura 2: Dimensiones.

De esta manera, el elemento (i,j,k) se corresponde a la posición k * (TAM_X * TAM_Y) + j * TAM_X + i

En nuestra primera implementación habíamos utilizado una clase que funcionaba como un wrapper del vector de datos. De esta manera, las posiciones se accedían mediante setters y getters dados x,y,z. Sin embargo, nos dimos cuenta que esto suponía cierto overhead debido al context-switching del llamado a funciones de la clase. Si bien esto puede parecer un tiempo insignificante, debemos tener en cuenta que se ejecuta decenas de millones de veces por iteración. Debido a lo expuesto, decidimos utilizar el array directamente y accederlo mediante macros de la siguiente manera:

```
#define G3D(V,X,Y,Z)  V[(Z) * ((TAM_X) * (TAM_Y)) + (Y) * (TAM_X)
+ (X)]      // Hace el GET de una posición

#define S3D(V,X,Y,Z,S)  V[(Z) * ((TAM_X) * (TAM_Y)) + (Y) * TAM_X
+ (X)]=S   // Hacel el SET de una posición
```

Figura 3: Defines utilizados en la implementación.

Comparamos la performance de ambas implementaciones, obteniendo con esta última una mejora de tiempos de aproximadamente un 20

Cuando implementamos los ciclos anidados sobre las matrices, tuvimos en cuenta que siempre el ciclo externo itere sobre Z, luego el del medio sobre Y el el interno sobre X. De esta manera, los arrays se recorren secuencialmente aprovechando el principio de localidad espacial. La única matriz que varía en el tiempo es la que contiene las concentraciones que se desean calcular (C(x,y,z)). Tanto la matriz de coeficientes de difusión D como sus derivadas respecto a los tres ejes son constantes en el tiempo. Por este motivo, a expensas de utilizar más memoria decidimos precalcular las tres derivadas y guardar cada una de ellas en una matriz (vector) con el objetivo de no repetir cálculos.

2.6. Inconvenientes

- Al utilizar el método explícito tuvimos problemas de estabilidad que resultaban en parte de la masa tumoral se desplazara a los bordes del cubo que representa el dominio del problema. Estos fueron solucionados reduciendo el el delta de tiempo (dt) a un valor de 0.01 días.
- El término no lineal que involucra el logaritmo ln(Cm/C) presentaba problemas cuando el valor de C se aproximaba a 0, obteniéndose valores NaN.

  Para solucionar esto, se reemplazó la expresión por ln(CM/SQRT(C2 + epsilon)), donde epsilon es un valor positivo pequeño.

  Generalmente C $>>$ epsilon, por lo que la diferencia entre ambas expresiones se puede despreciar.
- Bajo ciertas condiciones, debido a la existencia de derivadas negativas, el término C(x,y,z) podría resultar negativo. Esto no tiene sentido biológico por lo que se decidió saturar en estos casos a 0.
- De la misma forma que en 3, todo valor de C que resulte mayor a CM será saturado a CM.
- La matriz de difusión dada por la cátedra presenta saltos bruscos en los niveles de difusión a lo largo del espacio lo que resultaba en derivadas de gran magnitud que podrían provocar problemas numéricos. Para evitar este problema se aplicó una función de blureo (promedio entre celdas vecinas) con el fin de lograr un suavizado de las derivadas. La de suavizado es la siguiente:

D_Blur(x,y,z) = (D(x,y,z) + D(x+$\Delta x, y, z) + D(x - \Delta x, y, z) + D(x, y + \Delta y, z) +$
$D(x, y - \Delta y, z) + D(x, y, z - \Delta z) + D(x, y, z + \Delta z))/8$

2.7. Paralelización utilizando OpenMP

Para paralelizar con OpenMP utilizamos las directivas #pragma correspondientes sobre el loop principal de la siguiente manera:

PONER ACA LOS PROBLEMAS QUE TUVIMOS CON OPENMPI Y COMO SE SOLUCIONARON

```
#pragma omp parallel for collapse(3)  schedule(static) private(c,d,c_plus_i,c_minus_i,c_plus_j,c_minus_j,c_plus_k,c_minus_k,i,j,k,c_new) num_threads
for (k = 1; k < TAM_Z-2; k++)
{
    for (j = 1; j < TAM_Y-2; j++)
    {
        for (i = 1; i < TAM_X-2; i++)
        {

            cálculo de una celda
```

