Mecánica Computacional Vol. XXIV
A. Larreteguy (Editor)
Buenos Aires, Argentina, Noviembre 2005

MEJORAS EN LA PERFOMANCE DE UN PROGRAMA PARALELO DE FISICOQUÍMICA HIDRODINÁMICA COMPUTACIONAL EN UN CLUSTER BEOWULF HETEROGÉNEO

P. Milano[*]**, E. Mocskos**[*]**y G. Marshall**[*]

[*]Laboratorio de Sistemas Complejos
Departamento de Computación, Facultad de Ciencias Exactas y Naturales (FCEyN)
Universidad de Buenos Aires, Ciudad Universitaria,
Pabellón I, (C1428EGA) Buenos Aires, Argentina.
email: marshalg@mail.retina.ar, http://www.lsc.dc.uba.ar

Palabras clave: HPC, Cluster, Perfomance, Cálculo paralelo, Beowulf, Diferencias finitas

Resumen. *En este trabajo se presenta una mejora en la performance de un programa paralelo de fisicoquímica hidrodinámica computacional en un cluster Beowulf heterogéneo bajo el sistema operativo Linux y la biblioteca de pasaje de mensajes (MPI). El programa paralelo resuelve las ecuaciones de Nernst-Planck para el transporte iónico, la ecuación de Poisson para el campo eléctrico y las ecuaciones de Navier-Stokes para el fluido en el espacio de tres dimensiones y el tiempo, mediante la utilización de un esquema de diferencias finitas fuertemente implícito, métodos de relajación standard y técnicas de descomposición por subdominios. Las mejoras en el código paralelo se obtuvieron mediante: i) balance de carga semi-dinámico, que no requiere un conocimiento a priori de las características del cluster, ii) reordenamiento del proceso iterativo, con el fin de soportar el solapamiento del tiempo de cómputo con el de comunicación, y iii) uso de directivas de comunicación no-bloqueantes, diferentes tamaños de buffer y diversas implementaciones código-abierto de MPI . Por medio de estas estrategias, se ha logrado una mejora de aproximadamente un 40 % en relación a la versión paralela sin optimizar. En algunos casos, se alcanzó una escalabilidad con eficiencia cercana a la unidad. La versión paralela del programa ha sido utilizada en el estudio de problemas complejos tridimensionales en escalas que no se habían logrado alcanzar con versiones anteriores.*

1. INTRODUCCIÓN

En esta sección se presenta brevemente el problema de fisicoquímica hidrodinámica simulado y aspectos de su simulación en máquinas paralelas. En el experimento de electrodeposición en celda delgada (ECD), la celda electrolítica consiste en dos placas planas de vidrio entre las cuales se colocan dos electrodos (que pueden ser alambres de zinc o cobre) en forma paralela separados por una solución acuosa de un electrolito, una ilustración puede verse en la figura 1.

Una diferencia de potencial aplicada entre electrodos, produce un depósito por reducción de los iones metálicos. Dependiendo de la geometría de la celda, su orientación relativa a la gravedad, la concentración de electrolítico, el potencial aplicado y otros parámetros, el depósito puede ser fractal, densamente ramificado o dendrítico. ECD se ha convertido en un modelo paradigmático para el estudio de la formación de patrones de crecimiento (GPF), es decir del crecimiento inestable de interfases.[1–25]

El crecimiento de las dendritas induce un proceso de transporte iónico muy rico y complejo, el cual es gobernado, principalmente, por difusión, migración y convección. La convección se debe a fuerzas de Coulomb provocadas por cargas eléctricas locales y a fuerzas de empuje generadas por gradientes de concentración que llevan a gradientes de densidad.

En un experimento de ECD cuando se cierra el circuito, una corriente comienza a fluir a través de la celda y una capa límite de concentración se desarrolla en las cercanías de cada electrodo. En el ánodo, la concentración se incrementa por sobre su nivel inicial debido al transporte de aniones y a la disolución de los iones metálicos desde el mismo ánodo. En el cátodo, la concentración disminuye ya que los iones metálicos se reducen y se depositan, y los aniones se alejan. Estas variaciones de concentración llevan a variaciones en la densidad y al desarrollo de rollos convectivos por acción de la gravedad.[12, 23]

Durante este período inicial, la falta de cationes en el cátodo se supone uniforme. Simultáneamente, en una capa límite muy cercana al cátodo se desarrolla una carga eléctrica local que da lugar a las fuerzas eléctricas de Coulomb.

Luego de algunos segundos, se desarrolla una inestabilidad, disparando el crecimiento del depósito en el cátodo. El depósito se desarrolla como un arreglo tridimensional de filamentos metálicos porosos. Las fuerzas de Coulomb se concentran en las puntas de los filamentos de acuerdo al modelo desarrollado por Fleury et al[9] (y citas en ese artículo). Cada filamento poroso permite que el fluido penetre por su punta y sea expulsado por sus lados, formando un vórtice toroidal gobernado por las fuerzas eléctricas.

La interacción entre los vórtices toroidales generados por las fuerzas eléctricas y el rollo catódico generado por la acción de las fuerzas gravitatorias y su presencia del depósito ramificado producen un complejo movimiento helicoidal tridimensional afectando fuertemente la morfología de crecimiento del depósito.

El transporte iónico en ECD en celda delgada, en ausencia de crecimiento, puede describirse por un modelo matemático basado en primeros principios, el cual incluye las ecuaciones de Nernst-Planck para el transporte de iones, la ecuación de Poisson para el potencial eléctrico y las ecuaciones de Navier-Stokes para el fluido. Los modelos 2D y 3D han sido presentados

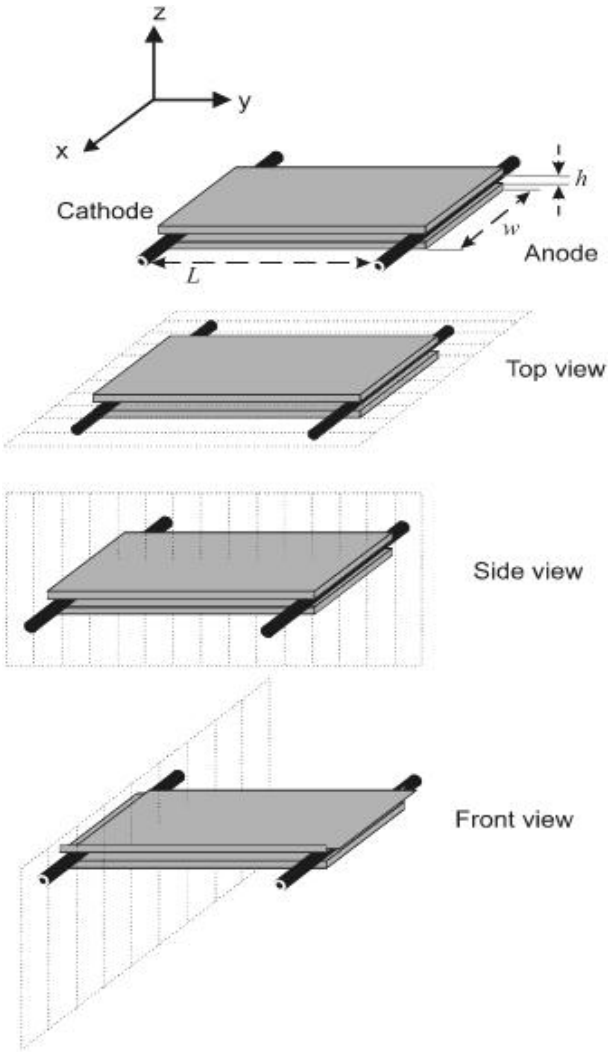

Figura 1: Setup experimental y convención de vista de celda[23]

y validados experimentalmente[18,19,23] con anterioridad. El sistema de ecuaciones adimensional puede escribirse:

$$\frac{\partial C_i}{\partial t} = -\nabla \cdot \mathbf{j}_i \tag{1}$$

$$\mathbf{j}_i = -M_i C_i \nabla \phi - \frac{1}{Pe_i} \nabla C_i + C_i \mathbf{v} \tag{2}$$

$$\nabla^2 \phi = Po \sum_i z_i C_i \tag{3}$$

$$\frac{\partial \boldsymbol{\zeta}}{\partial t} + \nabla \times (\boldsymbol{\zeta} \times \mathbf{v}) = \frac{1}{Re}\nabla^2 \boldsymbol{\zeta} + \sum_i \left[G_e z_i (\nabla\phi \times \nabla C_i) - G_{g_i} \nabla \times \left(C_i \frac{\mathbf{g}}{g} \right) \right] \tag{4}$$

$$\boldsymbol{\zeta} = -\bigtriangledown^2 \Psi \tag{5}$$

$$\mathbf{v} = \bigtriangledown \times \Psi \tag{6}$$

En donde C_i y $\mathbf{j}_i$ representan de manera adimensional la concentración y el flujo de cada especie iónica i (para un electrólito ternario como el $ZnSO_4$ /H_2SO_4, i= C, A and H, zinc, sulfato e iones de hidrógeno); $\mathbf{v}$, ϕ, P, $\mathbf{E}$, $\boldsymbol{\zeta}$ y $\boldsymbol{\Psi}$ son la velocidad del fluido, el potencial eléctrico, el campo eléctrico, la vorticidad, y el vector potencial de velocidades, respectivamente; $\mathbf{g}/g$ es el vector unitario que apunta en dirección de la gravedad.

Las cantidades $M_i = \mu_i \Phi_0 / x_0 u_0$, $Pe_i = x_0 u_0 / D_i$, $Po = x_0^2 C_0 e / \epsilon \Phi_0$, $Re = x_0 u_0 / \nu$, $Fr = u_0^2 / x_0 g$, $G_{e_i} = e C_i \Phi_0 / \rho_0 u_0^2$ y $G_{g_i} = x_0 C_i g \alpha_i / u_0^2$, corresponden a los números adimensionales de Migración, Peclet, Poisson, Reynolds, Froude, Grashof eléctrico y Grashof gravitatorio, respectivamente. Las cantidades z_i, μ_i, y D_i son, respectivamente, el número de cargas por ion, y las constantes de la movilidad y difusión de cada especie iónica i; μ_i y z_i son cantidades con signo, siendo positivas para los cationes y negativas para los aniones; g es la aceleración gravitoria; e es la carga electrónica, ϵ es la permisividad del medio y ν es la viscocidad cinética. x_0, u_0, ϕ_0, C_0 y ρ_0 son valores de referencia para la longitud, velocidad, potencial electrostático, concentración y la densidad del fluido. Para cerrar el sistema, se ha usado una aproximación del tipo Boussinesq: $\rho = \rho_o(1 + \Sigma_i \alpha_i \Delta C_i)$, donde $\alpha_i = \frac{1}{\rho_o}\frac{\partial \rho}{\partial C_i}$.

Los requerimientos de velocidad de cálculo y memoria del modelo presentado para simulaciones realísticas, en general están en el límite o superan las capacidad de una máquina serial relativamente potente; esto lleva a la utilización de cálculo paralelo. El sistema con arquitectura Beowulf que consiste en un cluster de PC's conectados a traves de un switch se ha generalizado como el más conveniente para este tipo de problemas.

Para poder tener una noción de la medida de performance de los algoritmos paralelos, se definen las métricas de *speedup*,[26] *eficiencia*, *overhead*[27] y *speedup escalado*[28] que se muestran a continuación:

speedup	*eficiencia*	*overhead*	*speedup escalado*
$S(n,p) = \frac{T_1(n)}{T_p(n,p)}$	$E(n,p) = \frac{T_1(n)}{pT_p(n,p)}$	$TO = pT_p(n,p) - T_1(n)$	$S_s(n,p) = p + (1-p)f$

en donde T_1 es el tiempo de ejecución del programa serial, T_p del paralelo con p procesadores y f es la parte serial del algoritmo.

El objetivo último de un programa de cómputo paralelo es lograr que sea escalable, es decir, que la eficiencia se mantenga constante a medida que se aumentan simultáneamente el tamaño del problema y el número de procesadores. Lo ideal es obtener programas escalables con eficiencias cercanas a la unidad.

Muchos factores intervienen en el overhead de trabajo, es decir, la diferencia entre el trabajo total que realiza un programa paralelo del que realiza un programa serial, para una misma instancia de programa. Podemos citar aquí, las comunicaciones inter-procesos, el tiempo ocioso (por desbalance de carga), y cómputos adicionales (cálculos hechos en el programa paralelo que no se hacen el serial). A continuación se analizan algunos de estos factores.

2. METODOLOGÍA

Como punto de inicio, se tomó una solución secuencial preexistente del problema de ECD en tres dimensiones ya presentado.[23] Una vez obtenida la versión paralela basada en pasaje de mensajes, se la optimizó con las técnicas que se presentan a continuación.

2.1. Balance de carga

Una de características más destacadas de los clusters tipo Beowulf es la de permitir su actualización en forma progresiva. Esto constituye una ventaja, ya que el costo de adquisición de nodos se distribuye en el tiempo, pero produce como resultado que los clusters cuenten con procesadores de capacidades de cálculo radicalmente diferentes y con distinta cantidad de memoria por nodo.

A modo de ejemplo, podemos citar la configuración del cluster *Speedy Gonzalez*,[29] que contaba con 16 nodos con la siguiente configuración en el momento de las pruebas realizadas:

- 4 nodos con procesador Pentium III 733 MHz
- 6 nodos con procesador AMD Athlon 950 Mhz
- 4 nodos con procesador AMD Athlon 1200 Mhz
- 2 nodos con procesador AMD Athlon 900 Mhz

Los algoritmos iterativos paralelos requieren que los procesos se sincronicen una vez terminada cada iteración, momento en el que se intercambian los valores con sus vecinos. Debido a esto, puede suceder que algunos procesadores más rápidos terminen antes de realizar sus cálculos locales y queden ociosos a la espera de que los nuevos datos de entrada sean generados por algún procesador vecino más lento. Si todos los procesadores, independientemente de su capacidad de cálculo, tienen igual cantidad de trabajo, entonces el cluster terminará funcionando como si todos sus procesadores tuvieran el poder de cálculo del más lento.

El siguiente ejemplo ilustra la problemática planteada: Se cuenta con un cluster de tres nodos, $\mathcal{A}$ con un poder de cálculo de 20 unidades y otros dos ($\mathcal{B}$ y $\mathcal{C}$) con poder de cálculo de 5 unidades. Un programa serial que tarda 5 unidades de tiempo corriendo en el nodo $\mathcal{A}$, tardará 20 unidades en $\mathcal{B}$ o en $\mathcal{C}$. Suponiendo que se obtuviera un algoritmo paralelo ideal (i.e. sin pérdida de tiempo en comunicación ni otras tareas de sincronización) y si se reparten los subdominios uniformemente entre los tres nodos, el algoritmo paralelo tardará $\frac{20}{3} = 6{,}66$ unidades de tiempo en finalizar (recordar que en $\mathcal{A}$, el algoritmo serial tardaba 5 unidades de tiempo).

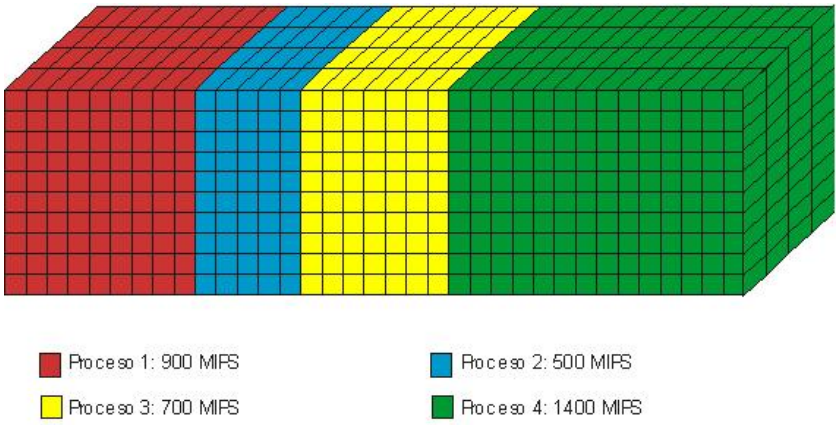

Figura 2: Resultado del balanceo de carga semi-dinámico en el ejemplo citado.

Queda en evidencia la necesidad de replantear la partición en subdominios para asignarle a cada procesador una cantidad de carga proporcional a su poder de cálculo. Para hacerlo, se adoptó un método de balance de carga semi-dinámico: La idea es poder detectar a priori, la capacidad de cálculo de cada uno de los nodos involucrados, para poder asignar a cada uno de ellos una cantidad de trabajo proporcional a su poder de cálculo.

Esta etapa, que se ha independizado de la simulación principal para lograr una mayor flexibilidad, consiste en realizar una simulación utilizando parámetros blandos (para asegurarse que la solución converja) durante una cantidad predefinida de iteraciones y sobre un espacio de memoria creado dinámicamente. La simulación es cronometrada en cada uno de los nodos, luego de lo cual, todos informan al resto el tiempo que ha tardado terminar. Este tiempo es una buena medida de su poder de cálculo y es transmitido entre todos los nodos utilizando una operación de comunicación colectiva.[27,30] Con esta información, todos los nodos pueden calcular independientemente el tamaño de subdominio que le corresponde.

Se considera a este método “semi-dinámico” puesto que, por un lado, se ajusta dinámicamente a cualquier tipo de clusters sin necesidad de conocer a priori ninguna característica de los nodos involucrados, pero por otro lado, una vez establecida la partición en subdominios, ésta permanece constante hasta la finalización del cálculo.

2.2. Solapamiento de cálculo con envío de mensajes

La solución paralela numérica de ECD en 3D (y en general todas las que usen métodos iterativos de resolución de ecuaciones diferenciales), requiere que cada vez que se termina una iteración, todos los procesos intercambien los valores de borde con sus vecinos, enviando los valores calculados localmente, y obteniendo los valores que los vecinos calcularon.

La ejecución de la siguiente iteración en cada nodo no puede continuar hasta tanto se hayan enviado y recibido los datos de sus vecinos. Aquí surgen varios aspectos que degradan la per-

formance del algoritmo paralelo:

1. **Diferencias en los tiempos de cálculo de cada nodo**: Puede suceder que, pese a haberse realizado el balanceo semi-dinámico, no todos los nodos terminen cada iteración en forma sincronizada. Entre las causas que pueden influir para que esto suceda podemos citar:

   a) *Ejecución de otros programas en el nodo*: Esto incluye a las rutinas del sistema operativo, que pueden ser diferentes en cada nodo y variar su carga en el tiempo.

   b) *Diferencias en la cantidad de memoria*: Pueden provocar que un nodo realice una mayor cantidad de operaciones de paginación, con el consiguiente detrimento de performance.

   c) *El balance de carga no resultó óptimo*: Como la rutina de balance de carga no representa de forma exacta la totalidad del proceso de cálculo, la asignación de subdominios puede no ser la óptima

   d) *La cantidad de celdas a calcular dentro de la malla no es igual en todos los procesos*: Para modelar el crecimiento en forma de dendritas,[19,23] en las simulaciones de ECD se incluyen *dedos* que van desde el cátodo (extremo izquierdo de la malla) hacia el ánodo. En las celdas ocupadas por estos *dedos* no se debe resolver las ecuaciones, por lo tanto los subprocesos que los contengan tendrán una carga menor a la supuesta por la rutina de balance, incrementando de este modo el tiempo ocioso.

   e) *Diferencias en el tiempo de envío/recepción de mensajes*: Por un lado, los procesos que se encuentran en los bordes de la malla tienen menos vecinos y por lo tanto intercambian menos mensajes. Por otro lado, el acceso al medio de comunicación (placa de red, etc.) puede diferir entre nodos, provocando que cada nodo tenga un tiempo de acceso diferente.