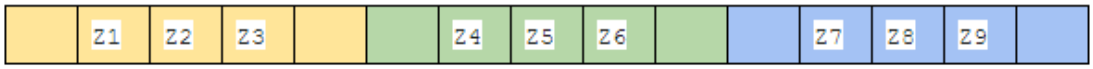
2.8. Paralelización utilizando MPI

El esquema de paralelización utilizado se basa en dividir el dominio 3D del problema en rodajas iguales(salvo el último proceso) y asignar cada una de estas a un proceso. Elegimos la división en rodajas por su simplicidad y porque solo requiere el pasaje entre procesos de las fronteras izquierda y derecha. Consideramos conveniente llevar a cabo esta división a lo largo del eje Z por dos motivos:

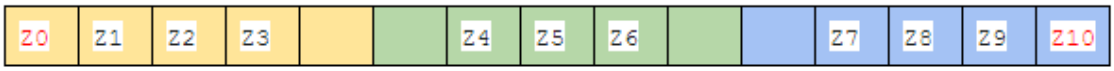
- En nuestra implementación sobre Vector¡double¿de la matriz tridimensional, todas las celdas pertenecientes a una misma coordenada Z se encuentran guardados en forma contigua y ordenada por Z. Esto simplifica enormemente la división del espacio de memoria, simplemente debe asignarse a cada proceso una porción del arreglo de C * (Tamaño_X * Tamaño_Y), donde C es la cantidad de rodajas asignadas a cada proceso.
- Al trabajar con rodajas del plano X-Y para un Z fijo, aprovechamos el principio de localidad espacial, dado que cada plano XY se encuentra guardado en forma contigua en el arreglo. Esto supone una mejora en el uso de la cache de los procesadores locales.

Veamos un ejemplo de cómo se lleva a cabo el proceso paralelizado. Definimos Zi como la rodaja que representa el plano X-Y para Z = i. De esta forma, nuestra matriz 3D se compone un arreglo de K rodajas. Supongamos K = 11, P = 3 procesos:

Z0 y Z10 son valores borde y no deben ser procesados. Esto nos deja con 9 rodajas para procesar, 3 por proceso. Sin embargo, los procesos necesitan los valores de frontera de los procesos vecinos por lo que se cada proceso aloca espacio para 5 rodajas. Luego, el proceso P0 lleva a cabo un MPI_Scatter, transmitiendo los siguientes valores a cada proceso:



Para este problema puntual, las condiciones de borde Z0 y Z10 son 0 por lo que estos datos se pueden enviar a los procesos de las puntas P0 y P2 (de hecho no es necesario dado que 0 es el default de los arrays)

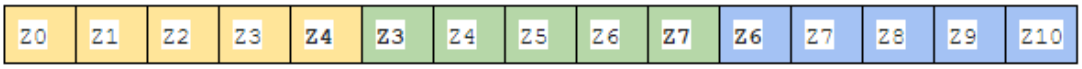


Por último, en cada iteración se ejecuta un proceso al que llamamos ”halo”que se encarga de que los procesos contiguos se intercambien los resultados de los bordes. Este procedimiento se lleva a cabo al inicio de cada iteración de la siguiente manera:

```
Shift left
P0[4] <- P1[1]
P1[4] <- P2[1]

Shift right
P1[0] <- P0[3]
P2[0] <- P1[3]
```

Resultando en el siguiente arreglo:



Con estos datos en cada proceso ya es posible realizar los cálculos locales. Cada proceso ejecuta su propia iteración. Al final de cada iteración existe una sincronización entre procesos (barrier). De esta forma nos aseguramos que el intercambio de bordes se haga con datos coherentes (no desfasados en el tiempo)

Finalmente, luego de la última iteración, utilizamos MPI_Gather para volver a componer la matriz 3D de concentraciones.