2. **Tiempo dedicado exclusivamente al envío y recepción de mensajes**: El tiempo de transferencia de mensajes entre cada par de procesos es proporcional al tamaño de los buffers de intercambio y se ve influenciado por el medio de comunicación (placa de red, cableado, Hub/Switch, etc.). El modelo de comunicaciones que normalmente se utiliza debido a su facilidad de programación es el de mensajes bloqueantes.[27] El cálculo se suspende durante el tiempo en el que se realizan las operaciones de comunicación, lo cual tiene un impacto negativo sobre la performance final del algoritmo. Para evaluar este problema con mayor detalle, se incluyeron cronómetros en la implementación para medir los tiempos internos. Utilizando estos cronómetros, se pudo medir el tiempo de cálculo, el tiempo empleado para el envío y recepción de mensajes, y el tiempo de espera entre procesos.

La figura 3 muestra cuán importante es el tiempo ocioso en las simulaciones de casos reales. Se realizaron corridas del algoritmo paralelo con 3 tamaños de malla diferentes ($40 \times 100 \times 40$, $40 \times 200 \times 40$ y $40 \times 400 \times 40$), cada una de ellas con 2, 4, 6, 8, 10, 12 y 14 procesos. Los

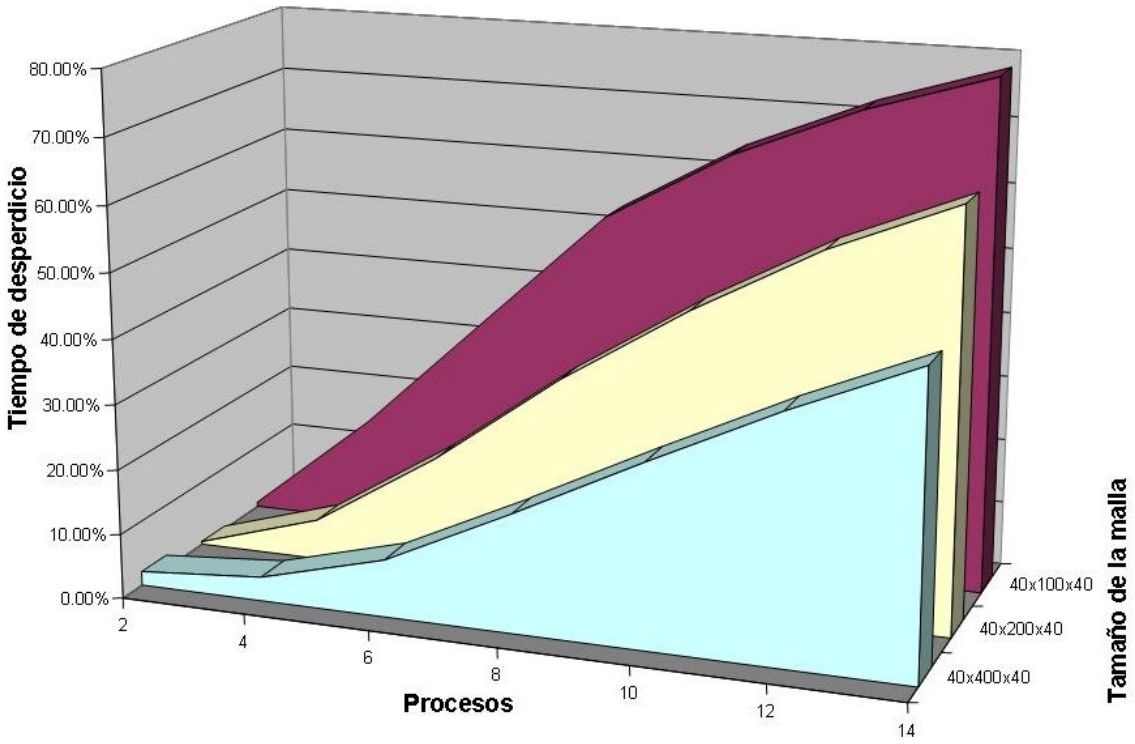

Figura 3: Porcentaje de tiempos de ejecución utilizados para comunicación de datos. *Tiempo de desperdicio* refiere al tiempo no usado en realizar cálculos. Se realizó la medición utilizando tres tamaños de problemas distintos: es claro que a mayor tamaño de problema, menor es el tiempo de desperdicio.

resultados mostraron que los tiempos de espera entre procesos junto con los de envío y recepción de mensajes son realmente considerables, llegando a ser de más del 78 % en el caso de mayor cantidad de procesos con menor tamaño de malla. También es de notar que a medida que se aumenta la cantidad de procesos utilizados para resolver un problema de tamaño fijo, aumenta también el porcentaje de tiempo utilizado para la comunicación, puesto que cada proceso tiene una malla menor para calcular, y la misma cantidad de datos para intercambiar. Esto último se aprecia al ver la reducción en los porcentajes de tiempo ocioso a medida que se amplía la malla de cálculo, sobre el eje elegido para división de dominios.

Puesto que este fenómeno es el mayor responsable de la pérdida de performance en este tipo de algoritmos paralelos, surge la necesidad de rediseñar la estrategia de comunicación con el objetivo de permitir la superposición de tiempos de cálculo con tiempos de comunicación y reducir, así, los tiempos de espera y sincronización entre procesos.

En primer lugar, se alteró el orden lexicográfico de cálculo, para calcular primero los valores de borde, y luego los valores restantes. A diferencia del diseño inicial, el cual se muestra en la figura 4 para los primeros tres procesos involucrados, no se comienza a calcular directamente desde el primer plano, sino que se calculan primero los últimos $\mathcal{N}$ planos de intercambio, donde $\mathcal{N}$ es un parámetro que se elije de acuerdo al problema a paralelizar.

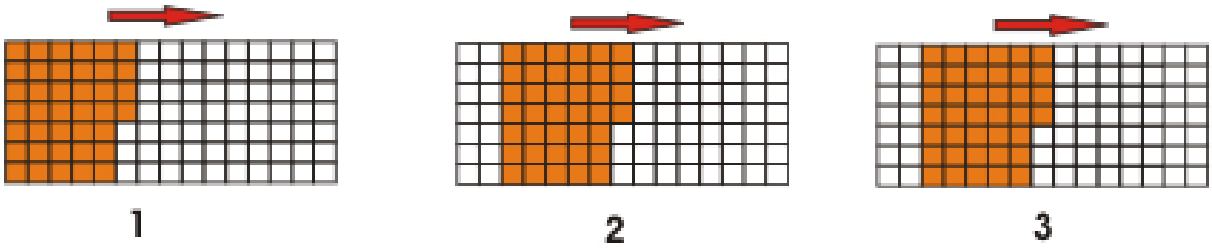

Figura 4: Ejemplo de órden de cálculo lexicográfico en los distintos procesos del algoritmo paralelo sin optimizar llevado a dos dimensiones. Las columnas no marcadas a la izquierda de los dominios de los procesos 2 y 3 corresponden a los *nodos fantasma* o *planos de intercambio* utilizados para sincronizar con los procesos vecinos.

De esta forma, luego de calcular los $\mathcal{N}$ planos, ya se dispone de los valores necesarios para enviar a los vecinos. La figura 5 muestra un esquema de la modificación para una ejecución con tres procesos involucrados. La parte (1) corresponde al cálculo de los primeros planos de intercambio, en (2) se muestra el estado luego de calcular los planos de intercambio faltantes. A partir de este momento (3) se puede lanzar la comunicación no bloqueante con los vecinos, que se superpone con el cálculo del resto de los valores internos (4).

La segunda modificación consiste en adelantar el punto de intercambio de mensajes para que sea realizado apenas están disponibles los valores de borde, es decir los planos de intercambio. Una vez realizada dicha comunicación, se continúa con el cálculo de las columnas restantes. Como es de esperar que el tamaño de la malla sea grande, al momento de realizar el intercambio de planos de cada proceso con sus vecinos, aún restará realizar la mayor parte de cálculo local. De este modo utilizando comunicaciones no bloqueantes se logra solapar el tiempo de transferencia de datos con parte del tiempo de cálculo de las columnas restantes, siempre y cuando las bibliotecas de funciones, el hardware y el sistema operativo lo permitan.

Al terminar de calcular todos los planos, será necesario esperar a que todas las comunicaciones finalicen (si es que aún no lo habían hecho), para poder entonces continuar con la siguiente iteración.

3. RESULTADOS Y DISCUSIÓN

3.1. Diferencias entre programa serial y paralelo

Debido a que la paralelización y posterior optimización del programa introdujo algunos cambios en la forma de resolver el problema, lo primero que se realizó es una comprobación de las diferencias. Para esto se adoptó la siguiente metodología: se ejecutó el algoritmo serial y el paralelo con 2, 4 y 8 procesos con tres tamaños diferentes de malla, en los que se varió la dimensión principal del experimento (eje y, ver figura 1). Luego se determinó el porcentaje de variación punto a punto entre cada resultado obtenido de la versión paralela y la serial correspondiente.

Como valor representativo de las diferencias se tomó el promedio, en la tabla 1 se muestran varios de los campos calculados.

Los resultados de las comparaciones, muestran que: Por un lado, dado un tamaño de malla

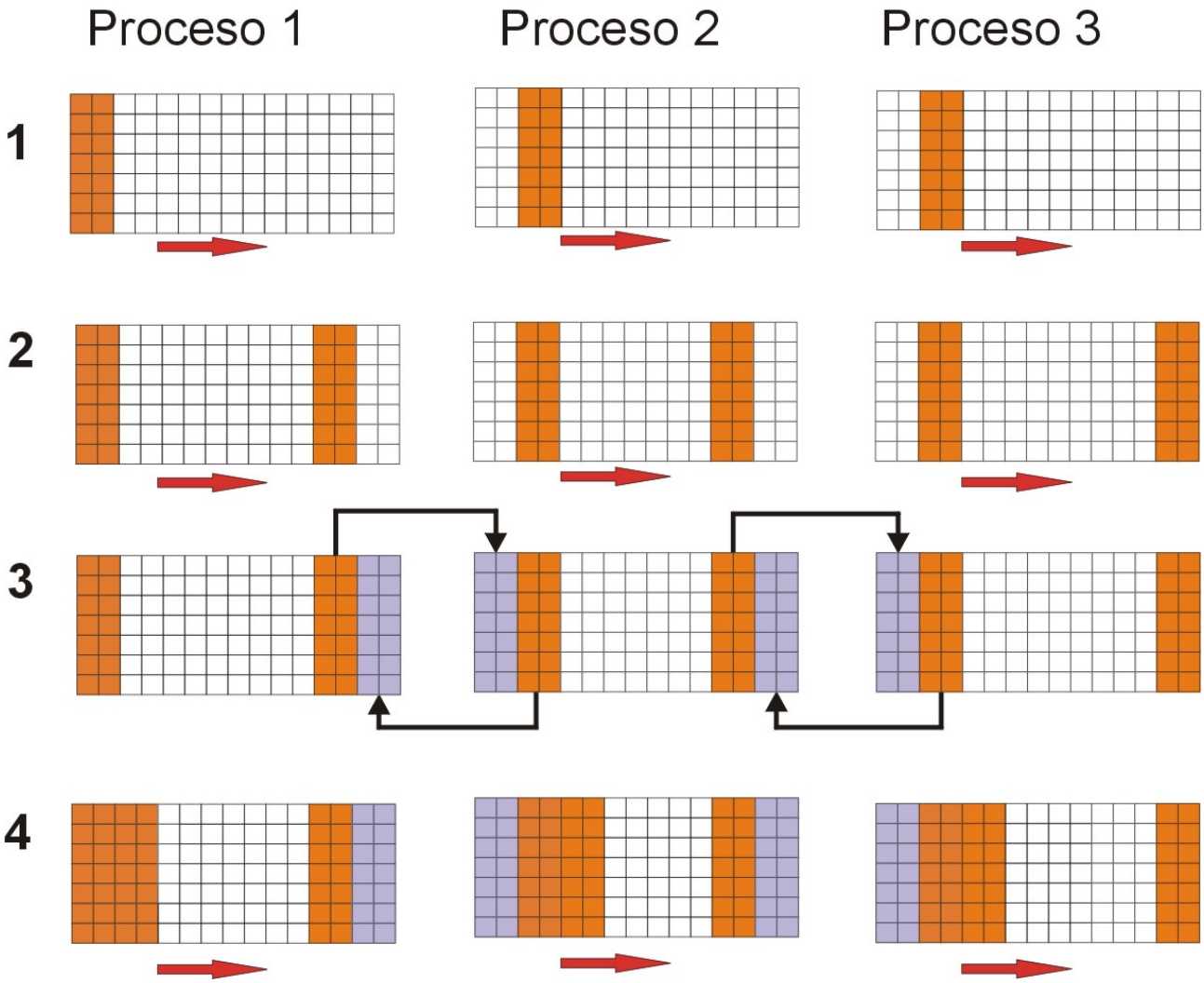

Figura 5: Esquema de la modificación al cálculo lexicográfico en una ejecución con tres procesos. La parte (1) corresponde a los planos de intercambio de la izquierda, la (2) a los de la derecha, la (3) al lanzamiento de la comunicación no bloqueante; y la (4) a la continuación del cálculo de los valores internos de la malla.

fijo, a medida que se aumenta la cantidad de procesadores, el porcentaje de variación también aumenta. Esto, creemos que se debe a la mayor cantidad de *gaps* en el cálculo de la malla dentro de cada iteración. Del mismo modo, se observa que, para los casos de tamaño de malla más pequeño, con la mayor cantidad de procesadores las variaciones son realmente grandes (35 %, 74 %, etc.). Estos valores decrecen muy notablemente con el aumento del tamaño de la malla sobre el eje y, el cual es la dimensión principal del experimento y sobre la que se realiza la división de dominios. Este resultado muestra que una ejecución en paralelo puede dar resultados diferentes a la ejecución serial si no se asigna un tamaño mínimo de subdominio a cada proceso.

Aniones	Procesos		
Dimensión	2	4	8
32	$0{,}004\,\%$	$0{,}130\,\%$	$0{,}462\,\%$
96	$0{,}000\,\%$	$0{,}005\,\%$	$0{,}094\,\%$
128	$0{,}000\,\%$	$0{,}002\,\%$	$0{,}017\,\%$

Potencial	Procesos		
Dimensión	2	4	8
32	0.564 %	8.930 %	32.646 %
96	0.032 %	0.077 %	0.796 %
128	0.000 %	0.000 %	0.005 %

Psi	Procesos		
Dimensión	2	4	8
32	$1{,}026\,\%$	$6{,}581\,\%$	$74{,}188\,\%$
96	$0{,}702\,\%$	$1{,}654\,\%$	$3{,}586\,\%$
128	$0{,}230\,\%$	$0{,}494\,\%$	$1{,}428\,\%$

Velocidad	Procesos		
Dimensión	2	4	8
32	0.841 %	10.740 %	35.823 %
96	1.808 %	3.526 %	5.111 %
128	1.104 %	1.348 %	2.421 %

Cuadro 1: Porcentaje promedio de diferencias entre el programa serial y el paralelo para diferentes tamaños de malla y cantidad de procesos y campos (concentración de aniones, potencial eléctrico, vector potencial de velocidades, vector velocidad).

Para tamaños de malla lo suficientemente grandes, los porcentajes de variación son pequeños, lo cual demuestra que los resultados de una ejecución en paralelo se aproximan mejor a la serial siempre que se mantenga una proporción entre el tamaño del dominio y la cantidad de procesos involucrados.

Los casos anteriores contemplan una visión *global* de las diferencias de resultados entre las corridas serial y paralela. Resulta interesante poder ver puntualmente cuáles son las zonas dentro de la malla de cálculo en donde se observan las mayores diferencias. También resulta interesante observar el comportamiento de estas diferencias a medida que el cálculo va avanzando. A modo de ejemplo, se tomó la comparación entre un resultado serial y uno paralelo usando 3 procesos. Para graficar los resultados se utilizó el paquete gráfico *Vis5d*.[31]

La figura 6 muestra las diferencias entre las corridas serial y paralela del módulo de Ψ (vector potencial de velocidades) en 4 pasos de tiempo sucesivos. Se ve claramente que las diferencias más significativas están en el plano de intercambio entre los vecinos, para luego ir disminuyendo hacia el interior del dominio de cada subproceso. Nuevamente, los *gaps* son los responsables de estas diferencias, motivo por el cual, como se mostró anteriormente, para un tamaño de malla fijo, las diferencias son mayores al aumentar la cantidad de procesos que intervienen en el cálculo. Es interesante que en este caso, el módulo de la diferencia se mantiene aproximadamente constante a lo largo de los pasos de tiempo mostrados. Esta diferencia es inherente a la paralelización por división en subdominios.

La figura 7 corresponde al campo A (concentración de aniones) en cinco pasos de tiempo sucesivos. Aquí se muestra un ejemplo en el que se han incluido en forma artificial depósitos electrolíticos en forma de tres dedos. En estos gráficos se puede apreciar que las principales diferencias se encuentran en las zonas cercanas a la interfase de los dedos, esto se debe a que es la zona en donde el gradiente es mayor, es decir, que se producen los mayores cambios en cada paso de tiempo. A diferencia del ejemplo anterior, se observa en este caso que a medida que avanza el tiempo las diferencias se van atenuando (desaparecen las zonas de color rojo y

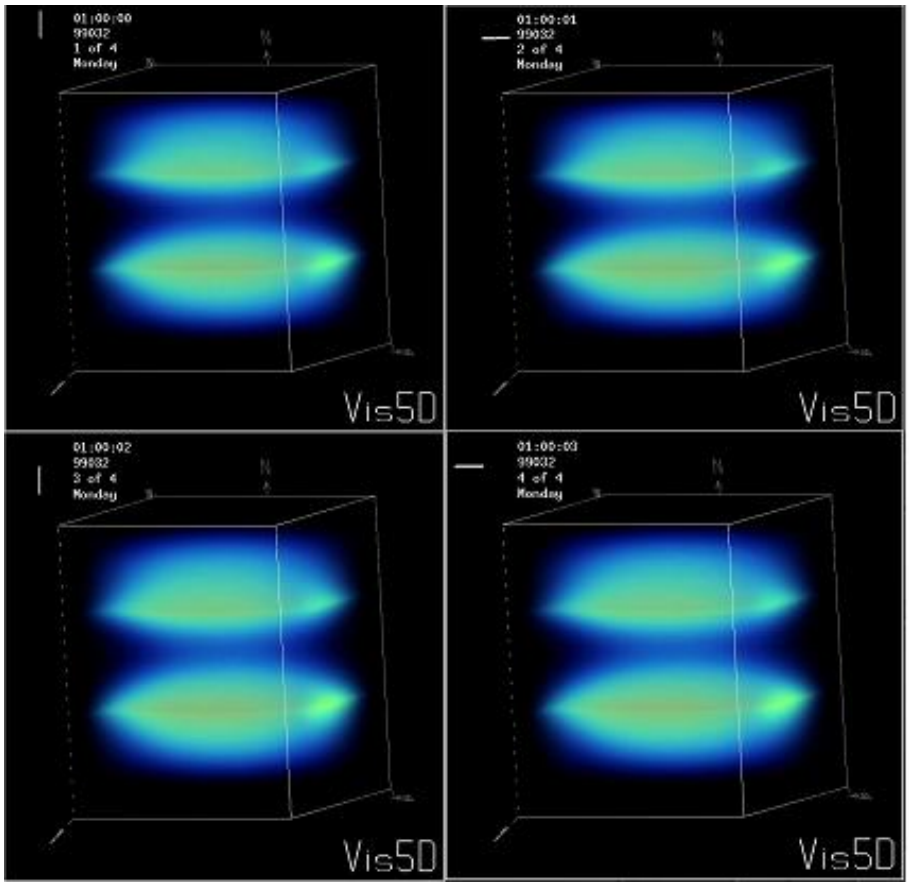

Figura 6: Localización de las diferencias entre la ejecución serial y la paralela para tres procesos. Se gráfica el módulo del vector potencial de velocidades (Ψ).

aparecen zonas verdes y azules).