Nuestra primera versión de la función ”halo”que ejecuta cada proceso se veía de la siguiente manera:

```
if (rank > 0)
      MPI_Send(&slice[TAM_X  * TAM_Y], TAM_X  * TAM_Y, MPI_DOUBLE, ran
1, 0, MPI_COMM_WORLD);

if (rank < numProcs - 1)
      MPI_Send(&slice[(TAM_X  * TAM_Y) * (workingSliceSize - 1)], TAM_
* TAM_Y, MPI_DOUBLE, rank + 1, 0, MPI_COMM_WORLD);

MPI_Barrier(MPI_COMM_WORLD);


if (rank > 0)
      MPI_Recv(&slice[0], TAM_X  * TAM_Y, MPI_DOUBLE, rank - 1, 0,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);

if (rank < numProcs - 1)
      MPI_Recv(&slice[(TAM_X  * TAM_Y) * (workingSliceSize-1)], TAM_X
TAM_Y, MPI_DOUBLE, rank + 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

El problema de esta versión es que todos los procesos hacen MPI_SEND al principio y se quedan bloqueados esperando el ACK de los proceso de al lado antes de hacer MPI_RECEIVE ellos mismos. Bajo ciertas circunstancias esto produce deadlocks.

Para solucionar este problema, dividimos los procesos en dos grupos según la paridad de su número. Los procesos pares hacen SEND primero y RECV después y los impares al revés. Como un proceso par solo espera datos de uno impar y viceversa, el deadlock no se produce.

Nota: Cuando entregamos la primer versión de este informe nos quedaba aún solucionar un bug por el cual ciertas rodajas no se estaban calculando. Pensamos que podría tratarse de un problema de bordes, pero ocurre en una cantidad de secciones mucho menor a la cantidad de procesos. Es posible que se deba a un problema relacionado con que la dimensión $Z + 2$ no sea divisible por la cantidad de procesos. La siguiente imagen muestra este fenómeno:

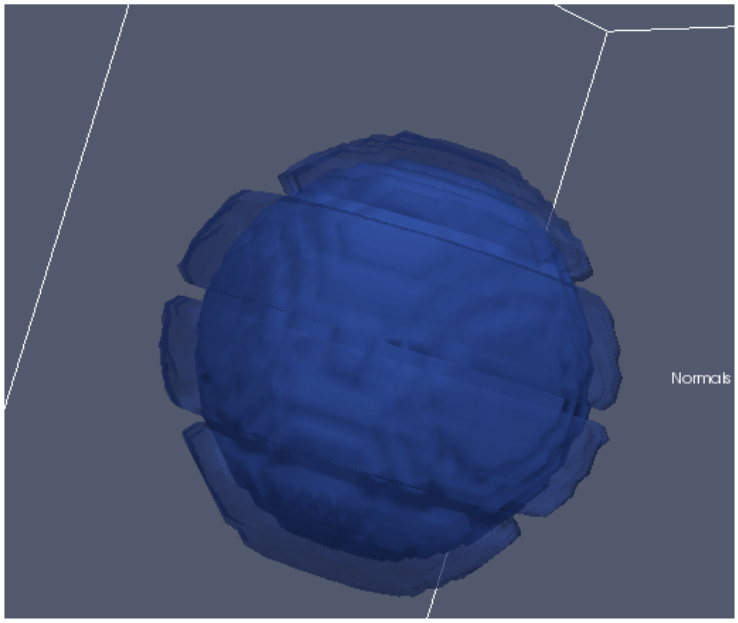


Figura 4: Defecto en la versión MPI cuya causa aún no se encontró..