3.2. Diferencias entre programas paralelos con y sin orden lexicográfico

Como se explicó anteriormente en la sección 2.2, al calcular primero los valores correspondientes a los planos de intercambio en cada nodo, se podría mejorar la eficiencia del cálculo, pero al usar esta estrategia se introducen *gaps* adicionales en cada subdominio, ya que se calcula primero la parte final del intervalo cuando aún no se calcularon los valores previos. Por este motivo, es de esperar que este cambio impacte en la cantidad de iteraciones que el algoritmo necesita para converger.

Para verificar la medida de este impacto, se realizaron comparaciones entre corridas con orden lexicográfico y no-lexicográfico. Estas comparaciones consisten en identificar, para una cantidad determinada de pasos de tiempo (en este caso, los primeros 5) la cantidad de iteraciones necesarias para avanzar al siguiente paso. Una vez obtenidos estos valores, se sumaron y se calculó el porcentaje de variación.

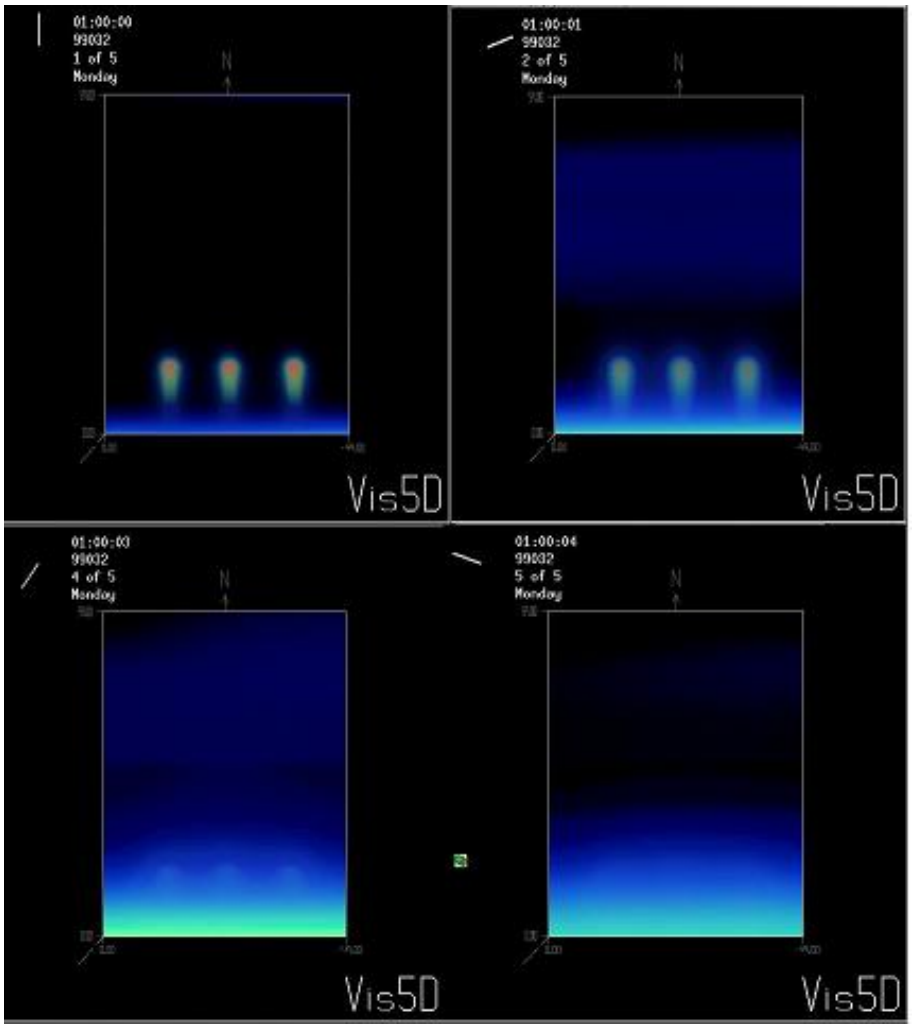

Figura 7: Localización de las diferencias entre la ejecución serial y la paralela para tres procesos. Se gráfica la concentración de aniones(A).

En las tablas 2, 3 y 4 se ven los resultados de las distintas ejecuciones del programa paralelo con y sin orden lexicográfico. Como era de esperar, las correspondientes a las que no poseen orden no-lexicográfico utilizaron algunas iteraciones más, pero la diferencia es muy pequeña. Esto motiva la utilización de este método en conjunción con comunicaciones no bloqueantes, (en entornos de SO y Hardware que lo permitan), para aumentar la performance.

Aquí también se observa que los casos con tamaños de malla más grande en el eje de partición (eje y, ver figura 1) son los que tienen menor diferencia con respecto al orden lexicográfico. La razón de este comportamiento es la menor proporción entre *gaps* e intervalos de cálculo.

DIM=50	2 procesos		4 procesos		8 procesos	
Paso	Lex	No Lex	Lex	No Lex	Lex	No Lex
1	2548	2553	2562	2595	2588	2669
2	1719	1722	1730	1755	1749	1812
3	892	899	901	913	903	950
4	810	826	802	825	813	855
5	655	669	596	612	614	641
Tot	6624	6669	6591	6700	6667	6927
Dif	0.68 %		1.65 %		3.90 %	

Cuadro 2: Diferencias entre distintas ejecuciones de programas paralelos con y sin orden lexicográfico. Se varia la cantidad de procesadores y la longitud principal del experimento, en este caso se usan 100 puntos.

DIM=100	2 procesos		4 procesos		8 procesos	
Paso	Lex	No Lex	Lex	No Lex	Lex	No Lex
1	2587	2587	2585	2586	2567	2580
2	2027	2028	2023	2026	2005	2020
3	563	564	560	562	557	560
4	428	428	427	428	279	286
5	414	414	416	420	104	104
Tot	6019	6021	6011	6022	5512	5550
Dif	0.03 %		0.18 %		0.69 %	

Cuadro 3: Similar a la tabla 2, pero usando 100 puntos sobre el eje principal.

DIM=200	2 procesos		4 procesos		8 procesos	
Paso	Lex	No Lex	Lex	No Lex	Lex	No Lex
1	2977	2980	2977	2981	2992	3011
2	2458	2460	2461	2461	2470	2489
3	1151	1149	1148	1148	1124	1131
4	766	765	753	755	695	716
5	807	806	509	516	501	510
Tot	8159	8160	7848	7861	7782	7857
Dif	0.01 %		0.17 %		0.96 %	

Cuadro 4: Similar a la tabla 2, pero usando 200 puntos sobre el eje principal.

3.3. Problemas en comunicaciones no bloqueantes

Los primeros resultados obtenidos mediante el uso del solapamiento entre cálculo y comunicaciones no bloqueantes (ver sección 2.2) fueron desalentadores. La performance de las simula-

ciones con comunicaciones no bloqueantes, dependía del tamaño de los buffers de intercambio, resultando notablemente peor que la performance de la misma simulación con comunicaciones bloqueantes. El problema se debía al manejo de las comunicaciones no bloqueantes por parte de las librerías, tanto de MPICH[32,33] como de LAM-MPI.[34] Ambas implementaciones de MPI son *single threaded* y funcionan de la siguiente manera:

Cuando se inicia un envío no bloqueante, estas implementaciones de MPI llenan los buffers TCP y derivan la tarea de envío/recepción a las capas inferiores del sistema operativo, a partir de lo cual el control retorna al programa. Mientras el programa continua ejecutándose, los datos en el buffer se transmiten, pero cuando este buffer se vacía, debido a que no hay un *thread* encargado de monitorearlo, no se vuelve a llenar. Esto tiene como resultado que todas las comunicaciones queden *en suspenso* y se completen recién en el punto de sincronización entre procesos, antes de pasar a la siguiente iteración. Resulta entonces que no se aprovecha la posibilidad de solapamiento entre los tiempos de ejecución y de comunicación y en vez de mejorar la performance, se empeora.

Es por este motivo que se abordaron dos alternativas:

- La primera alternativa consistió en obtener una licencia temporal de una implementación comercial de MPI denominada SCALI,[35] que emula threads sobre Linux y permite el progreso de las comunicaciones no bloqueantes en segundo plano.

- La segunda alternativa consistió en investigar una manera de *solucionar* el problema del llenado de buffers en MPICH.

Esta solución consistió en lo siguiente: Periódicamente, dentro del ciclo de cálculo, se realizaron llamadas a una función de testeo de MPI (MPI_Test) con la única finalidad de pasar el *control* del programa a la rutina de MPI . Con esto, se le da la oportunidad a MPI a que vuelva a llenar los buffers vacíos de TCP , para darle continuidad a la transferencia en segundo plano.

El problema, entonces, se trasladó a determinar la cantidad óptima de llamadas a MPI_Test dentro del ciclo, puesto que demasiadas llamadas generarían un overhead innecesario, mientras que muy pocas no darían tiempo de mantener los buffers de TCP llenos.

Adicionalmente, la cantidad óptima de llamadas a MPI_Test, varía según el tamaño del problema (y por ende de los buffers de intercambio). Por esto, fue necesario encontrar una manera óptima, pero que se ajuste dinámicamente, para realizar las llamadas a MPI_Test.