La solución finalmente, fue que se descubrió que el ciclo de la coordenada Z dejaba la última posición sin procesar y eso sucedía para cada proceso, dándonos como resultado los çanales"que se generaban dándole forma de panal.

3. Resultados obtenidos

3.1. Resultados

La simulación fue realizada sobre un cubo de 90x90x122 centrado en la posición (90,95,90) del dominio definido por el archivo de difusiones provisto por la cátedra. La elección de un espacio más pequeño nos permitió alivianar el trabajo basándonos en la justificación de que no resulta práctico el análisis de tumores más grandes. Dentro del centro del cubo se plantó un tumor esférico de radio 15mm con cantidad de células uniforme CM / 2. Podemos observar como el término difusivo contribuye a la expansión de las células tumorales en una forma no uniforme. Sin embargo, el término no lineal log(CM/C) contribuye al aumento de la cantidad de células, por lo que el centro del tumor mantiene su ”densidad”. Con los defaults provistos en el TP se observa que la acción de reducción debida a radioterapia no alcanza a compensar el crecimiento del tumor.

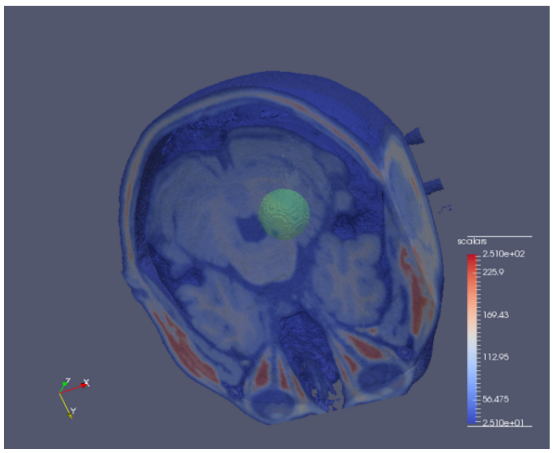


Figura 5: Tumor original.

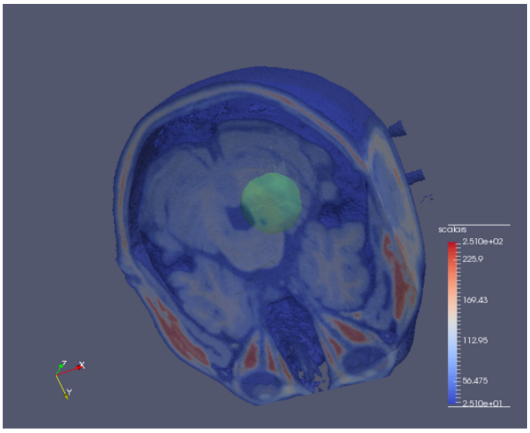


Figura 6: Tumor luego de 1000 iteraciones (10 días).

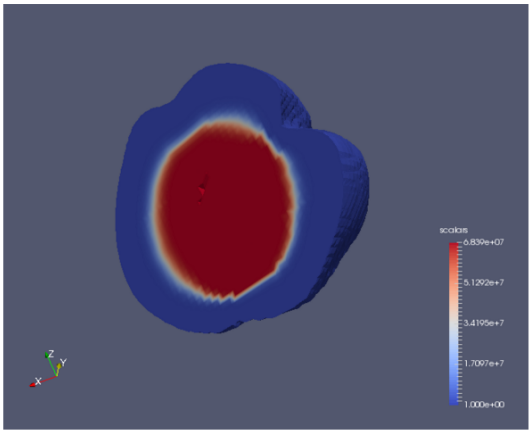


Figura 7: Densidad del tumor.

3.2. Mediciones de tiempo y comparaciones

Las tres implementaciones, serial, OpenMP y MPI difieren solamente en la paralelización de las iteraciones del loop temporal. Por tal motivo, consideramos que lo más relevante es tomar las mediciones de tiempo como lo que tarda en ejecutar dicho loop.