Luego de diversas pruebas, se encontró que la mejor forma de realizar esto fue la siguiente: Como el tamaño de los buffers de intercambio, es proporcional al eje x y al eje z (ver figura 1), pero no al eje y, se alteró el orden del ciclo para que la iteración sobre el eje y sea la más interna de las tres. De esta forma, se puede insertar la llamada a MPI_Test en el segundo nivel de anidación del ciclo, logrando que se realice $X \times Z$ veces, donde X y Z son las dimensiones correspondientes al eje x y al z respectivamente.

El pseudocódigo siguiente muestra la manera en que se implementó el arreglo para permitir en MPICH el progreso de las comunicaciones no bloqueantes en segundo plano:

```
Para i desde 1 hasta X
```

```
    Para j desde 1 hasta Z
        MPI_Test(...)
            Para k desde 1 hasta Y
                  Calcular()
            Fin para
    Fin para
Fin para
```

Este método demostró ser el más adecuado, puesto que realiza una cantidad de llamadas a MPI_Test proporcional al tamaño de los buffers de intercambio, para cualquier tamaño de dominio de la simulación. Además, las llamadas están lo suficientemente espaciadas como para permitir el envío de datos en segundo plano entre cada una de ellas.

La aplicación de las llamadas a MPI_Test dentro del ciclo de cálculo, mejoró la performance de la aplicación con comunicaciones no bloqueantes, pero no logró que dicha performance sea mejor que la obtenida mediante la implementación de comunicaciones bloqueantes, como era de esperar. Este segundo escollo, puso de manifiesto que aún restaba un problema por solucionar, puesto que la teoría indica que las comunicaciones no bloqueantes debían permitir valores de performance muy superiores.

Para avanzar sobre este punto, se realizaron mediciones de los tiempos de cada operación de envío y recepción, tanto para las comunicaciones bloqueantes (MPI_Send y MPI_Recv) como para las no bloqueantes (MPI_Isend y MPI_Irecv), realizando un aumento paulatino del tamaño del buffer de intercambio.

	Operación			
Buffer	MPI_Send	MPI_Recv	MPI_Isend	MPI_Irecv
21kb	0.200	3.800	1.300	0.005
42kb	2.200	5.500	3.000	0.006
84kb	5.200	21.000	7.300	0.008
168kb	13.000	35.000	0.011	0.004
336kb	39.400	48.200	0.030	0.025
672kb	74.000	86.300	0.090	0.025
1.31mb	122.000	145.000	0.080	0.025

Cuadro 5: Medición te tiempos de operaciones de envío/recepción bloqueantes y no bloqueantes para distintos tamaños de buffer de intercambio.

Los resultados obtenidos en milisegundos se pueden ver en la tabla 5. Aunque dicha operación debería tardar un tiempo fijo correspondiente al lanzamiento de la operación, la misma se comporta de manera muy extraña, tardando para tamaños de buffer menores a los 100k más tiempo que la totalidad de una operación no bloqueante, algo completamente inesperado. Al aumentar el tamaño de buffer a más de 168kb, los tiempos de lanzamiento de la operación

pasan a ser los esperados (recordar que la transferencia de datos propiamente dicha se realiza en segundo plano y no está cronometrada).

A partir de estos resultados, se revisó el algoritmo paralelo. En el mismo, existía un buffer de intercambio por cada una de las 16 matrices. Esto se reemplazó por un buffer único, en donde los datos de cada matriz se diferencian por el offset dentro del mismo. Al aumentar el tamaño del buffer 16 veces, se permite que las comunicaciones no bloqueantes se comporten de manera correcta. Esta última modificación, permitió finalmente obtener valores de performance mucho mejores para el caso de la utilización de comunicaciones no bloqueantes, que son explicadas con mayor detalle en el capítulo 3.5.

3.4. Comparación de implementaciones de MPI

Una vez solucionados los problemas inherentes a los tiempos de la comunicación no bloqueante, restaba entonces elegir la implementación de MPI más conveniente entre las disponibles en el cluster *Speedy*. Las implementaciones consideradas fueron:

- MPICH (Gratuita y Open Source)
- LAM-MPI (Gratuita y Open Source)
- SCALI (Comercial)

Para decidir cuál de ellas era la más conveniente, se ejecutaron seis corridas de una simulación con cinco procesos, utilizando ambas implementaciones empleando tanto comunicaciones bloqueantes como no bloqueantes. En el caso de MPICH y LAM-MPI con comunicaciones no bloqueantes se utilizó el parche descripto anteriormente. Los resultados se pueden ver en la tabla 6, en la que el tiempo está expresado en minutos.

	Scali	MPICH	LAM-MPI
Bloqueante	84.15	83.80	84.97
No Bloqueante	56.70	50.30	52.05

Cuadro 6: Comparación entre las distintas implementaciones de MPI para un problema tipo. Los tiempos están expresados en minutos.

A la vista de estos resultados, se decidió utilizar la implementación MPICH, tanto para el caso de comunicaciones bloqueantes como no bloqueantes, utilizando en este último el parche descripto anteriormente.

3.5. Medición de tiempos, perfomance y speedup

Para tener una base sobre la cual calcular el speedup y la eficiencia del algoritmo, se realizaron 24 ejecuciones de test con tres tamaños de malla diferentes con 2, 4 , 6, 8, 10, 12 y

14 procesadores. Se efectuó también la misma ejecución usando el algoritmo serial. Los tiempos empleados en la ejecución de estas corridas se muestran en la tabla 7, en la que el tamaño está expresado en cantidad de elementos por matriz, y los tiempos están expresados en minutos.

Tamaño	Procesos	1	2	4	6	8	10	12	14
40x100x40	Bloq	345.73	174.00	103.40	88.35	91.94	96.44	104.00	114.40
	Nobloq	345.73	176.52	96.40	74.20	59.59	53.53	47.50	46.66
40x200x40	Bloq	606.55	304.60	161.80	123.90	112.70	110.40	114.30	120.20
	Nobloq	606.55	308.20	152.40	101.80	78.77	65.20	59.61	57.87
40x400x40	Bloq	974.65	498.20	253.00	178.20	149.00	135.60	130.20	129.00
	Nobloq	974.65	502.30	247.50	164.10	122.90	106.10	91.51	82.02

Cuadro 7: Tiempos de corridas para diferentes tamaños y cantidad de procesos. El tamaño está expresado en cantidad de elementos por matriz, y los tiempos están expresados en minutos.

Se puede ver que dado un tamaño de malla fijo y partiendo del tiempo registrado para la ejecución serial, puede verse que la reducción en el tiempo de cálculo es cada vez menos marcada a medida que se aumenta la cantidad de procesos involucrados. Inclusive se llega a invertir la tendencia y a aumentar los tiempos de cálculo si la cantidad de procesos es demasiado grande. Esto resulta más notorio en las corridas con tamaño de malla menor y en los casos de comunicación bloqueante. Se ve entonces que, siempre para un tamaño de malla fijo, hay un límite en la cantidad de procesadores que se pueden utilizar para lograr un incremento en la performance, tal cual predice la teoría.

En cuanto a los resultados con comunicaciones no bloqueantes, se puede apreciar que la mejora obtenida en los tiempos de ejecución, es notable con respecto a los resultados con comunicaciones bloqueantes. Inclusive, se pueden ver mejoras en los tiempos en todos los casos testeados, algo que con comunicaciones bloqueantes no sucede.

Tamaño	Procesos	1	2	4	6	8	10	12	14
40x100x40	Bloq	1.00	1.99	3.34	3.91	3.76	3.58	3.32	3.02
	Nobloq	1.00	1.96	3.59	4.66	5.80	6.46	7.28	7.41
40x200x40	Bloq	1.00	1.99	3.75	4.90	5.38	5.49	5.31	5.05
	Nobloq	1.00	1.97	3.98	5.96	7.70	9.30	10.18	10.48
40x400x40	Bloq	1.00	1.96	3.85	5.47	6.54	7.19	7.49	7.56
	Nobloq	1.00	1.94	3.94	5.94	7.93	9.19	10.65	11.88

Cuadro 8: Speedup obtenidos para diferentes tamaños y cantidad de procesos.

La tabla 8 muestra los speedup obtenidos para diferentes tamaños y cantidad de procesos, los mismos resultados se grafican en las figuras 8 y 9.

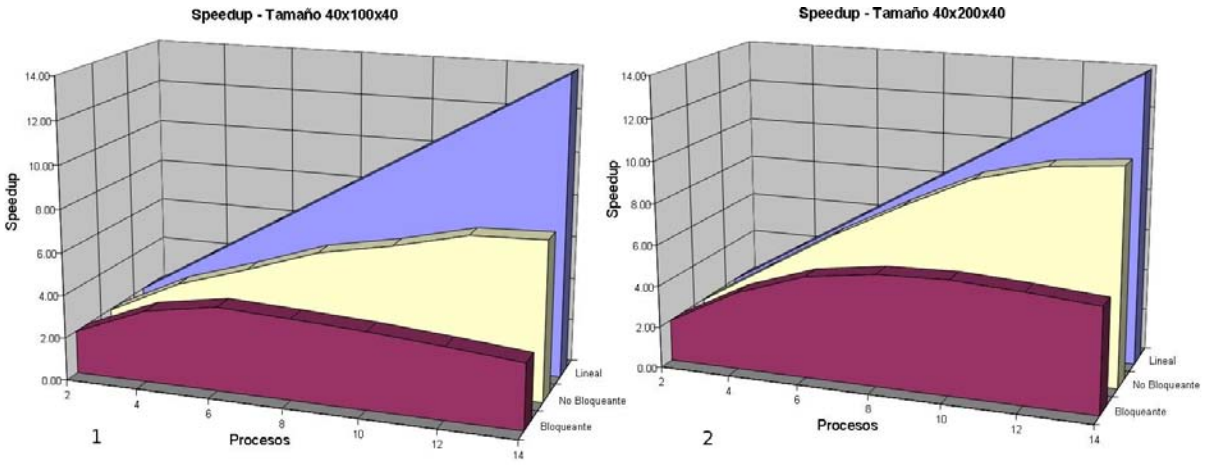

Figura 8: Speedup en una malla de (1) $40 \times 100 \times 40$ y (2) $40 \times 200 \times 40$. Los valores graficados se muestran en la tabla 8.