En primer lugar encontramos una gran diferencia de performance entre la versión serial cuyo loop principal respeta la localidad espacial (loop z,y,x) respecto a una versión con loop x,y,z. Para 100 itereaciones la versión sin esta optimmización tarda 14 segundos mientras que la optimizada tarda 5.51 segundos, es decir, una mejora cercana al 300 %. De especial interés para nosotros es determinar cómo escalan nuestras implementaciones paralelas a medida que se van agregando procesos o threads. Para las tres implementaciones, utilizamos los siguientes parámetros:

- Cantidad de iteraciones: 100
- Tamaño del dominio: 90x90x122

Todas las implementaciones fueron compiladas utilizando la opción -O3 del compilador. Los test de la implementación MPI se llevaron a cabo en una Macbook Air modelo 2011, i5 1.7Ghz doble núcleo, 4GB RAM La implementación serial optimizada tardó 5.51 segundos en completar, utilizando optimización O3. La implementación MPI presenta una gran mejora de tiempos, con un óptimo en 15 procesos, de acuerdo a la siguiente tabla:

Procesos	Tiempo 100 It (seg.)	Speed-Up
1	5.51	-
3	2.30	2.40
6	2.28	2.42
10	2.20	2.50
15	2.06	2.67
20	2.23	2.47
24	2.29	2.40
30	2.52	2.19
40	2.85	1.93

Tiempo empleado para 100 iteraciones, versión MPI

La figura 9 grafica el Speed-Up logrado respecto a la versión serial. Se utilizó una escala logarítmica para una mejor apreciación de la curva.

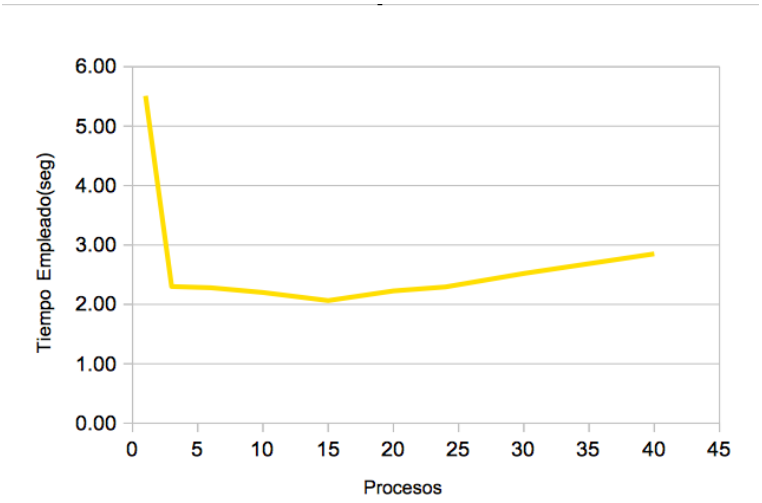


Figura 8: Tiempo de procesamiento (MPI)

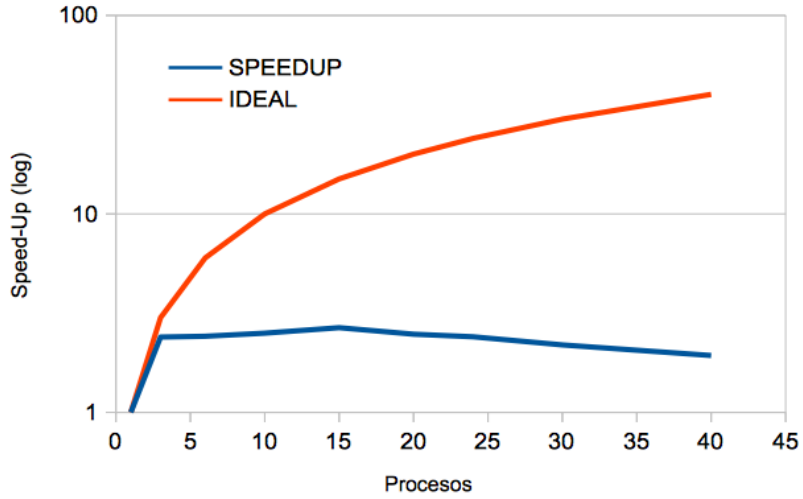


Figura 9: Speed up MPI.

Puede observarse que, a partir de los 15 proceos, seguir agregando procesos solo empeora la performance.

Tuvimos problemas para ejecutar la implementación OpenMP en OSX. Por tal motivo, la versión OpenMP se ejecutó en una PC del laboratorio. Estos nos permite realizar una análisis relativo de escalabilidad pero no comparar contra la versión MPI.

La mejora obtenida por la versión OpenMP es sostenida respecto a una recta hasta los 4 threads, luego a partir de los 5 threads, la pendiente del gráfico, como podemos observar, comienza a ”plancharse”teniendo un comportamiento asintótico