Se puede apreciar con claridad cómo el speedup se incrementa, para los casos bloqueantes, hasta los seis procesadores, para luego disminuir en forma casi lineal. Por el contrario, en los casos no bloqueantes el speedup se incrementa de forma mucho más marcada, lo cual demuestra el mejor aprovechamiento de los tiempos de cálculo, al solapar las operaciones de envío y recepción de mensajes. Se ve también que cuando la malla es pequeña, aún en el caso no bloqueante, el speedup está muy por debajo del máximo teórico (speedup lineal).

Al aumentar el tamaño de la malla a $50 \times 200 \times 50$ elementos, se aprecia un incremento en el speedup en general. En el caso bloqueante se nota el incremento de speedup hasta los 10 procesos y el caso no bloqueante se acerca bastante más al speedup lineal.

Por último, las corridas con tamaño de malla de $40 \times 400 \times 40$ tienen los mejores valores de speedup. El gráfico de la corrida no bloqueante se acerca aún más al speedup lineal.

La segunda medición de relevancia es la eficiencia del algoritmo paralelo para las mismas ejecuciones del caso anterior. Los resultados obtenidos se muestran en la tabla 9.

Tamaño	Procesos	1	2	4	6	8	10	12	14
40x100x40	Bloq	1.00	0.99	0.84	0.65	0.47	0.36	0.28	0.22
	Nobloq	1.00	0.98	0.90	0.78	0.73	0.65	0.61	0.53
40x200x40	Bloq	1.00	1.00	0.94	0.82	0.67	0.55	0.44	0.36
	Nobloq	1.00	0.98	0.99	0.99	0.96	0.92	0.85	0.75
40x400x40	Bloq	1.00	0.98	0.96	0.91	0.82	0.72	0.62	0.54
	Nobloq	1.00	0.97	0.98	0.99	0.99	0.93	0.89	0.85

Cuadro 9: Valores de eficiencia obtenidos, calculados en base a las mediciones de la tabla 7.

Se puede observar que la eficiencia es mayor cuando hay menos procesadores y cuando el tamaño de malla es mayor, lo cual es consistente con lo observado anteriormente. El peor de

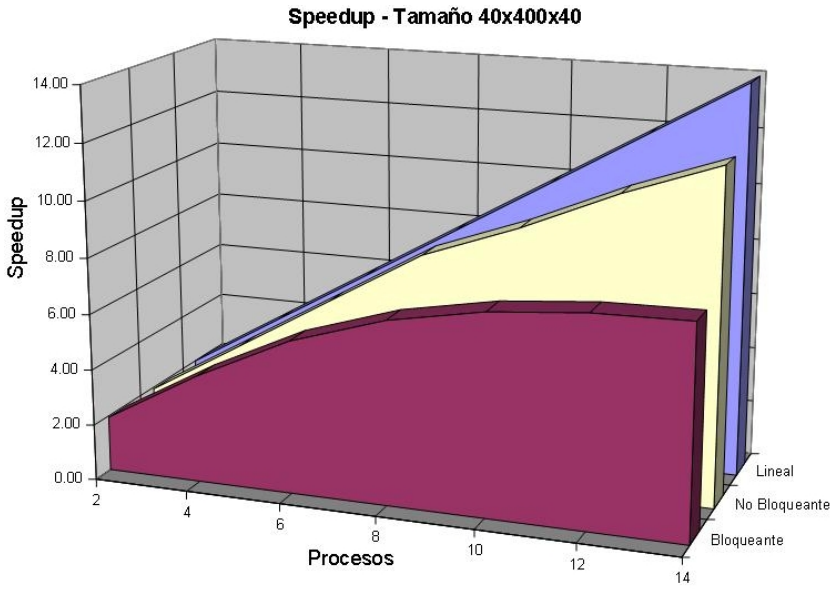

Figura 9: Speedup en una malla de $40 \times 400 \times 40$. Los valores graficados se muestran en la tabla 8.

los casos, fue una eficiencia del 22 % para el caso de malla (40x100x40) de menor tamaño con la mayor cantidad de procesos (14) y usando comunicaciones bloqueantes. En este caso, cada proceso tenía entre 7 y 8 planos para calcular y 4 para intercambiar, por lo que la relación entre superficie de cálculo y superficie de intercambio es bastante desfavorable. En cambio, para el tamaño de malla más grande, en la partición en la misma cantidad de procesadores se obtuvo una eficiencia del 54 % para el caso bloqueante. Es importante notar que, las mismas ejecuciones con comunicaciones no bloqueantes, llevaron la eficiencia del 22 % al 53 % y del 54 % al 85 % respectivamente. Para poder apreciar mejor los valores de eficiencia, se realizaron gráficos comparativos para cada tamaño de malla, entre las ejecuciones con comunicaciones bloqueantes, no bloqueantes y la máxima eficiencia teórica (eficiencia 1).

El gráfico 10(1) corresponde a la corrida de menor tamaño de malla, se observa la notable disminución de la eficiencia en el caso bloqueante, que es atenueada en gran medida en el caso no bloqueante. De todos modos, estamos muy alejados de la eficiencia máxima.

El aumento del tamaño de la malla a $40 \times 200 \times 40$ (figura 10(1)) da como resultado una eficiencia mucho mayor. El caso no bloqueante se acerca mucho más a la eficiencia del 100 % y es aún mayor la diferencia con el caso bloqueante. Esto puede deberse a que, al ser mayor el tamaño de la malla, es también mayor el tiempo de cálculo y se pueden solapar por completo las operaciones de envío y recepción de mensajes con el cálculo local.

En el gráfico 11, que corresponde a la corrida con un tamaño de malla de $40 \times 400 \times 40$, se observan, como era de esperar los mejores valores de eficiencia. El caso no bloqueante alcanza una eficiencia casi óptima hasta los 8 procesadores, reduciéndose lentamente a partir de los 10 procesadores.

No es un detalle menor que, para poder realizar una comparación cierta entre las corridas

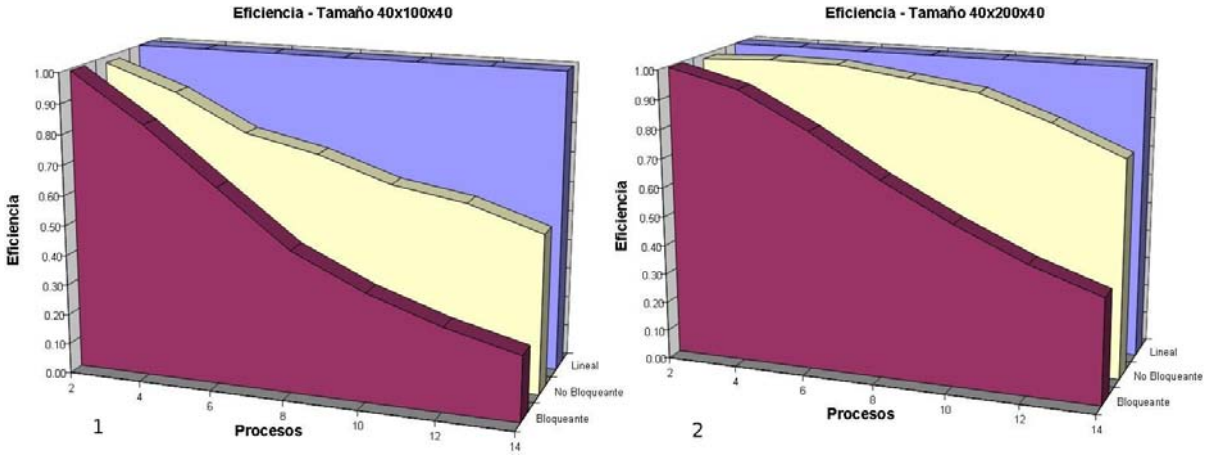

Figura 10: Eficiencia en una malla de (1) $40 \times 100 \times 40$ y (2) $40 \times 200 \times 40$. Los valores graficados se muestran en la tabla 9.

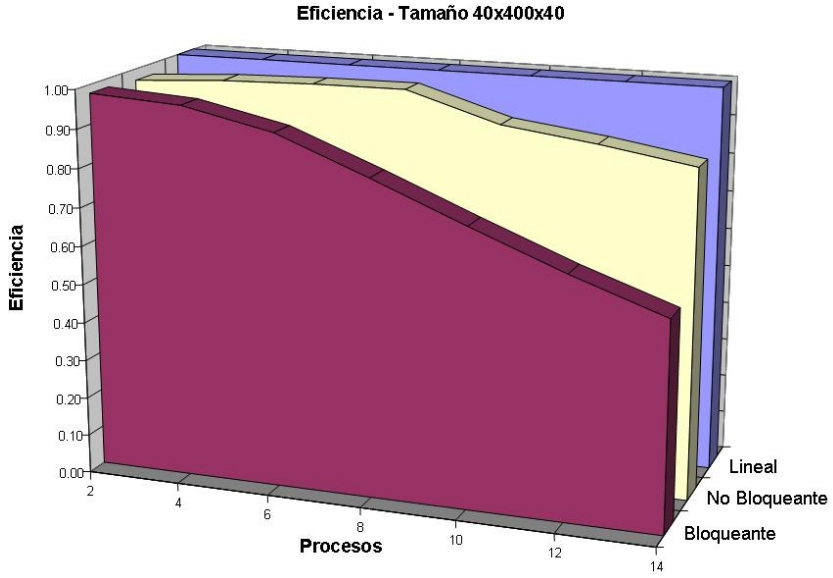

Figura 11: Eficiencia en una malla de $40 \times 400 \times 40$. Los valores graficados se muestran en la tabla 9.

serial y paralela, es requisito que el tamaño de la malla sea lo suficientemente pequeño como para poder correr en un solo procesador (un único nodo del cluster). Esto limita inherentemente el tamaño de la malla de cálculo a utilizar en estas pruebas y es la razón por la que se hicieron simulaciones con tamaño máximo de malla de $40 \times 400 \times 40$. No obstante, para brindar una noción cierta acerca de los tiempos de corrida para otros casos que quedan fuera del alcance de un único nodo de los disponibles en el cluster del LSC, se puede calcular, en base a los valores obtenidos anteriormente, una estimación de los mismos.

Supongamos entonces una ejecución sobre una malla de $120 \times 1200 \times 120$ elementos. La

proporción entre cálculo local e intercambio de mensajes es la misma que para el caso de $40 \times 400 \times 40$, con lo que podemos esperar que los valores de eficiencia y speedup sean los mismos (Speedup de 11.88 y eficiencia de 0.85 para 14 procesos utilizando comunicaciones no bloqueantes). Con estos valores, podemos estimar los tiempos de la siguiente forma:

- Ejecución secuencial: 18 días, 6 horas y 35 minutos (26.315 minutos)
- Ejecución paralela (14 procesos): 1 día, 12 horas y 55 minutos (2215 minutos)

Así, una simulación pasaría de tardar más de 2 semanas y media, a tardar 1 día y medio. Esta progresión, muestra claramente los beneficios de la paralelización, en cuanto a los tiempos necesarios para la obtención de los resultados. Sin embargo no hay que olvidar que el hecho de poder ejecutar escenarios antes inaccesibles para el poder de cálculo de una sola computadora es, en sí mismo, un logro substancial.

4. CONCLUSIONES

En este trabajo, se mostró una implementación paralela sobre un sistema Beowulf de un modelo tridimensional secuencial para la simulación numérica de un problema de ECD.

Asimismo, se introdujeron varias técnicas tendientes a su optimización:

- Balance de carga semi-dinámico para un mejor aprovechamiento de los recursos del cluster heterogéneo, sin necesidad de conocer de antemano las características del mismo.
- Alteración del orden lexicográfico de cálculo.
- Solapamiento de operaciones de cálculo con intercambio de mensajes para reducir los tiempos ociosos de comunicación y espera entre procesos
- Estudio del desempeño de las comunicaciones no bloqueantes en segundo plano en diferentes implementaciones de MPI , y de su comportamiento con diferentes cantidades y tamaños de buffer.

Estas técnicas permitieron mejoras en la performance del orden del 40 % con respecto a una paralelización no optimizada. Se estudiaron las diferencias numéricas obtenidas entre el código serial y el paralelo.

Adicionalmente, se logró en algunos casos, mediante la utilización de comunicaciones no bloqueantes, eficiencias cercanas a la ideal.

Si bien tanto el método de paralelización como el de optimización son lo suficientemente genéricos como para ser aplicados a cualquier algoritmo que resuelva problemas de cálculo numérico mediante métodos iterativos, se eligió para el desarrollo práctico un problema de electrodeposición en celdas delgadas (ECD) en 3D,[23] pero en una escala que antes estaba restringida a centros de supercomputación de países desarrollados.

REFERENCIAS

[1] T. Vicsek. *Fractal Growth Phenomena*. World Scientific, Singapore, 2nd edition, (1992).

[2] F. Argoul, J. Huth, P. Merzeau, A. Arneodo and H. L. Swinney. Experimental evidence for homoclinic chaos in an electrochemical growth process. *Physica D*, **62**(170) (1993).

[3] R. M. Brady and R. C. Ball. Fractal growth of copper electrodeposits. *Nature*, **309**, 225 (1984).

[4] V. Fleury, J. N. Chazalviel, M. Rosso, and B. Sapoval. Experimental Aspects of Electrochemical Deposition of Copper Aggregates. *Phys. Rev. A*, **44**, 6693 (1991).

[5] V. Fleury, J. N. Chazalviel and M. Rosso. Theory and Experimental Evidence of Electroconvection around Electrochemical Deposits. *Phys. Rev. Lett.*, **68**, 2492 (1992).

[6] V. Fleury, J. N. Chazalviel and M. Rosso. Coupling of Diffusion, Migration and Convection in the Vicinity of Electrochemical Deposits. *Phys. Rev. E*, **48**, 1279 (1993).

[7] V. Fleury, J. Kaufman and B. Hibbert. A Mechanism of Morphology Transition in Ramified Electrodeposition. *Nature*, **367**, 435 (1994).

[8] K. A. Linehan and J. R. de Bruyn. A Mechanism of Morphology Transition in Ramified Electrodeposition. *Can. J. Phys.*, **73**, 177 (1995).

[9] V. Fleury, M. Rosso, and J. N. Chazalviel. Electrochemical Deposition Without Supporting Electrolyte. In B. Sapoval F. Family, P. Meakin and R. Wool, editors, *MRS Symposia Proc*, Boston, (1994). Material Research Society.

[10] V. Fleury, M. Rosso and J. N. Chazalviel. Diffusion Migration and Convection in Electrochemical Deposition, Vortex Pairs, Vortex Rings Arches and Domes. In H.D. Merchant, editor, *symposium on "Structure and Properties of Deposits"*, pages 195–217, Warrendale penn, (1995). 3M (Metals Minerals and Materials Society), TMS Publications.

[11] V. Fleury and J.N. Chazalviel. 2-D and 3-D Convection During Electrochemical Deposition, Experiments and Models. In B. Sapoval F. Family, P. Meakin and R. Wool, editors, *MRS Symposia Proc*, Boston, (1994). Material Research Society.

[12] J. Huth, H. Swinney, W. McCormick, A. Kuhn and F. Argoul. Role of convection in thin-layer electrodeposition. *Phys. Rev. E*, **51**, 3444 (1995).

[13] J. R. de Bruyn. Fingering instability of gravity currents in thin-layer electrochemical deposition. *Phys. Rev. Lett.*, **74**, 4843–4846 (1995).

[14] D. Barkey, D. Watt, Z. Liu, and S. Raber. The role of induced convection in branched electrodeposit morphology selection. *J. Electrochem. Soc.*, **141**(5), 1206 (1994).

[15] C. Leger, J. Elezgaray, and F. Argoul. Experimental demonstration of diffusion-limited dynamics in electrodeposition. *Phys. Rev. Lett.*, **78**, 5010 – 5013 (June 1997).

[16] J. N. Chazalviel. Electrochemical aspects of the generation of ramified metallic electrodeposits. *Phys. Rev. A*, **42**, 7355–7367 (December 1990).

[17] G. Marshall, E. Perone, P. Tarela and P. Mocskos. A macroscopic model for growth pattern formation of ramified copper electrodeposits. *Chaos, Solitons and Fractals*, **6**, 315 (1995).

[18] G. Marshall and P. Mocskos. Growth model for ramified electrochemical deposition in the presenceof diffusion, migration, and electroconvection. *Phys. Rev. E*, **55**, 549–563

(January 1997).

[19] G. Marshall, P. Mocskos, H. L. Swinney and J. M. Huth. Buoyancy and electrically driven convection models in thin-layer electrodeposition. *Phys. Rev. E*, **59**, 2157 – 2167 (February 1999).

[20] S. Dengra, G. Marshall and F. Molina. Growth model for ramified electrochemical deposition in the presenceof diffusion, migration, and electroconvection. *J. Phys. Soc. Japan*, **69**, 963–971 (March 2000).

[21] G. Gonzalez, G. Marshall, F. V. Molina, S. Dengra and M. Rosso. Viscosity effects in thin-layer electrodeposition. *J. Electrochem. Soc.*, **148**(7), C479–C487 (July 2001).

[22] G. Gonzalez, G. Marshall, F. V. Molina and S. Dengra. Transition from gravito- to electroconvective regimes in thin-layer electrodeposition. *Phys. Rev. E*, **65**(5), 051607 (May 2002).

[23] G. Marshall, E. Mocskos, F. V. Molina and S. Dengra. Three-dimensional nature of ion transport in thin-layer electrodeposition. *Phys. Rev. E*, **68**(2), 021607 (August 2003).

[24] S. Dengra. *Experimental and theoretical studies of ion transport in thin-layer electrodeposition cells*. PhD thesis, Departamento de Computación, Universidad de Buenos Aires, (August 2004).

[25] F. V. Molina G. Marshall and A. Soba. Ion transport in thin cell electrodeposition: modelling three-ion electrolytes in dense branched morphology under constant voltage and current conditions. *Electrochimica Acta*, **50**(16–17), 3436–3445 (May 2005).

[26] G.M. Amdahl. Validity of the single-processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proceedings*, volume 30, pages 483–485, Reston. Va., (1967). AFIPS, AFIPS Press.

[27] Peter S. Pacheco. *Parallel programming with MPI*. Morgan Kaufmann Publishers Inc., (1996).

[28] John L. Gustafson. Reevaluating amdahl's law. *Commun. ACM*, **31**(5), 532–533 (February 1988).

[29] Laboratorio de sistemas complejos, departamento de computación, facultad de ciencias exactas y naturales, universidad de buenos aires. http://lsc.dc.uba.ar.

[30] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, Cambridge, MA, (1994).

[31] Vis5d, a free opengl-based volumetric visualization program for scientific datasets in 3+ dimensions. http://vis5d.sourceforge.net/.

[32] Mpich (a portable mpi implementation). http://www.mcs.anl.gov/mpi/mpich.

[33] William D. Gropp and Ewing Lusk. *Installation Guide for* `mpich`, *a Portable Implementation of MPI*. Mathematics and Computer Science Division, Argonne National Laboratory, (1996). ANL-96/5.

[34] Local area multicomputer - mpi parallel computing environment. http://www.osc.edu/lam.html.

[35] Scali mpi connect. http://www.scali.com/.