La siguiente tabla muestra los tiempos medidos. Debe tomarse en cuenta que la versión serial ejecutada en la PC del laboratorio de la facultad tardó 7.83 segundos:

Procesos	Tiempo 100 It (seg.)	Speed-Up
1	7.83	-
2	5.43	1.44
3	3.79	2.06
4	3.00	2.60
5	3.62	2.16
10	3.25	2.41
20	3.09	2.53
50	3.05	2.57
100	3.10	2.53

Tiempo empleado para 100 iteraciones, versión OpenMP

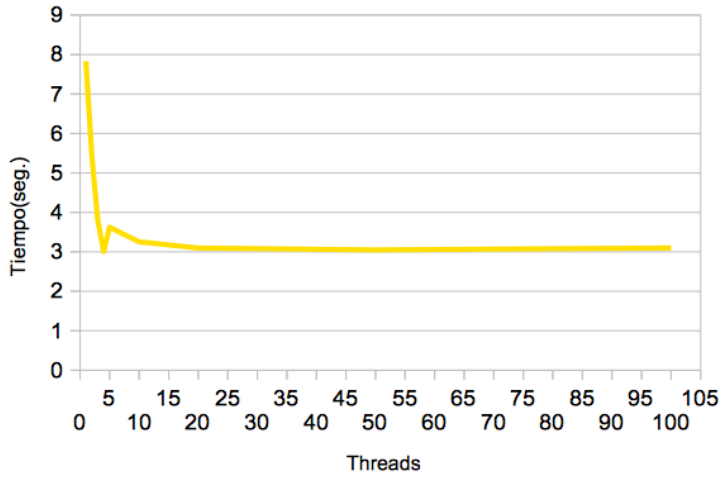


Figura 10: Tiempo de procesamiento - OpenMp.

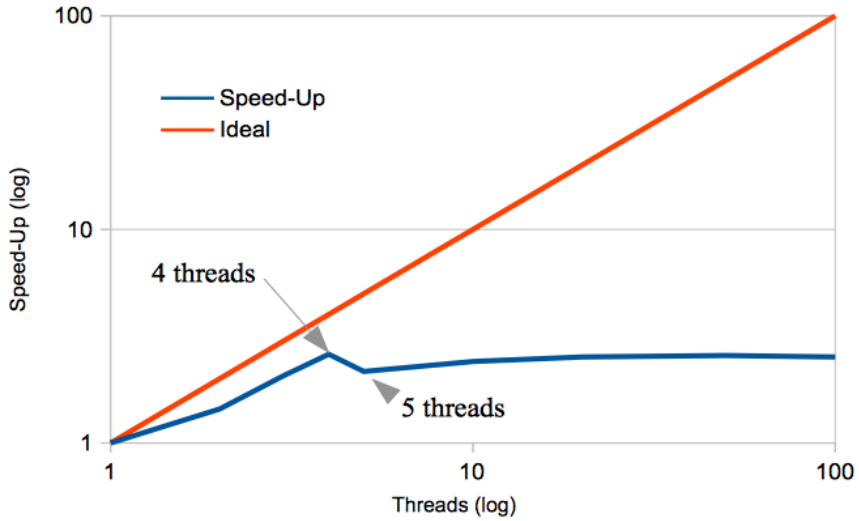


Figura 11: Speed up OpenMp.

4. Conclusiones

- La elección del método de discretización explícito parece razonable para este problema. Tanto su sencillez de implementación como el tiempo de procesamiento por iteración comparado con otros métodos incondicionalmente estables supone solo la elección de un delta t de 0.01 dias. La versión paralela puede procesar 100 iteraciones correspondientes a un día en 2 segundos en una laptop, lo cual parece una velocidad mas que razonable.
- La paralelización con MPI aumenta drásticamente la performance de la simulación. Sin embargo, para la instancia de prueba(90x90x122) no parece escalar mas hallá de los 15 procesos simultáneos. Creemos que esto es debido mayormente a que, al aumentar la cantidad de procesos, disminuye la cantidad de procesamiento propio (rodajas interiores de cada proceso) respecto al overhead (mensajes de intercambio de bordes). Por lo tanto, debe buscarse un compromiso entre la cantidad de procesos y la cantidad de rodajas asignadas a los mismos.
- Es importante prestar atención a las posibles optimizaciones de los procesos seriales. En nuestro caso, un buen uso de la propiedad de localidad espacial de la cache significó un aumento de performance comparable con el de la paralelización.